# Dealing with P2P traffic in modern networks: measurement, identification and control

Silvio Valenti

École Doctorale
d'Informatique,
Télécommunications
et Électronique de Paris

# Thèse

présentée pour obtenir le grade de docteur

de l' École Nationale Supérieure des Télécommunications

Spécialité : Informatique et Réseaux

## Silvio VALENTI

# Dealing with P2P traffic in modern networks: measurement, identification and control

# La gestion du trafic P2P dans les réseaux modernes : mesure, identification et contrôle

Soutenue le 21 septembre 2011 devant le jury composé de

| | | |
|---|---|---|
| Président - Rapporteur | Ernst BIERSACK | EURECOM |
| Rapporteur | Luca SALGARELLI | Università degli studi di Brescia |
| Examinateurs | Olivier CAPPÉ | Télécom ParisTech |
| | Timur FRIEDMAN | Université Pierre et Marie Curie |
| | Sandrine VATON | Télécom Bretagne |
| Invité | Giovanna CAROFIGLIO | Alcatel Bell Labs |
| Directeur de thèse | Dario ROSSI | Télécom ParisTech |

*Molte cose ci possono bastare, e devono bastare, nella vita:*
*l'amore, il lavoro, i soldi. Ma la voglia di conoscere non*
*basta mai, credo. Se uno ha voglia di conoscere, almeno.*
Antonio Tabucchi – Viaggi e altri viaggi

# Acknowledgments, Remerciements, Ringraziamenti

It is clear that without the support of many people my PhD thesis would not have been possible. In this small page I will try to thank all the very important people that helped me achieve this result, perhaps without even knowing.

Innanzitutto devo ringraziare Dario per avermi guidato e supportato in questa lunga avventura del dottorato e per l'incommensurabile quantità di cose che ho imparato da lui in questi tre anni. In particolare lo ringrazio per avermi trasmesso la sua passione per la ricerca, senza la quale non avrei potuto ottenere questi risultati. Ma soprattutto lo voglio ringranziare da un punto di vista umano, per il suo vulcanico entusiasmo, il suo ottimismo e umorismo che hanno allievato il peso di questo lavoro.

Thanks to all the people I have worked with, that have invaluably enriched this experience. I hope you have also enjoyed working with me as much: Luca, Giovanna, Alessandro, Marco, Michela.

Let me thank as well the members of the jury and, the rapporteurs most of all, for their effort in reviewing and attending this thesis.

Je dois remercier tous les personnes que j'ai rencontrés à Télécom ParisTech et qui ont partagé avec moi les longues heures de travail (mais surtout les innombrables pauses café, les déjeuners et les apéritifs). Merci pour tous les moments ensemble, vous avez été fondamentaux pour rendre cette expérience spécial. Dans un ordre aléatoire: Paola, Federico, Stefano, Dorice, Salma, Amy, Sameh, Mayssa, Maciej, Thomas, Xavier, Anand.

Ringrazio la banda di amici Italiani a Parigi, con i quali ho condiviso dei momenti indimenticabili in questa bellissima città, di cui senza il dottorato non avrei potuto approfittare. In un ordine altrettanto casuale: Paolo, Erika, Claudio, Gege, Max, Jessica, Marianna, Aruna, Stefano, Antonio, Davide, Matteo, Giuseppe, Nicola, Mattia. D'ailleurs c'est trop difficile de lister tous les autres personnes que j'ai rencontrées à Paris, mais un sincère remerciement va à eux aussi.

Ancora grazie agli amici di casa, perché, anche se mi sono allontanato fisicamente, sono sempre stati vicini. Li ringrazio per essere sempre lì al mio ritorno e perché so che posso sempre contare su di loro. Thanks also to who, despite my distance, lack of time or reticence, has succeeded in touching my heart.

Infine un ringraziamento alla mia famiglia che mi ha permesso di arrivare fin qui. Sebbene distanti, sono stati sempre un punto di riferimento importantissimo, senza il quale non avrei potuto affrontare gli ostacali di questo lavoro. Spero di avervi reso fieri di me tanto quanto lo sono io di voi.

For all those I have undoubtedly miss, please let me assure you that, even though I have forgotten you now, if you have been really important for me, I will remember you for the years to come.

# Résumé

## Introduction

Les applications pair-à-pair (P2P) font certainement partie des services qui génèrent la majorité du trafic sur les réseaux modernes, à cause de leur diffusion ainsi que du type des services qu'elles fournissent (par exemple partage de fichiers ou la diffusion en direct de contenu vidéo). À la fin de l'année 2007, certains études du trafic Internet, comme [92, 100], indiquaient une forte contribution des services P2P à la totalité du trafic, qui, en différentes régions du globe, était en moyenne entre le 49 et 83 percent, dépassant aussi le trafic Web. La même année a marqué aussi le début de la diffusion des applications P2P-TV, qui utilisent le paradigme P2P pour distribuer contenu vidéo en direct et qui potentiellement avaient été montrées [86] capable de générer une quantité de données très importante.

Pourtant, la situation a évolué dans une façon différente des prévisions. D'un côté des travaux plus récents montrent une prédominance du trafic web, en particulier due au vidéo (ex. Youtube, Megavideo) et au service de partage de fichiers (ex. Megaupload, Rapidshare) sur le trafic P2P; de l'autre côté certains travaux de mesure [76, 169] ont observé que le volume absolu du trafic P2P est toujours en train de croire (et va doubler en 2015 selon les prévisions) et dans quelque cas ils ont aussi remarqué une inversion de tendance, où le P2P redevient populaire entre les usagers, par exemple à cause des restrictions des certains services (Rapidshare). En outre, le P2P en Adobe Flash Player (et donc dans le browser) ainsi que les nouveaux services de diffusion de contenu basés sur le P2P sont des ultérieurs indices que le trafic P2P va garder son rôle d'importance dans les réseaux du futur.

En conséquence de leur diffusion et du volume de données généré, les applications P2P posent constamment des nouvelles challenges pour les opérateurs de télécommunications: en fait, le trafic P2P doit être bien contrôlé et managé dans le réseau, pour garantir qu'il n'endommage pas les performances des autres applications avec qui il partage la bande passante. L'objectif de cette thèse est de développer des outils et de protocoles pour supporter les opérateurs dans la gestion du trafic P2P. En particulier, nous voulons fournir des solutions pour mieux (i) *identifier*, (ii) *mesurer* et (iii) *contrôler* ce type de trafic. Dans les paragraphes suivants nous allons introduire chacune de ces thématiques.

Quoi que soit la stratégie adopté par l'opérateur pour gérer le trafic P2P, il doit d'abord *l'identifier dans une façon efficace et fiable*. Seulement avec une connaissance précise des paquets qui appartiennent aux flux P2P, les opérateurs ont la possibilité d'implémenter des mécanismes de qualité de service (QoS), pour donner une priorité inférieure au trafic P2P, puisqu'il ne dégrade pas les performances des autres services. Cependant, malgré l'effort consacré par la communauté au sujet de la *classification du trafic* [32, 34, 63, 77, 80, 83, 101, 102, 105, 119, 124, 131, 132, 132, 140, 161, 165, 184], cet problème n'a pas encore été complètement résolu. Malheureusement, les techniques classiques pour la classification du trafic, comme l'utilisation des numéros de porte de la couche transport ou l'inspection des contenus de paquets (DPI), sont devenus beaucoup moins efficaces avec les trafic P2P. Pour cette raison, des nouvelles solutions ont étés proposés, comme la *classification comportementale* [101, 102, 184], qui analyse la distribution du trafic généré par une applications: l'idée est que, comme les activités des applications diffèrent entre eux, dans la même façon le trafic généré va avoir des caractéristiques différents. Ce type de classificateurs ont des caractéristiques particulièrement intéressantes, car, en basant la classification seulement sur des données à niveau flux, comme ceux fournis par NetFlow, ils sont très légères en termes de coût de calcul et donc adaptés au gros volume de trafic des réseaux modernes. Dans la première partie de cette thèse nous allons développer un classificateur comportemental, Abacus, pensé pour les applications P2P-TV, qui est le première capable d'attribuer le trafic à une spécifique application, au lieu qu'à une famille de protocoles.

Une différente solution très utilisé par les opérateur pour gérer l'énorme quantité des données et mesures des réseaux modernes est représenté par *l'échantillonnage* à différents niveaux: dans cette façons ils réduisent le volume des données qu'ils doivent analyser et stocker. Cependant, la qualité de l'information est aussi réduite, du coup on peut se demander si cette type de données permettent toujours une bonne caractérisation et éventuellement une correcte classification du trafic. Dans la deuxième partie de cette thèse, nous cherchons de répondre à cette question, en analysant différentes techniques pour réduire le volume des données, à partir de diffèrent types d'échantillonnage (à niveau paquet et à niveau flux) avec plusieurs politiques (ex. aléatoire, systématique), jusqu'à l'use de dispositifs de mesure à niveau flux (ex. NetFlow): en particulier nous allons évaluer leur impact sur la classification statistique du trafic Internet.

Finalement, un approche complémentaire à développer des outils que les opérateur puissent utiliser pour mieux gérer le trafic P2P, est celui de modifier les protocoles eux mêmes, en les rendant plus “gentils” avec le réseau. Cet stratégie a été récemment adoptée par les développeurs de BitTorrent qui ont proposé au sein de l'IETF un nouveau protocole LEDBAT (pour Low Extra Delay BAckground Transport protocole) et qui l'ont en suite implémenté dans le client officiel. Ce protocole a l'objectif de fournir un service à basse priorité aux applications P2P, qui vont réduire la vitesse de transmission en présence d'autres protocoles traditionnels (HTTP, Mail), tout en utilisant la bande passante disponible. En particulier, le protocole cherche d'introduire un petit délai sur le goulot d'étranglement du flux (qui on assume être à l'accès) si que en particulier les performances des applications interactifs (ex. VoIP, jeu vidéo) ne soient pas impactés. Dans la dernières partie de cette thèse, nous allons évaluer ce protocole, au moyen d'analyse formelle, mesure et simulation, et nous allons aussi proposer des solutions à ses problèmes, notamment un problème d'équité dans le partage de ressources.

## Contributions

Cette thèse est divisée en trois parties, chacune consacrée à un différent aspect de la gestion et contrôle du trafic P2P. Dans cette section nous allons lister les majeurs contributions de chaque partie, qui seront après approfondies dans les sections suivantes.

La première partie est dédie à l'étude de la *classification du trafic P2P*, en particulier au moyen des algorithmes de classification comportementales, dont les avantages ont été présentés dans la section précédente. Notre première résultat dans cette domaine est la définition d'un ensemble de critère pour une exploration exhaustive de l'espace d'attributs qu'on peut définir à partir de données à niveau flux fournis par dispositifs tels que NetFlow. Par attributs nous entendions tous genres de propriétés et caractéristiques qui puissent être utilisés pour identifier l'application qui a généré un certain flux de données sur le réseau. La définitions de critères claires et précis nous permet d'obtenir un grand nombre d'attributs, un ensemble le plus complet possible, au fin de pouvoir découvrir quels sont les propriétés les plus utiles pour un classificateur. En plus, notre système est facilement extensible, dans une façons telle qu'il contient aussi les propriétés utilisées par les autres classificateurs, ce qui permet de les confronter facilement entre eux. Nous allons utiliser deux métriques pour quantifier le contenue d'informations de chaque attribut, notamment l'Information Gain et le ReliefF. Après avoir mesuré et comparé l'utilité de chaque attribut, nous allons utiliser des algorithme de classification supervisée (Decision Trees comme C4.5) pour évaluer l'efficacité de nos attributs.

En suite, nous allons utiliser la connaissance des plus importants attributs ainsi que celle du fonctionnement interne des application P2P-TV, pour définir un algorithme de classification Abacus, très légère et adapte pour cette type d'applications. Ce classificateur utilise une signature très simple pour identifier une application, qui est calculée à partir du compte du nombre des paquets et octets échangés par un hôte avec les autres pairs dans des courtes fenêtres temporelles, sans aucun accès aux données à l'intérieur des paquets. Cette simple signature est toutefois capable de capturer les particularité des différentes applications, permettant donc de les reconnaitre dans une façon efficace et fiable. La classification finale est basée sur Support Vector Machine, un algorithme de classification supervisée qui a été démontré très performant pour la classification du trafic réseau. Nous allons conduire une campagne d'expérimentation très vaste qui montre comme notre classificateur n'est pas seulement capable d'identifier avec une grande précision entre les applications, mais les signatures sont aussi portables entre diffèrent réseaux et temps. Finalement nous comparons notre solution avec un autre classificateur, Kiss [77] basé, par contre, sur l'inspection du contenu des paquets qui a été montré très efficace pour la même classe d'applications. Abacus atteint la même précision que Kiss, mais il est beaucoup plus légère en terme de coût de calcul.

Dans la deuxième partie, nous allons nous concentrer plutôt sur les techniques pour la *réduction de données*, et en particulier sur leur impact sur la qualité de la caractérisation et classification du trafic. Nous commençons par étudier le comportement d'Abacus dans certains cas plus critiques. D'abord nous testons ses performances quand il utilise que de traces NetFlow: nos expériences montrent que Abacus fonctionne correctement avec ces données, comme nous avions prévu du début de son conception, bien que les records NetFlow aient un granularité temporelle plus grande (les fenêtre temporelle de Abacus a du être élargie de 5 s jusqu'à 2 minutes). En suite, nous allons évaluer l'impact de déplacer un classificateur comme Abacus de la frontière du réseau, lieu pour le quel il était conçu, à l'intérieur dans le core, où, à cause du routage, seulement une partie des flux directs à un hôte est observé. Nos expériences montre que la classification avec Abacus reste possible et précise à condition que l'échantillonnage de flux ne soit pas biaisé et qu'il ne modifie pas la distribution des flux.

Notre évaluation continue en considérant aussi l'échantillonnage à niveau paquet, pratique très diffuse entre les opérateur pour réduire la quantité de données à traiter. Dans ce cas nous conduisons un double étude au moyen de une version modifiée de l'outil `tstat` qui nous permets de traiter de traces en appliquant différents types d'échantillonnage avec plusieurs politiques et d'extraire un grand nombre d'attributs à niveau flux. Le premier pas est de mesurer la distorsion introduite par l'échantillonnage avec de métriques statistiques: nous allons bien voire que il y a une forte distorsion même avec un faible échantillonnage, peu importe la politique adoptée. Par contre, quand nous utilisons ces données échantillonnées pour la classification, dans la deuxième partie de cet étude, nous découvrons que l'identification du trafic reste possible, à condition que le donéee utilisées pour l'apprentissage du classificateur soient échantillonnés au même taux que le données de test.

Dans la dernière partie de cette thèse nous allons nous occuper de contrôle de congestion pour les applications P2P, en particulier de LEDBAT, le nouveau protocole du client BitTorrent officiel. Dans un premier temps nous conduisons un étude de mesuré de l'implémentation officielle dans des scénarios contrôlés: nous découvrons que le protocole, malgré quelque dysfonctionnement dans les premières versions, à des bonne propriétés et se comporte correctement comme un protocole à basse priorité.

Quoiqu'ils soient très utiles pour évaluer le performance d'un protocole, les études de mesure ne sont pas suffisants: pour cette raison nous avons implémentés la spécification du protocole présenté dans le IETF draft[166] dans le simulateur de réseaux à niveau paquet `ns2` et nous avons simulé le protocole en différents scénarios. Nos simulations montrent que le protocole souffre d'un problèmes pour ce qui concerne le partage de ressource entre deux flux LEDBAT pas synchronisés: en fait, le deuxième flux a une mauvaise estimation du délai d'attente, qui le porte à être plus agressive et à s'approprier de toute la bande passante. Nous présentons quelques solutions pour ce dysfonctionnement et nous les testons toujours au moyens de simulation. Au cours de notre recherche nous avons reconduit le problème d'équité, qui afflige les protocoles basés sur le délai, à la spécification du contrôleur implémenté dans le protocole, en particulier à la décroissance additive (ce qui avait déjà été montré par Jain dans les années '80 [52]). Pour cette raison, dans notre dernière solution nous réintroduisons la décroissance multiplicative et nous allons prouver que cette modification est très efficace, au moyen d'un modelé mathématique ainsi que des simulations en scénarios qui imitent le réel fonctionnement d'un réseau P2P.

Dans les sections restants de ce résumé, nous allons résumer chaque partie de la thèse avec leur plus importants résultats.

## Classification comportementale du trafic pair-à-pair

Cette section corresponds à la première partie de la thèse et s'occupe de l'application de techniques de classification comportementale pour le trafic P2P.

### Définition d'attributs pour la classification comportementale

Nous allons résumer dans cette section les résultats présentés dans le Chap. 3 de la thèse. Dans ce chapitre, nous concevons un ensemble de critères pour la définition d'attributs pour la classification comportemental du trafic P2P. Comme nous l'avons déjà dit précédemment, la classification comportemental [80, 141, 180, 184, 184] se base sur de mesure à niveau flux du trafic généré par un hôte pendant qu'il fait tourner une application: si des bonnes propriétés sont choisies qui caractérisent bien le trafic, alors des données très simple comme ceux fournies par NetFlow sont suffisants pour obtenir un bonne précision de classification. Cependant, la recherche a produit dans ce domaine des travaux plutôt fragmentaire jusqu'à là: les solutions présentés [80, 97, 158, 173, 180] sont très hétérogènes et en plus testés sur des datasets très spécifiques.

Par contre l'ensemble des critères définis dans le Chap. 3 a le même objectif que le travail présenté en [129] pour la classification statistique: fournir un cadre de référence le plus complet possible de tous attributs pour la classification comportemental du trafic P2P qu'on puisse définir à partir seulement de données à niveau flux style NetFlow [57]. Grâce à une définition précise et complète, on peut définir un grand nombre d'attributs, qui comprennent aussi les propriétés utilisées par autres classificateur, ce qui aide à les comparer entre eux. En suite on applique deux métriques pour évaluer le contribue que chaque attribut peut donner à la classification, avec l'objectif d'identifier les attributs les plus importants. Enfin on utilise un algorithme de classification basé sur les arbres de décisions pour évaluer les performance de nos attributs.

### Critère pour la définition des attributs

Nous nous concentrons sur le trafic relatifs à un hôte $X$ sur le quel turne une application P2P dans une courte fenêtre temporelle $\Delta T$ et nous définissons les critères suivants.

- **Granularité temporelle,** la durée de la fenêtre temporelle d'observation; nous utilisons deux durées, 5 s et 120 s.
- **Entité,** que nous allons classifier qui peut être à niveau réseau (adresse IP), à niveau transport (porte TCP/UDP) ou bien la combinaisons de deux
- **Granularité spatial.** Nous pouvons mesurer le nombre de hôtes contactés, ou des paquets ou octets échangés.
- **Direction.** Nous pouvons considéré le trafic reçu ou envoyé par un hôte.
- **Catégories.** Nous pouvons partitionner l'ensemble des entités selon diffèrent critères: par exemple hôtes avec qui on fait que de la signalisation ou bien aussi de transfert de données; ou encore entre les hôtes découverts dans la dernier fenêtre temporelle et ceux qui avaient déjà été contactes précédemment.
- **Opérations.** Finalement nous pouvons appliquer plusieurs opérations mathématique ou statistique à ces données: pour simplicité nous utilisons que des simples rapport entre les données, ou biens d'opérations statistique élémentaires comme la moyenne ou l'écart type.

Enfin nous avons un ensemble d'une centaine d'attributs (la liste se trouve dans le Chap. 3), que nous allons évaluer au moyen de métriques expliqué dans la section suivante.

### Métriques pour la sélection d'attributs

Pour mesurer le contribue que chaque attribut peut apporter à la classification nous utilisons deux métriques: l'*Information Gain*, de la théorie de l'information, et le *ReliefF*, avec un interprétation plus géométrique.

L'Information Gain mesure la diminution d'incertitude que la connaissance de la valeur d'un attribut $X$ apporte sur la valeur de l'application label. Cette métrique se base sur le concept d'entropie et est normalement mesuré en bits. Le ReliefF, par contre, mesure la capacité d'une propriété de distinguer entre instances très proches entre eux dans l'espace des attributs. L'algorithme pour la calculer donc regarde les valeurs assumées par un attributs entre de groupes de points proches. Pour plus de détails nous invitons le lecteur à lire les descriptions présentées dans le Chap. 3.

### Results

Dans cette section, nous allons décrire les résultats les plus significatives obtenus pendant l'évaluation des performance des attributs définis. Nous omettons les détails regardants le dataset utilisé pour les expérimentes, qui peuvent être trouvés dans le Chap. 2; nous nous limitons à lister les applications dont le trafic a été analysé qui sont soit P2P-TV (PPlive, TVAnts, Sopcast, Joost), soit VoIP (Skype) soit partage de fichiers (eDonkey, BitTorrent).

Tout d'abord, nous avons comparé les classements d'attributs fournies par les deux métriques. Nous avons utilisé l'indexe de Jaccard [172] pour quantifier la similarité entre les sous-ensembles identifiés par les deux métriques, qui est défini comme le rapport entre la cardinalité de l'intersection

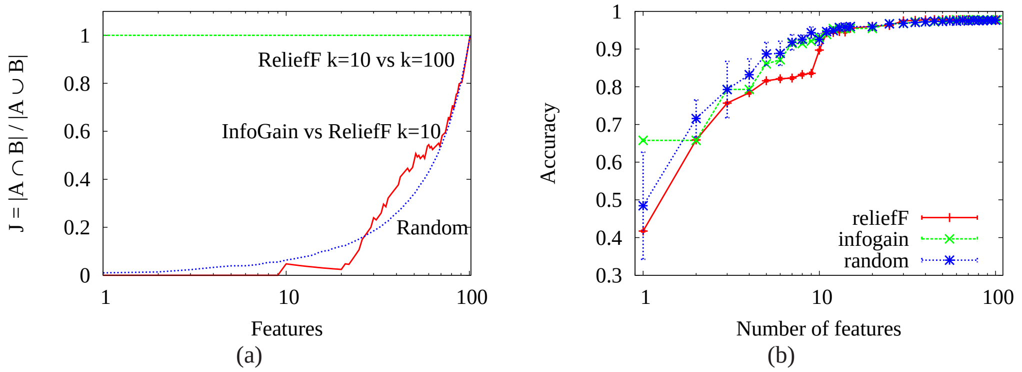


Figure 1: (a) Comparaison de l'ordre des attributs et (b) précision de la classification en fonctionne de nombre d'attributs considérés.

sur la cardinalité de l'union des deux ensembles. Fig. 1-(a) représente les valeurs assumés par l'index en fonctionne du nombre d'attributs considérés. Nous voyons que (i) le nombre de points $k$ considérés par le ReliefF n'a pas d'impact sur l'ordre qui reste pareil et que (ii) les ordres du ReliefF et Infogain sont très différents, vu que l'index a la même valeur qu'il aurait pris dans le cas où on prends des ensembles aléatoires d'attributs.

D'ailleurs, en Fig. 1 nous montrons la précision de classification obtenu en utilisant l'algorithme C4.5 en fonction du nombre d'attributs considérés, prises selon les ordre des deux métriques. Même si les ordres sont très différents entre eux, les performances de classifications convergent très vite vers des très bonne résultats: nous voyons que déjà avec une dizaine d'attributs la précision atteinte est indépendante de l'ordre choisi. En considérant tous les attributs nous sommes capables de garantir une précision du 98%.

## Abacus: un classificateur comportementale pour les applications P2P-TV

Cette section contient les résultats présentés dans Chap. 4 et Chap. 5. Après notre étude des attributs pour la classification du trafic P2P, nous nous concentrons sur les application P2P-TV, qui ont eu une large diffusions et qui génèrent déjà beaucoup de trafic, d'où l'attention reçu par la communauté de recherche. Pour cette raison nous avons conçu un algorithme de classification comportemental spécifique pour ce type d'applications, basé sur une simple mesure des données échangées par les pairs pendant des petites fenêtres temporelles. Nous allons décrire en bref l'algorithme et évaluer ses performances; finalement nous allons le comparer avec Kiss [77], un classificateur basé sur l'inspection des paquets, qui a été conçu pour le même type d'applications.

### L'algorithme de classification

Pendant les expériences présentés dans la section précédente nous avons découvert que les attributs plus importants pour la classification étaient souvent ceux qui concernaient la distribution de nombre de paquets et octets échangés avec les autres pairs. La raison de cette importance peut être facilement expliqué si nous pensons au mode de fonctionnement des applications P2P-TV. Elles font deux activités au même temps: la signalisation pour maintenir le réseau P2P et l'échange de données avec les autres pairs. Les différentes implémentations peuvent choisir leur propre façon: par exemple elle peuvent préférer contacter beaucoup de pairs avec des messages très courts, ou bien d'échanger les données les plus possible avec les même hôtes avec des longs flux.

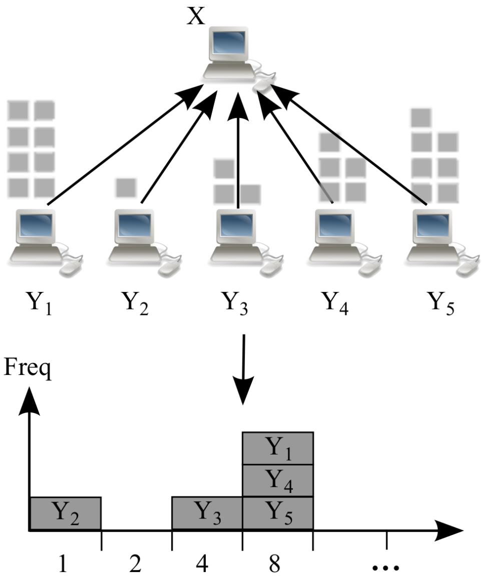


Figure 2: Procédure pour le calcul de la signature Abacus.

La signature utilisé par Abacus cherche exactement de capturer cette différence. Nous allons expliquer la procedure pour la calculer avec l'aide de la Fig. 2. Le classificateur observe le trafic reçu par un hôte $X$ dans une courte fenêtre temporelle (par défaut c'est 5 s, mais présentons aussi de résultats avec des fenêtres d'observation plus longue), pendant la quelle il conte le nombre de paquets et d'octets reçus depuis chaque pair $Y_i$. En suite, les pairs sont divisés en bins selon le numéro de paquets et d'octets envoyés: par exemple, concernant les paquets, le premier bin contient le pairs qui ont envoyé 1 paquet (comme $Y_2$ dans la figure), le deuxième le pairs qui ont envoyé 2 paquets, le troisième jusqu'à 4 paquets (comme $Y_3$ dans la figure) et si de suite avec une croissance exponentielle de numéro de paquets. Finalement, on obtient une distribution de pairs qui est exactement la signature Abacus. Dans le relatif chapitre les signatures moyennes des différentes applications sont montres et on peut remarquer des différence très évidentes entre eux.

Fig. 3 montre la procédure complète avec la quelle l'algorithme exécute la classification. D'abord il calcule les signatures pour les diffèrent applications à partir d'un training set. Ces données sont utilisés pour l'apprentissage d'un algorithme de classification supervisé très utilisé dans la classification du trafic, c'est à dire les Support Vector Machines, qui produise un modelé. Ce modelé sert pour classifier les traces de test et, en suite, les résultat sont comparés avec les label réels des applications qui ont généré le trafic. Enfin, comme le modelé peut reconnaitre seulement le trafic qui était dans le training set, l'algorithme utilise aussi un critère de rejet avec qui il peut distinguer le trafic inconnu.

### Résultats

Le relatif chapitre contient une campagne d'expérimentation très approfondie sur un dataset très étendu (décrit en détail dans le même chapitre), où nous avons étudié l'impact des différents paramètres de l'algorithme ainsi que la portabilité des signatures entre différentes conditions de réseau et temps. Dans ce résumé nous allons présenter seulement les meilleurs résultat, que nous avons obtenus avec la combinaison de la signature basé sur les paquets avec celle basé sur les octets. La précision de classification est reporté en Tab. 4.7.

La table montre que Abacus atteint une précision de classification très haute, avec 95% de

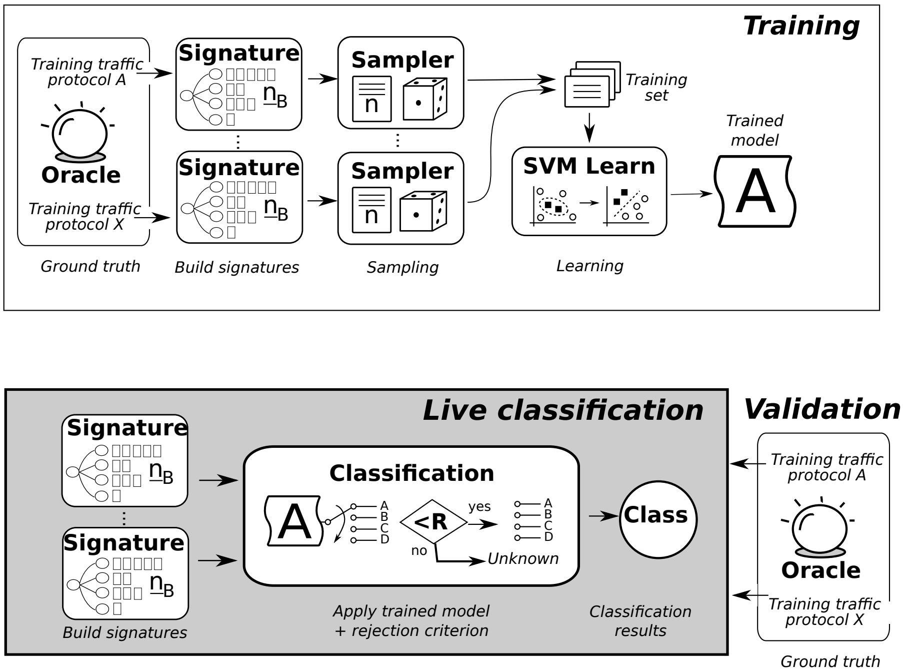


Figure 3: Procès de classification: apprentissage du model dans la partie supérieur et validation dans la partie inférieure.

Table 1: Extended Abacus Signatures: Confusion Matrix of P2P-TV Application

| | Signatures: Confusion Matrix | | | | |
|---|---|---|---|---|---|
| | PPLive | TVAnts | SopCast | Joost | Unk |
| PPLive | **95.42** | 0.22 | 1.86 | 0.36 | 2.14 |
| TVAnts | 0.06 | **99.84** | 0.10 | 0.00 | 0.00 |
| SopCast | 0.98 | 0.15 | **97.55** | 0.03 | 1.29 |
| Joost | 0.21 | 0.01 | 0.01 | **94.97** | 4.80 |

décisions correctes dans le pire des cas. Dans le relatif chapitre, nous montrons aussi l'efficacité du critère de rejet, qui est capable de reconnaitre le 98% de trafic inconnu.

### Comparaison avec l'algorithme Kiss

Nous avons vu que l'algorithme de classification comportementale Abacus est capable d'une précision de classification très haute, avec un coût de computation très contenu et aussi un taux de faux positives très faible. Cependant il y un certain scepticisme dans la communauté opérationnelle sur les algorithme comportemental, qui sont retenus pas aussi performants que les algorithmes de classification basés sur l'inspection du contenu des paquets. D'ailleurs c'est reconnu que les classificateurs de ce dernier type ont besoin de beaucoup plus de ressources de calcul.

Pour cette raison, nous avons décidé de comparer notre classificateur avec un autre basé sur l'inspection des paquets et très performant pour les applications P2P-TV aussi. L'algorithme en question s'appelle Kiss [77] et il est basé sur une inspection stochastique du contenu des paquets d'un flux. Notre choix est tombé sur cet algorithme car nous avons accès à l'implémentation, du coup nous avons pu faire tourner les deux algorithmes sur les même traces pour avoir un

Table 2: Résultats de classification

(a) Flows

Abacus

| | pp | tv | sp | jo | un |
|---|---|---|---|---|---|
| pp | **13.35** | 0.32 | - | 0.06 | 86.27 |
| tv | 0.86 | **95.67** | 0.15 | - | 3.32 |
| sp | 0.33 | 0.03 | **98.04** | 0.1 | 1.5 |
| jo | 0.06 | 2.21 | - | **81.53** | 16.2 |
| op06 | 0.1 | 0.1 | 1.03 | 0.06 | **98.71** |
| op07 | 0.21 | 0.03 | 0.87 | 0.05 | **98.84** |

Kiss

| | pp | tv | sp | jo | un | nc |
|---|---|---|---|---|---|---|
| pp | **98.8** | - | - | - | 0.2 | 1 |
| tv | - | **97.3** | - | 0.01 | 0.69 | 2 |
| sp | - | - | **98.82** | - | 0.21 | 0.97 |
| jo | - | - | - | **86.37** | 3.63 | 10 |
| op06 | - | 0.44 | 0.08 | 0.55 | **92.68** | 6.25 |
| op07 | - | 2.13 | 0.09 | 1.21 | **84.07** | 12.5 |

(b) Bytes

Abacus

| | pp | tv | sp | jo | un |
|---|---|---|---|---|---|
| pp | **99.33** | - | - | 0.11 | 0.56 |
| tv | 0.01 | **99.95** | - | - | 0.04 |
| sp | 0.01 | 0.09 | **99.85** | 0.02 | 0.03 |
| jo | | - | - | **99.98** | 0.02 |
| op06 | 1.02 | - | 0.58 | 0.55 | **97.85** |
| op07 | 3.03 | - | 0.71 | 0.25 | **96.01** |

Kiss

| | pp | tv | sp | jo | un | nc |
|---|---|---|---|---|---|---|
| pp | **99.97** | - | - | - | 0.01 | 0.02 |
| tv | - | **99.96** | - | - | 0.03 | 0.01 |
| sp | - | - | **99.98** | - | 0.01 | 0.01 |
| jo | - | - | - | **99.98** | 0.01 | 0.01 |
| op06 | - | 0.07 | - | 0.08 | **98.45** | 1.4 |
| op07 | - | 0.08 | 0.74 | 0.05 | **96.26** | 2.87 |

pp=PPLive, tv=Tvants, sp=Sopcast, jo=Joost, un=Unknown, nc=not-classified

comparaison les plus fiable possible. Il faut remarquer qu'il y a pas beaucoup de travaux de comparaison dans la littérature, en raison précisément de la difficulté des partager pas seulement les traces, mais aussi les algorithme entre équipes de recherche différentes.

Dans ce qui suit nous allons d'abord présenter l'algorithme Kiss; en suite nous allons comparer la précision de classification des deux algorithmes; finalement nous allons comparé qualitativement les algorithmes, pour comprendre pour quelles applications ils sont plus adaptés.

L'algorithme Kiss se base sur un test du $\chi^2$ appliqué sur le contenu des paquets, avec le but de identifier la syntactique du protocole parlé par l'application. En bref, l'algorithme regarde les premières 12 octets des premières 80 paquets d'un flux des données. Les octets sont divisés en groupes de 4 bits et avec le $\chi^2$ test, l'algorithme évalue le niveau d'entropie de chaque groupe de bits. Le classificateur est du coup capable d'identifier les groupes de bits qui prennent des valeur constants (identificatif), cyclique (compteurs) ou complètement aléatoires (crypté); cela correspond à identifier la syntaxe du protocole. Les valeurs de l'index statistique de chaque groupe sont utilisés comme signature pour Support Vector Machine.

Bien qu'ils aient deux approches complètement orthogonaux, Abacus et Kiss ont été montrés très efficaces pour classifier les applications P2P-TV. Pour cette raison nous les avons testé sur un ensemble des traces qui contient quatre applications de ce type et aussi sur des traces qui ne contient pas ce trafic, pour évaluer le taux de faux positives. Les détails sur le dataset sont présentés dans le Chap. 5.

Les résultats de classification en terme de pourcentage des octets et signatures correctement classifiés sont présentés en Tab. 2. Nous pouvons observer une une précisions très haute pour les deux classificateurs, surtout pour ce qui concerne les octets: celle-ci est la métrique la plus importante pour les opérateurs qui sont plutôt intéressés à l'identification des gros volumes de trafic, alors qu'ils peuvent tolérer de mal classifier des petits flux. La faible précisions en terme de signatures qu'on voit pour PPLive est due par contre à le fait que l'application utilise des sockets différents pour les différentes activités: il est très facile identifier le socket qui transmet les données, tandis qu'il est plus difficile identifier le socket pour la signalisation.

Dans la Tab. 3 nous avons reporté les plus importantes caractéristiques des deux algorithme, qui nous permettent de les comparer depuis un point de vue plus qualitatif. Comme les deux algo-

Table 3: Caractéristique principaux de Abacus et Kiss

| Characteristic | Abacus | Kiss |
|---|---|---|
| Technique | Behavioral | Stocastic Payload Inspection |
| Entity | Endpoint | Endpoint/Flow |
| Input Format | Netflow-like | Packet trace |
| Grain | Fine grained | Fine grained |
| Protocol Family | P2P-TV | Any |
| Rejection Criterion | Threshold | Train-based |
| Train set size | Big *(4000 smp.)* | Small *(300 smp.)* |
| Time Responsiveness | Deterministic *(5sec)* | Stochastic *(early 80pkts)* |
| Network Deploy | Edge | Edge/Backbone |

rithmes appartiennent à deux familles de classificateurs complètement différentes, leur propriétés le sont autant. Nous voyons tout suite que Abacus a l'avantage de pouvoir marcher avec seulement des données à niveau flux, alors que Kiss a forcement besoin des paquets, et en plus de leur contenu. Même si les deux classificateurs sont capable d'une classification fine (ils reconnaissent l'application spécifique et non pas seulement la famille de protocole), Kiss peut marcher avec n'importe quel type de protocole, tandis que Abacus dans la forme actuelle marche seulement avec les applications P2P. Les algorithme se différencient aussi pour la méthode utilisée pour identifier les protocoles inconnus, qui est basé sur une seuil dans Abacus, alors que dans Kiss on ajoute une classe spécifique pour cela dans l'apprentissage. L'apprentissage est plus longue dans Abacus, qui nécessite d'un training set plus gros, ce qui, toutefois, ne constituent pas un gros problème, vu que cette opération est normalement fait pas en ligne. Finalement Abacus a été conçu pour être déployé à la frontière du réseau, même si on verra dans les prochaines sections que cette condition peut être dans quelques façons relâchée.

Dans ce résumé, nous omettons l'étude des ressources de calcul requis par les deux algorithmes, que on peut trouver dans le Chap. 5. Nous mentionnons juste les résultats principaux, qui voient Abacus être moins chère que Kiss d'un ordre de grandeur dans le pire de cas. Dans le cas moyenne la différence peut atteindre facilement deux ou trois ordres de grandeur, ce qui fait d'Abacus le candidat parfait pour des conditions avec un gros charge de trafic.

## Techniques pour la réduction de donnés

Dans cette partie nous allons traiter l'impact sur la classification et la caractérisation du trafic des différentes techniques pour la réduction du volume de données. Nous commençons pas analyser l'impact de NetFlow et de l'échantillonnage à niveau flux sur la classification comportemental du trafic. En suite, nous étudions plutôt l'effet d'échantillonnage à niveau paquet sur la caractérisation et classification statistique du trafic.

### Échantillonnage à niveau flux et NetFlow

Dans cette section nous utilisons le classificateur Abacus, défini dans la section précédente, comme cas d'étude pour analyser le performance de la classification de trafic avec des données NetFlow, ou dans de conditions de échantillonnage à niveau flux. Nous allons traiter les deux cas séparément dans la suite.

### Abacus et NetFlow

Parmi les avantages des algorithmes de classification comportementale, le plus important est que, vu que ils n'ont pas besoin de regarder à l'intérieur des paquets, ils sont particulièrement adaptés pour utiliser des données à niveau flux, comme celles produites par exemple par NetFlow. Abacus, le classificateur comportemental qu'on a défini précédemment, serait particulièrement indiqué pour ces données, vu qu'il base la classification que sur le nombre de paquets et d'octets reçus par un hôte dans des courtes fenêtres temporelles. Nous allons verifier cette affirmation en testant Abacus sur de vraies données de type NetFlow calculées à partir des nos traces à niveau paquet.

Tout d'abord nous allons décrire le mode de fonctionnement de NetFlow, à partir duquel nous pourrons mieux comprendre les modifications qu'il faut apporter à Abacus pour qu'il puisse travailler avec ce type de données. NetFlow [56], défini originairement par Cisco et après standardisé par l'IETF sous le nom de IPFIX [57], est sans doute le standard de facto pour le monitorage à niveau flux des réseaux. Un dispositif NetFlow trace des flux de paquets, dont ils calcul certains informations agrégés. Un flux est composé par une série de paquets qui ont certains attributs en commun: normalement ils ont les même valeur pour la uplet adresses IP origine et destination, portes à niveau transport origine et destination et type de protocole à niveau transport. Cette uplet est utilisé comme index pour accéder à la table des flux, qui contient les informations pour chaque flux: identificatifs à niveau réseau et transport, timestamps de début et fin du flux, compteurs des nombre de paquets et octets, flags TCP et si de suite. Ces information sont exportées vers un NetFlow collector lorsque le flux est considéré terminé, évènement déterminé par une parmi les conditions suivantes:

- un paquet de terminaison explicite du flux est capturé (ex. TCP FIN paquet)
- le flux reste inactive pour un temps plus long que le paramètre `inactive_timeout`
- le flux reste active pour un temps plus long que le paramètre `active_timeout`
- la table des flux est pleine et il faut libérer de l'espace pour les nouveaux flux

Les valeurs standard des deux timeouts sont respectivement 15 s pour l'`inactive_timeout` et 30 min pour l'`active_timeout`. Du coup, si les données NetFlow contiennent tout ce qui sert pour calculer une signature Abacus (c'est à dire les numéros de paquets et octets reçus par un hôte depuis ses pairs), la granularité temporelle de NetFlow est beaucoup plus large des 5 s que nous avons utilisés jusqu'à là. De toutes façons, nous avons aussi testé des plus longues fenêtres temporelles dans le chapitre dédie à Abacus avec seulement une petite réduction de les performances de l'algorithme. Pour cette raison nous étions confidents que Abacus peut marcher aussi avec de véritable NetFlow records.

Pour nos expériences, nous avons quand même du modifier un petit peu la version originelle d'Abacus. Première chose, nous avons pris comme `inactive_timeout` une valeur de 120 s (le minimum possible est 60 s), que nous allons utiliser pour la durée de la fenêtre d'observation. En suite, étant donné que l'exportation des flow-records n'est pas synchrone avec la terminaison de la fenêtre temporelle, nous avons souvent des flow-records qui se terminent après la fenêtre. Par conséquent, nous devons diviser ces records entre les deux fenêtres, en assignant les paquets à chaque intervalle, proportionnellement à la durée temporelle de chaque segment. En fin, nous allons abandonner le critère de rejet que nous avions défini, et nous allons adopter un approche similaire à celui de Kiss: nous allons inclure dans le training set aussi des classe pour les autres applications P2P (eDonkey, Skype, BitTorrent) ainsi que une classe pour le trafic ”inconnu“, où

Table 4: Matrice de confusion: précision de classification pour signatures (S) and octets (O)

| | PPLive | | TVAnts | | SopCast | | Joost | | eDonkey | | BitTorrent | | Skype | | DNS | | Other | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | S | O | S | O | S | O | S | O | S | O | S | O | S | O | S | O | S | O |
| PPLive | **63.6** | **96.0** | 1.0 | 3.2 | 0.7 | 0.3 | 0.1 | - | - | - | 0.1 | 0.4 | 2.9 | - | 9.4 | - | 22.3 | - |
| TVAnts | 3.1 | 6.8 | **54.4** | **92.9** | 1.0 | 0.3 | 0.2 | - | - | - | 0.2 | - | 7.4 | - | 9.5 | - | 24.3 | - |
| SopCast | 0.7 | 0.2 | 0.4 | 0.4 | **49.7** | **99.4** | - | - | 0.1 | - | 0.3 | - | 4.8 | - | 15.9 | - | 28.1 | - |
| Joost | 0.2 | - | - | - | - | - | **53.2** | **99.9** | 0.3 | - | 0.2 | - | 4.5 | - | 19.1 | - | 22.5 | - |
| eDonkey | - | - | - | - | - | - | - | - | **94.4** | **98.9** | - | - | - | - | 0.7 | 0.2 | 4.8 | 0.9 |
| BitTorrent | 0.6 | - | 0.5 | 0.1 | 0.8 | 0.8 | 0.3 | 1.9 | - | - | **12.5** | **89.1** | 5.2 | 1.7 | 61.3 | 5.8 | 18.8 | 0.6 |
| Skype | - | - | - | - | 0.1 | 0.3 | - | - | - | - | 0.2 | 0.4 | **86.1** | **90.5** | 5.8 | 2.5 | 7.8 | 6.4 |
| DNS | 0.1 | - | - | - | 0.1 | 0.3 | - | 0.2 | 0.3 | 0.9 | - | 0.5 | 6.5 | 3.9 | **63.9** | **91.2** | 29.1 | 2.9 |
| Other | 0.1 | - | - | - | - | - | - | - | 0.4 | 0.1 | - | 0.1 | 3.5 | - | 8.3 | - | **87.6** | **99.8** |

nous allons mettre tout le trafic auquel nous sommes pas intéressés mais que nous voulons pas confondre avec les services P2P.

Dans la Tab. 4, nous présentons les résultats de nos expérimentes, en terme de précision de classification pour les signatures et pour les octets. Nous observons que la précision reste très haute en terme de octets, mais par contre il y a une diminutions si on considéré celle relative aux signatures. Pourtant, les opérateurs sont plutôt intéressés à la classification des flux avec plus d'octets, que d'après nos résultat sont tous correctement identifiés. Nous voyons que tous types d'applications P2P sont reconnu, y compris P2P-TV, partage de fichiers et VoIP. Aussi les DNS, qui, même étant client-server, présente des évidentes caractéristiques P2P, est correctement identifié, comme d'ailleurs le trafic inconnu.

### Abacus et l'échantillonnage à niveau flux

Dans cette section nous allons étudier par contre l'impact de déplacer un classificateur comportemental depuis le réseau d'accès vers l'intérieur du réseau dans le core. En fait les classificateurs comportementals ont été conçus pour être placés la où tout le trafic destiné ou généré par un hôte puisse être observé, ce qui normalement signifie le réseau d'accès. Cela est nécessaire parce que le classificateur puisse capturer le plus d'information possible pour mieux caractériser le comportement de l'application.

Au contraire, lorsque nous déplaçons le classificateur vers l'intérieur du réseau, disons au deuxième ou troisième saute router, nous voyons seulement une fraction du trafic, à cause du routage qui transmet les paquets sur des parcours différents. Pour cette raison, le classificateur va avoir une vision partielle du trafic d'un hôte, ce qui peut ne pas être suffisant pour identifier correctement l'application que l'a généré.

Dans cette section nous allons utiliser Abacus comme cas d'étude pour évaluer l'impact de ce phénomène, assimilable à un échantillonnage à niveau flux, sur la précision de la classification de trafic. Dans le Chap. 6, nous présentons d'abord une étude sur la distribution du trafic P2P sur l'espace d'adresses IP, avec le but d'identifier si le trafic est reçu d'avantage depuis certaines intervalles d'adresses, que donc il est important que le classificateur puisse observer, ou si le trafic est bien distribué sur cet espace. Nous omettons cet étude dans ce résumé, qui toutefois montre que il y a en effet une concertation du trafic in certain groupe d'adresse IP, qui apparaissent être fondamentaux pour la classification.

Le deuxième aspect que nous avons étudié est la distorsion introduite par l'échantillonnage des flux dans les signatures Abacus. Nous la pouvons observer dans la Fig. 4, où nous représentons comme les signatures moyennes de différentes application P2P changent à l'augmenter de l'échantillonnage à niveau flux. À partir de gauche nous montrons les signatures avec un taux d'échantillonnage $1/k$ avec $k \in \{1, 2, 4, 8\}$. Qualitativement, nous observons peu de changement

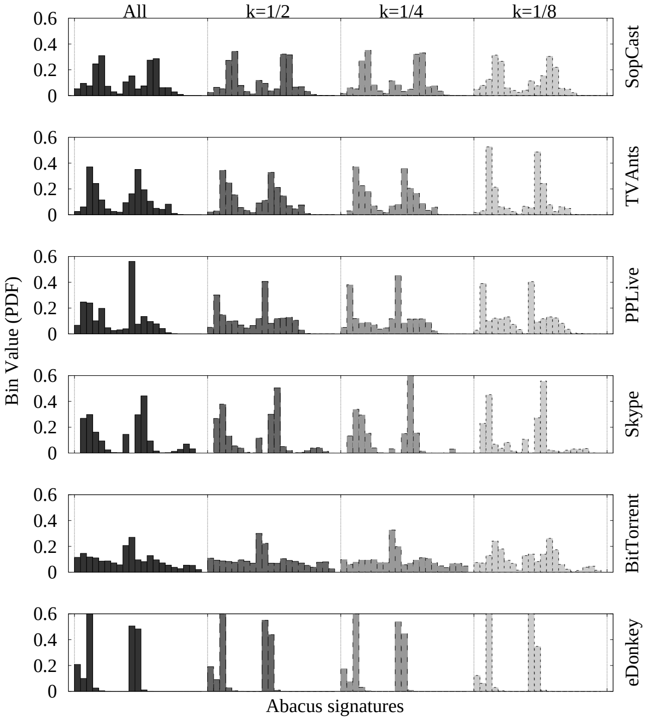


Figure 4: Signatures moyennes de chaque application pour des valeurs croissantes d'échantillonnage

dans les signatures, qui apparaissent plutôt robustes à l'échantillonnage. Cette remarque est particulièrement vrai pour de taux d'échantillonnage faibles, où les changement semblent assez peu marqués. Nous pouvons expliquer ce phénomène en pensant à comme les signature Abacus sont calculés: vu qu'elles sont normalisés sur le numéro des pairs observés, elles peuvent rester presque inchangés, à condition qu'il y ait pas de biais dans la façons ou le pairs sont échantillonnés. En fait, même avec peu de pairs, si on observe toujours soit de pairs de signalisation soit de pair de données, alors la distribution et la signature restent pareil. Par contre quand $k = 8$, il y a des applications dont la signature change visiblement (notamment PPlive et BitTorrent), ce qui pourra effectivement poser de problèmes au classificateur.

En suite nous avançons avec les expérimentes de classification. Nous allons utiliser un classificateur entrainé avec du trafic en absence d'échantillonnage pour classifier des signatures obtenues à partir du trafic échantillonné. Celle-ci est la seule solution possible, parce que l'opérateur ne peut pas connaitre à priori quel sera le taux d'échantillonnage, vu que ça dépend du routage et des conditions contingentes du réseau. Nous montrons les performances de classification dans la Fig. 5. Nous voyons que la précision de classification dégrade assez doucement jusqu'à $k = 8$, où nous avons une diminution de à peu près 30%; la précision en terme d'octets reste toujours meilleure que celle à niveau de signatures.

Les deux points qu'on observe dans la figure sont relatifs à une expérience que nous avons conduite en utilisant une table de routage réelle d'un router du backbone en Amsterdam, avec une vingtaine de liens vers des autres routers. Nous avons téléchargé le contenu de sa base de données de routage et nous avons déduit la table de routage. Avec cette information nous avons pu simuler le vrai routage (les résultats sont présentés dans le chapitre) et voire quelle seraient le performance d'Abacus si déployé sur un de router connecté à celui d'Amsterdam. Nous voyons

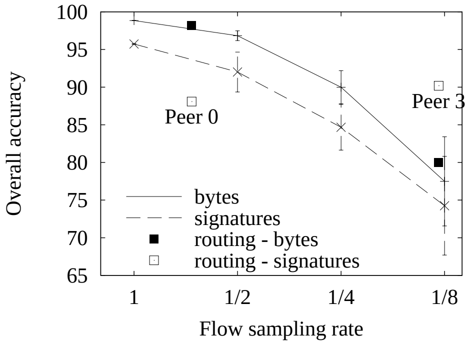


Figure 5: Précision de classification en fonctionne du taux d’échantillonnage.

que si le classificateur avait été placé sur le pair 0, alors, vu que la majorité du trafic est transmis sur ce lien, la précisions de classification serait aussi très haute. Pour le pair 3, le résultat sont encore mieux que prévu, parce que la précision est plus haute de celle que nous avons obtenu avec le même taux d’échantillonnage aléatoire.

## Échantillonnage de paquet et classification du trafic

Dans cette section nous allons nous concentrer plutôt sur l’échantillonnage de paquets et sur son impact sur la classification statistique du trafic. Cette pratique est très commune parmi les opérateurs, parce que elle permet tout de suite de réduire la charge sur le dispositif de monitorage du trafic déployé, vu que seulement une fraction des paquets est analysée. Cependant, à cette réduction corresponde une aussi important réduction d’information, qui peut rendre certaines activités de management du réseau difficiles, si non impossibles. Mais, comme l’échantillonnage devient de plus en plus obligatoire, nous nous demandons si la classification de trafic reste toujours possible même dans ces conditions.

L’importance de ce sujet est démontré par les nombreux travails de chercher, qui soit évaluent les performances de politiques d’échantillonnage particuliers [28, 49, 55, 67, 69, 109, 133, 146], soit mesurent l’impact sur certaines activités de management du réseau, comme monitorage, détection d’anomalies et classification du trafic [42, 43, 88, 97, 120, 137, 188]; pourtant, l’impact de l’échantillonnage sur la classification du trafic est un sujet pas trop recherché malgré l’importance de cette activité dans les réseaux modernes.

En raison de cette manque, nous avons conduit un double étude sur l’effet de l’échantillonnage de paquets. Préliminairement nous avons modifié un logiciel pour le monitorage des flux sur le réseau `tstat` [16], qui nous permet de calculer nombreuses quantités à niveau flux, auxquels nous pouvons accéder soit singularisent soit dans une façons agrégé comme distributions des valeurs sur l’ensemble des flux observés. Nous avons ajouté au logiciel la possibilité d’appliquer des différentes politique d’échantillonnage avec des taux arbitraires: de cette manière nous pouvons répéter les expérimentes sur le même ensemble de traces avec plusieurs type d’échantillonnage.

La première partie de notre étude est dédiés à analyser la dégradation introduit par l’échantillonnage dans le mesure des attributs à niveau flux du trafic, dans une façons indépendante de l’application possible de cette mesure. L’avantage de cet approche est que nous allons isoler l’effet de l’échantillonnage, alors que souvent les autres travaux ont considéré plutôt les performances des activité de management avec des données échantillonnées. Pour ce type d’analyse nous allons utiliser les distributions des valeurs des attributs: en particulier nous allons mesurer la distance entre la distribution originelle et celle échantillonnée au moyen de la distance d’Hellinger qui est spécifique pour ce type

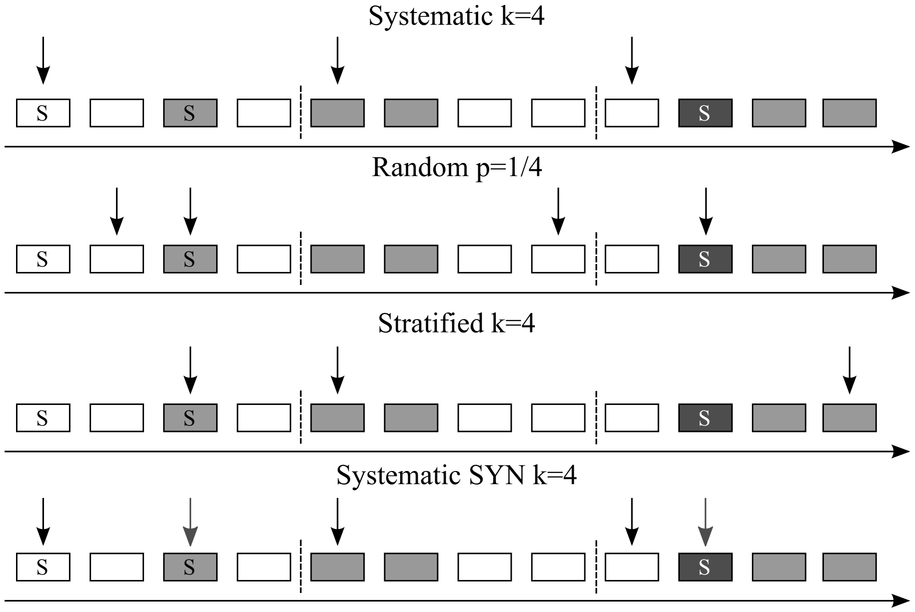


Figure 6: Illustration des politiques d'échantillonnage utilisées dans nos expérimentes.

de comparaison (dans le Chap. 7 nous utilisons aussi le Fleiss $\chi^2$ [55] pour nous confronter avec les études précédents).

La deuxième partie, par contre, se concentre sur la classification du trafic avec des données échantillonnées et, donc, utilise les valeurs des attributs pour chaque flux. D'abord nous étudions le contenu d'information des attributs avec la métrique de l'Information Gain, que nous avons déjà rencontré précédemment, avec laquelle nous allons mesurer comment la quantité d'information dégrade à l'augmenter de l'échantillonnage. Finalement nous allons utiliser les données échantillonnées pour la classification, avec un algorithme basé sur les arbres de décisions, et nous allons tester deux politique d'apprentissage qui donnent des résultats complètement différentes.

Tout d'abord, nous allons décrire les politique d'échantillonnage que nous allons utiliser dans nos expérimentes. Elles sont expliquées dans la suite et représentées dans la Fig. 6.

- **Systematic sampling**: les paquets sont échantillonnés dans une façon déterministe, un chaque $k$ paquets. Dans l'exemple, pour chaque fenêtre de 4 paquet, le premier est toujours choisi.

- **Random sampling**: les paquets sont choisis aléatoirement, en particulier chaque paquet est échantillonné avec une probabilité indépendante $p = 1/k$. Dans l'exemple, nous voyons que, le procès étant aléatoire, les paquets peuvent être choisis en séquence.

- **Stratified sampling**: $k$ paquets consécutifs sont groupés dans une fenêtre, dans laquelle un est échantillonné au hasard. Dans la figure, on peut voire que, à différence du systematic sampling, le processus ne sélectionne pas toujours le premier paquet, mais aléatoirement un parmi les 4.

- **Systematic SYN sampling**: est la superposition de deux processus indépendants: (i) un processus de systematic sampling, qui sélectionne un paquet chaque $k$; (ii) un processus qui sélectionne tous les paquets TCP avec le flag SYN actif. Dans la figure nous remarquons que tous les paquets déjà pris par le systematic sampling sont pris, plus tous les SYN paquets.

Les premières trois politiques sont très simples et sont normalement implémentés dans les dispositifs de monitorages déployés par les opérateurs, alors que la dernière est ce qu'on appelle une politique "intelligente", parce que elle est plus compliqué pour capturer les informations les plus importants.

Après le politique d'échantillonnage, nous décrivons en bref les attributs à niveau flux mesurés par `tstat`. Comme déjà dit il exporté ces mesures dans deux façons, pour chaque flux ou comme distribution des valeurs de tous les flux observés. Les propriétés observés appartiennent à différents niveaux de la pile protocolaire, du niveau réseau et transport jusqu'à des propriétés de niveaux applicatif. La liste complète se trouve dans l'Appendix B et un résumé est présenté dans le Chap. 7. En total nous avons 172 attributs agrégés et 91 attributs pour flux.

Nous devons aussi dépenser quelque mots sur le dataset utilisé pour les expériences. Il est composé par 4 traces, capturé dans des environnent très hétérogènes et espacé dans le temps, depuis un opérateur téléphonique italien et trois campus universitaires, un desquels est un lien wifi. Tous les détails des traces sont reporté dans le chapitre relatif, y compris la composition en terme de protocoles que est importante pour nos expériences de classification.

Dans le Chap. 7 nous présentons une analyse par couche protocolaire de la dégradation introduite par l'échantillonnage dans les attributs agrégés, que nous omettons dans ce résumé. Le résultat plus important était que les propriétés pour lesquelles il suffit de regarder un seul paquet du flux sont le normalement moins dégradées par l'échantillonnage, tandis que celles qui dépendent de l'inspection de plusieurs paquets sont plus touchés par l'échantillonnage, même si très faible. D'ailleurs nous avons remarqué que les propriétés qui peuvent être mesurées seulement si un spécifique paquet est échantillonné (par exemple les options TCP qui ont trouve dans le tout premier paquet de la connexion), sont souvent impossible à estimer, vu que la probabilité de sélectionner ce paquet est très faible avec l'échantillonnage.

Après cette analyse nous avons sélectionné un groupe d'attributs "robustes", pour lesquels la dégradation reste contenue au dessous d'une seuil, pour isoler l'effet du taux d'échantillonnage et observer plutôt l'effet de la politique d'échantillonnage. Les résultats sont montrés dans la Fig. 7, où chaque graphe correspond à une politique d'échantillonnage et chaque ligne à une trace différente. Chaque point report la moyenne de la dégradation sur le groupe d'attributs robustes pour le taux d'échantillonnage correspondant à l'abscisse. Nous remarquons une grande similarité entre les graphes de toutes les politique simples (systematic, random, stratified), qui contredit partialement des résultats de la littérature précédente qui montrait un avantage de politique d'échantillonnage stratifié. Notre intuition est que, vu le multiplexage statistique du trafic dans les réseaux modernes, la politique d'échantillonnage a un impact plus modéré, surtout quand des propriétés plutôt compliquées sont considérées. Au contraire, notre politique intelligente modifie dans une façon pas négligeable les courbes, parce que l'effet de sélectionner tous les SYN paquets change complément les distributions (par exemple déjà tous les flux sont pris, ce qui va être important pour la prochaine partie sur la classification du trafic). Cet effet est tant plus évident pour les traces avec une grande quantité des petits flux.

Âpres cette analyse de l'effet de l'échantillonnage, qui, comme vous avez vu, ne se concentre pas sur aucune application particulière des données, nous allons se concentrer plutôt sur la classification du trafic réseau. Pour faire cela, nous allons changer de prospective et nous allons abandonner les distribution des attributs pour regarder singulièrement chaque flux avec ses propres valeurs. En suivant un approche similaire à celui que nous avons adopté pour l'étude des les attributs pour la classification comportemental, tout d'abord nous allons mesurer le contenu d'information sur l'application de chaque attribut. Nous utilisons la même métrique que avant, c'est à dire l'Information Gain. En plus, nous allons utiliser toujours la politique Systematic SYN sampling, car elle nous permet de capture beaucoup d'information exploitable pour la classifi-

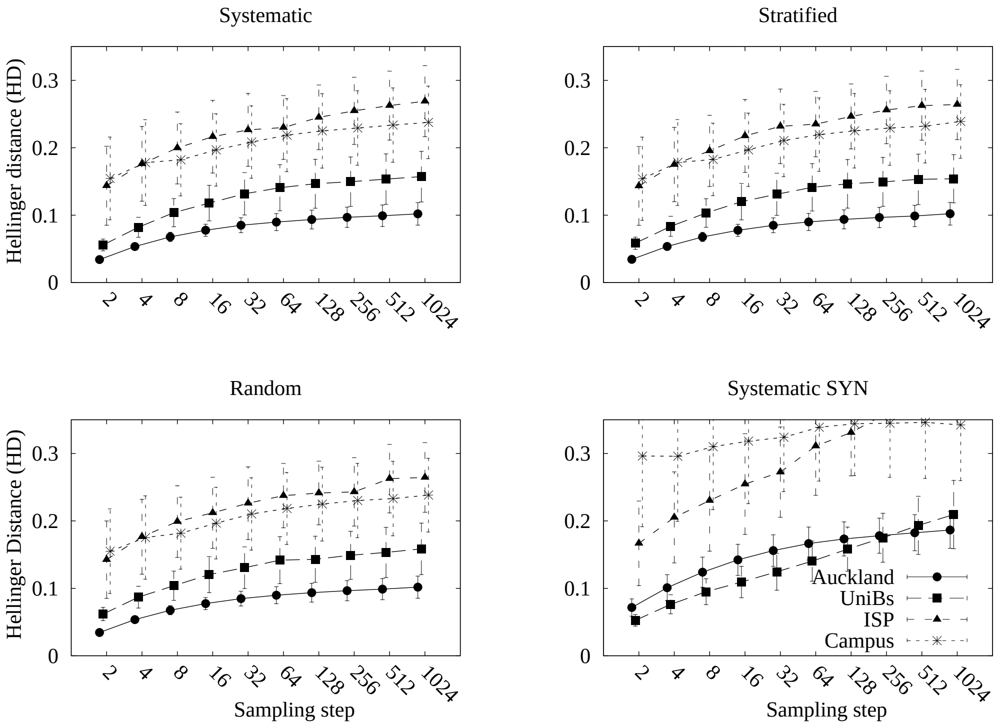


Figure 7: Moyenne et variance de la distance de Hellinger pour le groupe des attributs robustes en fonctionne du pas d'échantillonnage pour les différentes politiques.

Table 5: Information gain des attributs pour de taux d'échantillonnage différents.

| Features | Unsampled | | Sampled k=2 | | Sampled k=10 | |
|---|---|---|---|---|---|---|
| | Score | Rank | Score | Rank | Score | Rank |
| Server-IP-address | **1.68** | 1 | **1.68** | 1 | **1.68** | 1 |
| cwin-min-c2s | **1.49** | 2 | **1.20** | 6 | 0.60 | 14 |
| min-seg-size-c2s | **1.48** | 3 | **1.22** | 5 | 0.47 | 23 |
| cwin-max-c2s | **1.47** | 4 | **1.11** | 8 | 0.56 | 15 |
| max-seg-size-c2s | **1.43** | 5 | **1.17** | 7 | 0.46 | 24 |
| initial-cwin-c2s | **1.41** | 6 | 0.71 | 26 | 0.29 | 32 |
| First-time | **1.37** | 7 | **1.37** | 2 | **1.37** | 2 |
| cwin-min-s2c | **1.35** | 8 | **1.06** | 11 | 0.53 | 16 |
| Server-TCP-port | **1.34** | 9 | **1.34** | 3 | **1.34** | 3 |
| initial-cwin-s2c | **1.33** | 10 | 0.77 | 22 | 0.30 | 31 |
| Client-IP-address | **1.31** | 11 | **1.31** | 4 | **1.31** | 4 |
| cwin-max-s2c | **1.28** | 12 | 0.99 | 14 | 0.49 | 21 |
| min-seg-size-s2c | **1.22** | 13 | 0.96 | 16 | 0.51 | 19 |
| max-seg-size-s2c | **1.21** | 14 | **1.03** | 12 | 0.50 | 20 |
| Last-time | **1.14** | 15 | **1.09** | 9 | **1.02** | 5 |
| win-max-s2c | **1.08** | 16 | **1.07** | 10 | 0.98 | 6 |
| Completion-time | **1.03** | 17 | 0.97 | 15 | 0.42 | 25 |
| win-min-s2c | **1.02** | 18 | **1.01** | 13 | 0.94 | 7 |
| unique-byte-s2c | **1.02** | 19 | 0.74 | 23 | 0.42 | 27 |
| data-byte-s2c | **1.01** | 20 | 0.74 | 24 | 0.42 | 26 |

cation (les SYN paquets sont particulièrement important sous cet aspect, comme montré aussi dans [146]).

Dans la Tab. 5, nous listons le premiers dix attributs en ordre d'Information Gain décroissant, pour un échantillonnage SYN systematique avec $k = 1, 2, 10$. Cet ordonnancement nous permet

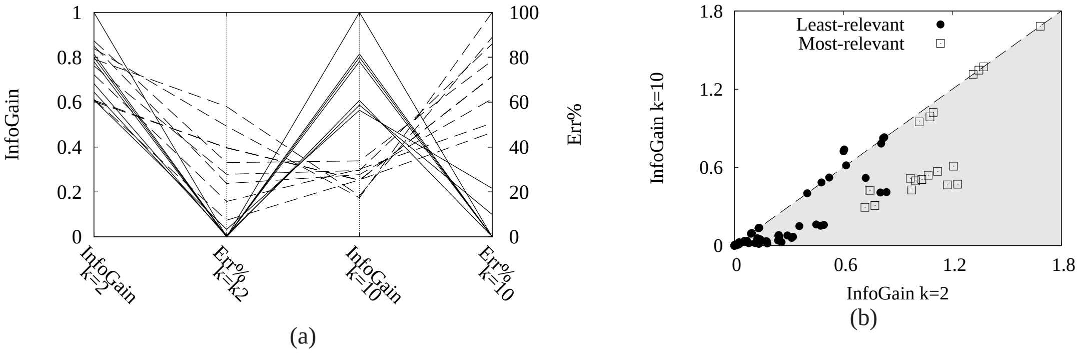


Figure 8: (a) Parallel coordinates graph pour les attributs plus rilevants et (b) scatter plot pour l'information gain de tous les attrbitus avec $k = 2, 10$.

de partager les attributs en deux catégories avec un seuil: tous ceux qui ont un Information Gain supérieur à un bit font partie des plus importants, les autres de le moins importants. Dans le Chap. 7, nous présentons un étude complet de la précision de classification de diffèrent ensembles des features: le plus important résultat est qu'il n'y a pas trop de différence entre les ensemble, ni aucun effet d'overfitting, mais les attributs les plus relevant offrent quand même le meilleur compromis entre la numérosité de l'ensemble et la précision de la classification. Ce qu'on peut observer dans la table, par contre, est que le contenu d'information dégrade assez doucement avec l'échantillonnage, surtout pour les attributs qu'on a vu être plus robustes, c'est à dire ceux qu'on peut estimer avec l'inspection d'un seul paquet. Notamment l'adresse du serveur apparait comme un discriminâtes assez puissant, peu importe le taux d'échantillonnage: ce phénomène est dû aussi aux caractéristique de la trace de l'Université de Brescia, utilisée pour cette évaluation, qui est en partie artificielle.

Dans la Fig. 8 nous avons 2 représentations différentes pour visualiser l'impact de l'échantillonnage sur l'information Gain. Dans la figure de gauche nous utilisons un graphe à coordonnée parallèles pour représenter l'effet d'échantillonnage sur les attributs plus relevant. Nous voyons qu'il y a deux comportements essentiellement. Les propriétés dénotées par une ligne continue sont les attributs mesuré au moyen d'un seul paquets: ils ont un petit distorsion en terme de erreur relatif, en gardant toujours un contenu d'information élevé. Par contre, les attributs dénotés par une ligne pointillée ont une dégradation plus marqué, vu l'augmenter de l'erreur relatif, mais ils font partie des attributs plus relevants parce que le contenu d'information est pas trop réduit, malgré l'échantillonnage. Dans le plot de droite, nous représentons le contenu d'information avec un scatter plot, où on a les valeur pour $k = 2$ sur l'abscisse et pour $k = 10$ sur l'ordonne; nous utilisons deux types de points différents pour les attributs plus et moins relevant respectivement. La figure confirme que l'ordonnancement des attributs n'est pas du tout stable, vu que pour certains l'information gain est plus grand avec un pas d'échantillonnage plus grand. En fait, nous voyons aussi un grand nombre des propriétés que nous avons classées comme moins importantes, qui tombe sur la bissectrice et qui donc devraient plutôt être considérés comme des bonnes discriminateurs. Pour cette raison, l'utilisation de l'ensemble d'attributs complet semble la stratégie meilleure pour obtenir de bonnes prestations de classification, à conditions que le classificateur soit capable de gérer tous cette information sans subir le phénomène de l'overfitting.

Finalement nous effectuons la classification du trafic avec les attributs des flux produits par `tstat`. Nous comparons deux politiques possibles pour l'apprentissage de l'algorithme de classification: nous appelons apprentissage homogène le cas où nous utilisons des données échantillon-

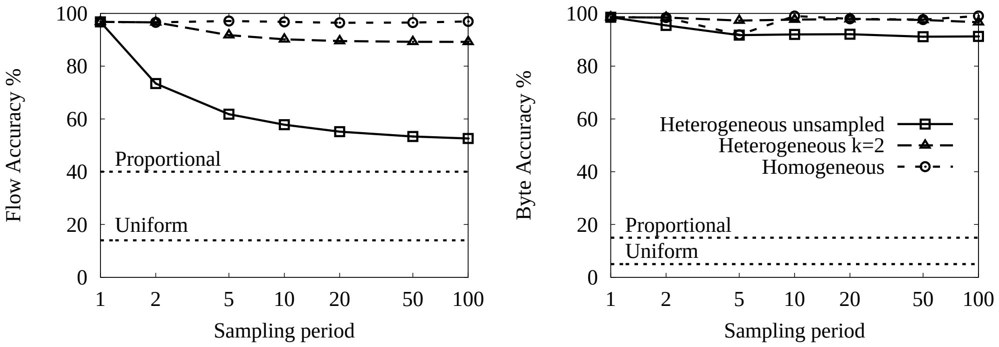


Figure 9: Impact de politique de apprentissage homogène et hétérogène en fonction du taux d'échantillonnage en terme de flux et octets.

nées au même taux pour training et test, alors que l'apprentissage hétérogène correspond à tous les autres cas. Dans la Fig. 9 nous observons la précisions de classification en terme de flux (gauche) et octets (droite) pour les deux politiques d'apprentissage en fonctionne du taux d'échantillonnage du test set. Le deux lignes labelles Uniform et Proportional représentent la précision qu'on aurait si on utilisé deux procès de classification naïves ou nous classifions les flux aléatoirement avec un probabilité uniforme entre le label (Uniform) ou avec un probabilité proportionnelle au nombre de flux dans le training set avec chaque label (Proportional). Nous voyons tout suite que la politique homogène a les performance meilleure, avec une bonne précision de classification même pour de taux d'échantillonnage importants, tandis que la politique hétérogène apprends seulement la distribution de flux, vu que sa précision est en ligne avec le processus Proportionnel.

## Control de congestion pour les application P2P

Dans la dernière partie de cette thèse nous changeons de point de vu: au lieu de développer des nouveaux outils pour les opérateur, qui leur permettent de mieux gérer leur réseaux, nous nous concentrons sur les applications P2P et sur leur protocoles en les modifiant pour les rendre plus gentils vers le réseau ainsi que vers les autres applications. Avec le même esprit, les développeur de BitTorrent ont récemment proposé, et implémentée dans la version officielle du logiciel, un nouveau protocole de niveau transport, LEDBAT pour Low Extra Delay BAckground Transport protocole, en cours de standardisation aussi chez l'IETF [166], qui a exactement le même objectif.

Depuis sa naissance, LEDBAT a été le sujet de grandes discussions: surtout le fait qu'il se base sur UDP et qu'il soit utilisé par les grands volumes de trafic générés par BitTorrent a fait penser que sa diffusion pouvait causer une nouvelle congestion globale d'Internet. Mais nous verrons que le protocole implémente en réalité un contrôle de congestion très efficace à niveau applicatif, qui fait de lui un protocole à basse priorité, plus prudent que TCP même. Pour souligner l'importance de ce protocole, nous montrons en Fig. 10 des mesures dans un réseau d'un opérateur réel, où nous pouvons voire sa diffusion. À partir du mois de Mars 2010, quand LEDBAT devient le défaut dans la version officielle du logiciel BitTorrent nous voyons que la pourcentage de trafic BitTorrent UDP augmente énormément, jusqu'à représenter à peu près le 50% de la totalité des données échangés sur le réseau. Nous voyons aussi que la pourcentage de trafic BitTorrent sur le réseau montre une légère inflexion, mais cette tendance est en train de changer comme montré par des études plus récentes [76].

Dans le sections suivantes, nous allons présenter nos études concernant ce nouveau pro-

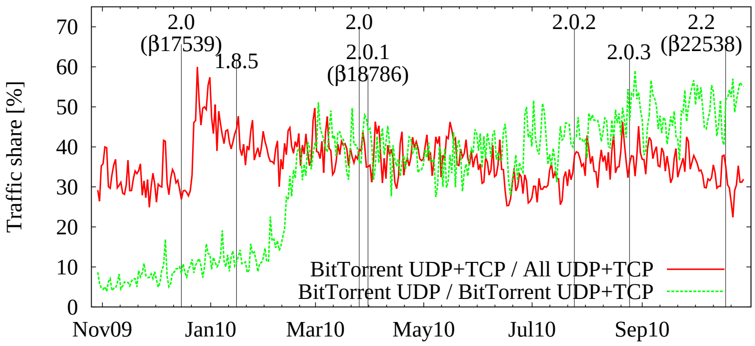


Figure 10: Proportion du trafic BitTorrent et BitTorrent Ledbat dans un réseau réel.

tocole. Tout d'abord nous avons conduit un étude de mesure du protocole, en particulier de l'implémentation dans le client BitTorrent officiel. Cet étude est antécédent la publication du draft du protocole, du coup l'implémentation était inconnu. Après la divulgation du draft, en connaissant les spécifiques du protocole, nous avons pu l'implémenter dans un simulateur à niveau paquet (`ns2`) et l'étudier au moyen de simulation. Notre analyse montre un problème de partage des ressource quand deux flux LEDBAT pas synchronisés insistent sur le même lien: nous recherchons la cause de cette iniquité et proposons des solutions efficaces.

## LEDBAT le nouveau protocole de BitTorrent

Comme déjà dit précédemment, notre premier étude du protocole LEDBAT est basé sur des mesures, d'abord sur un testbed actif et en suite sur Internet. La raison de cette méthodologie était que dans un premier temps la spécification du protocole n'était pas connue: les développeurs de BitTorrent avait annoncé le nouveau protocole basé sur UDP avec l'objectif de fournir un service à basse priorité. Cependant, ils n'avaient pas publié aucun description du protocole même, seulement il l'avait implémente dans la version bêta du logiciel. A ce point là donc, le seul moyen d'étudier le protocole était avec un approche black box, ce que nous avons adopté.

D'après les déclaration de BitTorrent, le protocole devrait utiliser une mesure du délai comme signal de congestion, style TCP Vegas [41]. Le but était d'introduire seulement un petit délai additionnel dans la file d'attente du goulot d'étranglement, qui les développeurs soutiennes être à l'accès du réseau. L'avantage de cela est averti surtout par les applications interactives (VoIP, jeu-video) qui souffre par des grandes délais. En plus, en monitorant le délai, le protocole peut s'apercevoir vite d'une congestion imminent et réduire d'avantage son débit pour ne pas endommager les autre protocole sur le même lien. Notre étude cherche donc de vérifier si l'implémentation dans le logiciels se conforme à l'annonce de BitTorrent et au même temps d'évaluer ses performance qui sont plus intéressantes pour les usagers.

Vu que le protocole était toujours en cours de perfectionnement, surtout du cote implémentation, nous avons cherché de suivre son évolution, en répétant les mêmes expériences pour différentes versions successives du logiciel, comme ça nous avons pu apprécier les modifications et les améliorations introduites à chaque fois. Pour effectuer nos expériences, nous avons utilisé un testbed actif: nous connectons le client qui turne sur diffèrent machines sur un router Linux sur le quelle nous utilisons `netem` pour émuler des conditions de réseau (délai et débit) arbitraires. Dans cette façons, nous pouvons facilement étudier le comportement du protocole dans différents situations ainsi que observer comment il s'adapte à conditions difficiles qui changent beaucoup.

Dans notre première expérience nous verrons comment le protocole utilise la bande passante

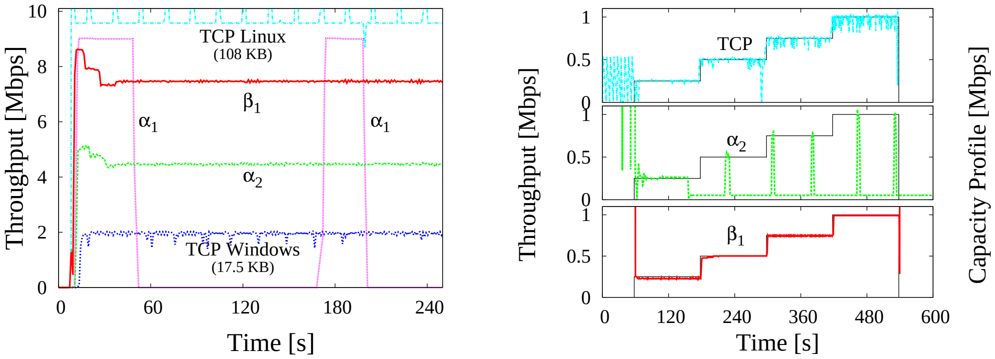


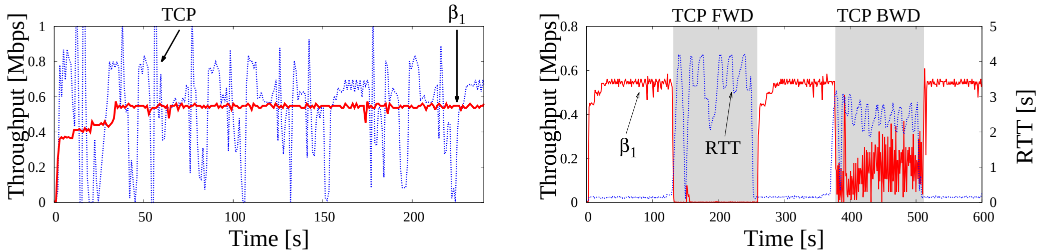
Figure 11: Débit pour différentes versions du protocole, (a) sans et (b) avec limitation de capacité.

disponible. En Fig. 11-(a) nous montrons le débit de différentes versions du protocole ($\alpha_1$, $\alpha_2$, $\beta_1$qui correspondent à deux alpha versions de LEDBAT et à la première stable beta) quand elles sont toutes seules sur un lien à 10 Mbps (chaque courbe correspond à une expérience différente, mais on surimpose les courbes pour un meilleur comparaison). Pour référence, nous montrons aussi le débit atteint par TCP classique, dans deux implémentations: celle de Linux et celle de MS Windows. Nous observons tout suite un dysfonctionnement dans la première version, qui n'est pas capable d'avoir un débit constant; l'erreur a été correct dans $\alpha_2$, mais le protocole n'est pas encore capable de profiter de toute la bande disponible, dû probablement à une mauvais valeur d'un paramètre; heureusement cela a été fixé dans $\beta_1$ qui ne souffre plus de ce problème. Si on compare avec les deux TCP, nous voyons que TCP Windows n'arrive pas non plus à bien utiliser la capacité du lien, à cause d'une valeur maximale de la fenêtre de réception trop petite (nous précisons que cette observation est valable pour Windows XP Home édition; dans les versions plus récentes une valeur plus adaptée aux réseaux moderne permet de profiter de toute la capacité), alors que la version Linux utilise correctement toute la capacité disponible.

Dans la Fig. 11, par contre, nous changeons la capacité du lien, en augmentant à des intervalles réguliers la bande disponible depuis 0 jusqu'à 1 Mbps avec des pas successives de 250 Kbps. Encore un fois, nous pouvons remarquer le travail des développeurs: si la première $\alpha_1$ n'arrive pas à s'adapter aux conditions de capacité variable, $\beta_1$ est tout à fait très efficient à égaler la capacité disponible, comme la plus mature implémentation TCP. En plus, étant basé sur le délai, nous pouvons remarquer aussi une meilleure stabilité, par rapport à les classiques oscillations de TCP autour de le débit moyen. Nous pouvons remarquer le même phénomène dans la Fig. 12-(a), où nous observons le débit atteint par un flux TCP et un flux LEDBAT sur un lien ADSL sur Internet. Le deux courbes correspondent de nouveau à deux expériences différentes et sont surimposé pour mieux les comparer. La plus importante observation est que le débit de LEDBAT et beaucoup plus stable, alors que TCP présent des oscillations évidentes, qui sont dûs à le contrôle de congestion basé sur les pertes.

Dans le Chap. 8, nous conduisons un étude similaire mais en changeant le délai sur le lien (qui dans les précédents expériences était fixé sur 50 s). Notre résultat plus important est que le délai sur le parcours de retour, qui ne devrait pas avoir aucune influence sur LEDBAT vu qu'il mesure seulement le délai d'aller, à en fait un impact pas négligeable parce qu'il ajoute du retard dans la boucle du contrôleur LEDBAT résultant dans un débit plus instable.

Finalement en Fig. 12-(b) nous étudions le comportement de LEDBAT quand il compète avec un flux TCP sur le même goulot d'étranglement. L'expérience à été fait encore sur un lien ADSL commercial sur Internet. Nous montrons le débit atteint par le flux LEDBAT, ligne rouge, et le RTT mesuré au même temps avec `ping`. Dans les deux périodes de temps démarqué par la zone grise,



(a) (b)

Figure 12: Expériences sur Internet: (a) versions différentes et (b) trafic d'interférence.

nous avons lancé un flux TCP concurrent, le première dans la même direction du flux LEDBAT et le deuxième dans le sens inverse. LEDBAT profite correctement de toute la bande disponible quand il est seule sur le lien et il réduit le débit correctement pour laisser la priorité au flux TCP dans le même sens. Quand il y a le flux dans le sens inverse, pourtant, le comportement est divers de ce qu'on attendait: LEDBAT est pas du tout insensible à cela, et l'effet d'un retard additionnel ainsi que le trafic de ACK sur le parcours de retour impact fortement sur le débit.

## Étude de simulation de LEDBAT

Dans cette section nous allons étudier le protocole LEDBAT au moyen de simulation, cela étant possible à partir du moment où la spécification de l'algorithme de congestion contrôle à été publié dans le draft chez IETF [166].

Tout d'abord nous allons décrire l'algorithme même, dans la forme où il à été spécifié dans le document: nous avons repris le pseudocode dans la Fig. 13, dans sa forme la plus simple que nous permet de mieux le comprendre. Les opérations de l'algorithme sont plutôt simples: la destination se limite à calculer le délai des paquets reçus comme différence entre le timestamp du paquet et son propre timestamp; le délai est en suite envoyé à la source, où se passe là plus parte de l'algorithme. En fait l'expéditeur garde un historique des délais mesures: le minimum observé est le `base_delay` qui veut être une estimation de la partie constante du délai, autrement dit le délai de propagation, qu'on observe quand la file d'attente est vide. Du coup, la différence entre le délai instantané et le délai de propagation correspond au délai dû à la file d'attente: le but de l'algorithme est d'introduire un délai additionnel sur le parcours égale à `TARGET`, qui était originalement à 25 ms et récemment à été choisi comme 100 ms. Du coup, la fenêtre de congestion est modulé par un contrôleur linaire en fonction de la distance du délai mesuré depuis le `TARGET`. Le dernier paramètre de l'algorithme est donc le coefficient multiplicatif du contrôleur, `GAIN` qui était pas spécifié dans la version originelle du draft. Nous allons choisir 1/`TARGET` comme valeur de `GAIN`, selon les directives qui veulent que le contrôleur ne puisse pas grimper plus vite que un TCP standard. Ce comportement contribue à le principal objectif du protocole qui était de fournit un service à basse priorité. Pour atteindre cela, il y a aussi le requis que le protocole détecte la congestion avant de TCP et qu'il réagisse plus vite: comme LEDBAT commence à diminuer le débit dès que il mesure un délai plus grand que `TARGET`, sa réaction précède celle de TCP standard.

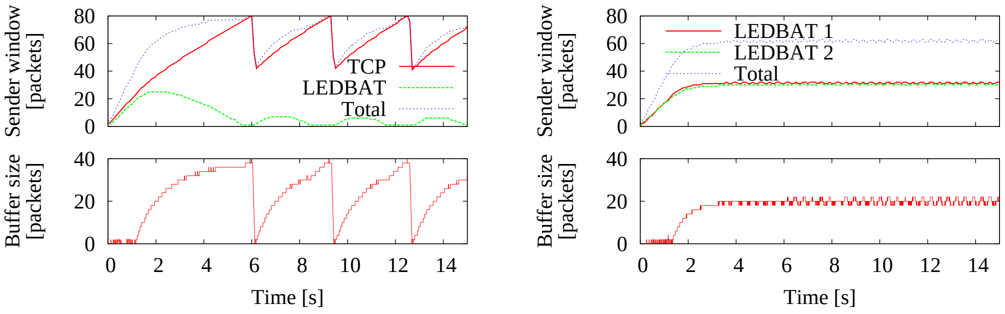
Nous avons implémenté cette algorithme dans le simulateur de réseau à niveau paquet `ns2`. L'algorithme à été implémenté comme un nouveau congestion contrôle pour TCP, en utilisant le timestamp option disponible dans TCP pour calculer les délai. Vu que le simulateur utilise le même système modulaire utilisé dans le noyau de Linux, notre code, disponible pour le téléchargement ici [10] peut être utilisé aussi dans un système d'exploitation réel.

Avec cet outil, nous avons pu étudier le comportement du protocole dans des scénarios très

```
on data_packet @ RX:
   remote_timestamp = data_packet.timestamp
   acknowledgement.delay =
       local_timestamp() - remote_timestamp

on acknowledgement @ TX:
   current_delay = acknowledgement.delay
   base_delay = min(base_delay, current_delay)
   queuing_delay = current_delay - base_delay
   off_target = TARGET - queuing_delay
   cwnd += GAIN * off_target / cwnd
```

Figure 13: Pseudocode de LEDBAT pour la source et la destination.



(a)

(b)

Figure 14: Évolution temporelle de la fenêtre de congestion de l'émetteur et de la dimension de la file d'attente pour TCP-LEDBAT (a) et LEDBAT-LEDBATinteraction (b).

simples, pour déterminer d'abord s'il atteint ses objectif. La plus importante caractéristique de LEDBAT veut être sa basse priorité: du coup nous allons étudier le comportement d'un flux LEDBAT et un flux TCP qui partagent le même lien. L'évolution temporelle des fenêtres de congestion et de la dimensionne de la file d'attente est représenté en Fig. 14-(a). Nous voyons que le flux TCP n'est pas du tout influencé par la présence de LEDBAT sur le même lien: nous pouvons observer en fait que à chaque fois que la file d'attente dépasse les 20 paquets, le flux à basse priorité diminue son débit pour laisser l'espace au plus agressif flux TCP, qui garde son comportement habituel à dent de scie. . Pourtant, LEDBAT est capable d'utiliser la bande laissée libre par le flux TCP, ce qui augment l'utilisation total du lien.

Dans la Fig. 14-(b) nous observons le comportement de 2 flux LEDBAT qui partagent le même lien. Nous voyons que, comme ils ont commencé au même temps, ils ont aussi mesuré le même délai de propagation. Donc, il estiment correctement la dimension de la file d'attente et arrivent facilement à partager dans une façon équitable la capacité. En outre, la taille de la file d'attente reste très stable, sur 20 paquets, que corresponds à les 25 ms de `TARGET` délai spécifiés dans le draft.

Malheureusement, la situation change radicalement si nous faisons commencer les flux dans deux instants différentes: dans ce cas là, le comportement observé dépende fortement de le délai entre les deux flux, comme nous pouvons observer dans les trois plots de Fig. 15-(a). Dans celui en haut, le deuxième flux commence quand le première n'a pas encore commencé à occuper

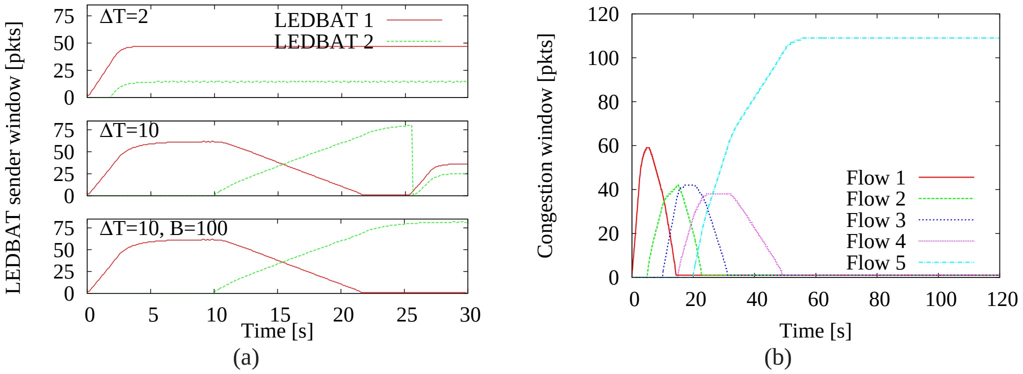


Figure 15: LEDBAT vs LEDBAT évolution temporelle de la fenêtre de congestion pour différent conditions initiales et latecomer advantage, même dans de scénarios multi-flux.

la file d'attente; du coup, bien que les deux flux aient bien mesuré le délai de propagation, les premier ayant commencé d'abord, arrive à occuper une partie de la bande disponible bien plus importante que le deuxième. Dans le plot de milieu, le deuxième flux commence quand le première a déjà atteint le `TARGET`: pour cela, le deuxième surestime le délai de propagation de `TARGET` ms et commence à augmenter sa fenêtre de congestion. Vu que le premier flux diminue son débit à la même vitesse à laquelle le deuxième augmente, celui-ci n'a pas la possibilité de corriger sa mauvaise estimation: finalement le premier flux se tais complètement alors que le deuxième continue à monter. Heureusement, à un certain moment le buffer est saturé et il y a une perte qui cause une brusque réduction de la fenêtre de congestion et le vidange de la queue. Après cette évènement, le deuxième flux corrige l'estimation du délai de propagation et l'équité est récupéré. Toutefois, si le buffer avait été assez grand pour pouvoir contenir une longue queue, alors nous nous seront retrouvés dans la situation du plot en bas, où nous voyons que une très mauvais répartition de la bande passante peut persister longtemps. En Fig. 15-(b) nous voyons que ce phénomène peut se présenter aussi quand plusieurs flux insistent sur le même liens, à conditions que le buffer soit assez grand pour contenir la file d'attente crée.

Une objection à nos simulations, qui à été faites sur la liste de diffusions du groupe de travail IETF, était que ce phénomène ne peux pas se présenter dans un véritable réseau, parce que les retards aléatoires introduits par le système d'exploitation, les routers et l'autre trafic ne laisseraient pas synchroniser autant les deux flux. Si dans un certaine mesure il est vrai que il y a beaucoup moins de synchronisation dans un réseau réel, pourtant ce phénomène de latecomer advantage peut arriver quand même. Pour démontrer cela, nous avons profité de la publication du code officiel de LEDBAT [85], pour tester la même situation dans un testbed réel. Les résultat que nous avons mesuré est présenté en Fig. 16 et il est tout à fait pareil à nos simulations. Nous voyons que le deuxième flux fait taire le premier, parque que il surestime le délai de propagation, comme nous pouvons remarquer dans le plot à droite où le offset calculé du `TARGET` est toujours égale à zéro pour le deuxième flux.

Après cette première évaluation nos conclusion sont que LEDBAT est un bonne protocole, qui est capable de respecter ses objectif: fournir un service à basse priorité, qui ajoute seulement un petit délai dans la file d'attente, mais qui est au même temps capable de profiter de la capacité disponible sur le lien. Par contre, il a un problème dans son design originel qui porte à une situation de partage pas équitable des ressource quand deux flux insistent sur le même lien. Nous allons étudier ce problème dans le reste de ce section, en proposant aussi des solutions efficaces.

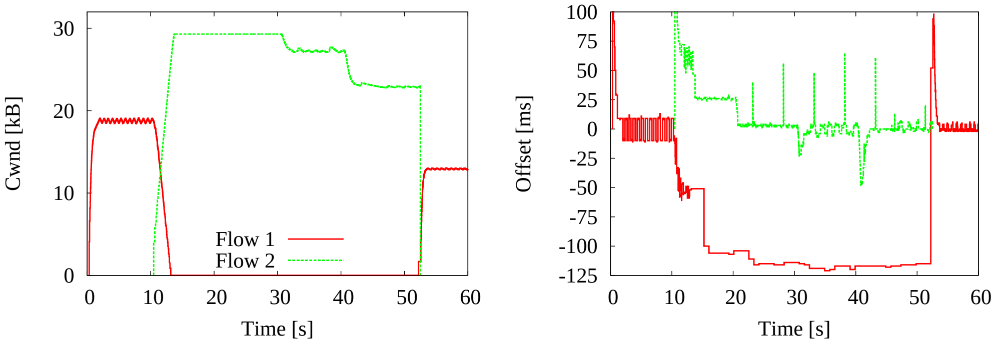


Figure 16: Experimental LAN testbed: Congestion window evolution (top) and offset from the target (bottom) for two competing backlogged libUTP flows.

## Améliorer l'équité de LEDBAT

Dans le Chap. 9, nous proposons quatre solutions naïves à le problème d'iniquité que nous avons remarqué dans le protocole LEDBAT. Nous omettons leur description dans ce résumé, parce qu'elles ne sont pas parfaite mais elles font plutôt partie d'un parcours que nous a servi pour trouver une vrai solution qui puisse garder la basse priorité et l'efficacité de LEDBAT, en ajoutant aussi l'équité. En fait, cette qualité n'est pas négligeable, parce que, comme les applications P2P, pour lesquelles LEDBAT a été conçu, ouvrent plusieurs flux en parallèle, le fait que le dernière puisse arrêter tous les autres est assez gênant. En particulier cela impacte les performance de l'application quand des critères comme le tit-for-tat de BitTorrentsont utilisés (un pair envoie de préférence aux pairs qui l'ont plus servi dans les intervalles de temps précédents).

Au cours de notre étude, nous avons découvert que la raison principale pour l'iniquité de LEDBAT doit être recherché dans la spécification du contrôleur de l'algorithme de congestion contrôle, et, en particulier, dans la composante de décroissance additive. Déjà dans les dernières années '80, Jain [52] avait remarqué qu'un algorithme de congestion ayant cette composante était instable et incapable de converger vers un équilibre: il indiquait la décroissance multiplicative comme un élément essentiel pour l'équité de l'algorithme.

Nous avons défini dans le Chap. 10 un nouveau protocole, fair-LEDBAT ou fLEDBAT, qui ne réintroduit pas seulement la décroissance multiplicative dans LEDBAT pour atteindre l'équité, mais apporte aussi d'autres petites modifications pour améliorer l'efficience du protocole. Commencions pour reprendre la spécification de LEDBAT dans une façon plus formelle: si on dénote avec $D_{min}$ le `base_delay`, avec $\tau$ le `TARGET`, avec $q(t)$ le délai dû à la file d'attente au temps $t$ et avec $cwnd(t)$ la fenêtre de congestion au temps $t$, nous pouvons exprimer le contrôleur de LEDBAT comme:

$$\begin{aligned}
\Delta(t) &= (q(t) - D_{min}) - \tau \\
cwnd(t+1) &= \begin{cases} cwnd(t) + \alpha \frac{\tau - \Delta(t)}{\tau} \frac{1}{cwnd(t)} & \text{sans pertes,} \\ \frac{1}{2} cwnd(t) & \text{en cas de perte.} \end{cases}
\end{aligned}$$

Par contre fLEDBATest spécifié ci de suite, où $\alpha$ et $\zeta$ sont les paramètres de l'algorithme.

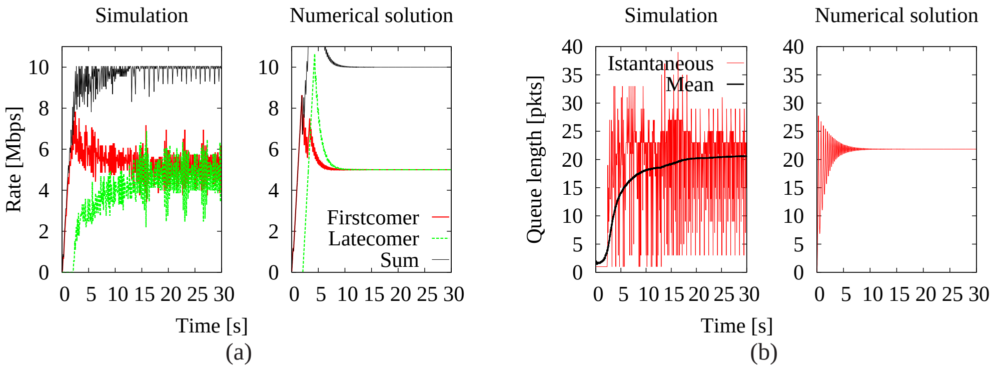


Figure 17: Comparaison de (gauche) simulation et (droite) solution numerique pour (a) debits et (b) dimension de la file d'attente.

$$cwnd(t+1) = \begin{cases} cwnd(t) + \alpha\frac{1}{cwnd(t)} & \text{sans pertes et } \Delta \leq 0, \\ cwnd(t) + \alpha\frac{1}{cwnd(t)} - \frac{\zeta}{\tau}\Delta & \text{sans pertes et} \Delta > 0, \\ \frac{1}{2}cwnd(t) & \text{en cas de perte}. \end{cases}$$

Nous voyons que nous avons apporté une double addition: (i) nous avons ajouté une terme additive avec $\alpha$ qui a l'objectif d'améliorer l'efficience de l'algorithme et (ii) nous avons réintroduit un terme multiplicatif avec $\zeta$ quand le délai d'attente dépasse le target délai $\tau$. Dans le Chap. 10 nous démontrons avec un modelé fluide du trafic que l'iniquité de LEDBAT est en effet dû à la composante additive et, en suite, que fLEDBAT a des bonne propriétés pas seulement de équité mais aussi d'efficience et convergence. Nous avons implémenté le nouveau protocole dans le simulateur `ns2` et nous avons aussi simulé notre modelé fluide numériquement: dans la Fig. 17 nous montrons les résultat de la simulation et de la solution numérique, où nous voyons que l'équité est rétablie et qu'il y a une parfaite correspondance entre la simulation et le modelé.

Dans le Chap. 10 nous conduisons une analyse très fine du nouveau protocole. Nous effectuons d'abord un tarage des paramètre $\alpha$ et $\zeta$ dans des scénarios avec un et plusieurs flux, pour comprendre l'intervalle des valeurs qui permet d'obtenir les meilleurs performances. En suite nos simulations se concentrent sur le modelé de trafic (backlogged ou par chunk) et sur des scénarios multiflux et multipairs, pour évaluer le protocole dans ses conditions d'utilisation typiques, vu que LEDBAT a été conçu pour être le protocole de défaut de BitTorrent. Dans tout les cas fLEDBAT se comporte mieux que LEDBAT et atteint une meilleur performance ainsi que une équité supérieure.

# Abstract

P2P applications are certainly to be counted among the most bandwidth greedy users of the Internet, due to their wide diffusion as well as the kind of service they provide (e.g. file-sharing, live-streaming). Therefore, they pose continuously renewing challenges to network operators: P2P traffic must be correctly monitored and managed in order for it to coexist peacefully with traffic of other applications, i.e. without degrading their performance. Moreover, not only do these services generate an enormous amount of traffic, they also impact the traditional assumption at the base of the network architecture, in which the content was supposed to be located on a few server machines and to be accessed from many remote clients. With P2P, instead, every host becomes potentially a content provider, thus imposing new requirements especially to the access part of the network.

In this thesis we investigate several strategies to help operators deal with the huge amount of P2P traffic which has recently invaded their networks. In the first part we focus on the identification of such traffic, a necessary preliminary step to all sorts of management activities operators may want to apply: from monitoring of traffic trends, to differential treatment or queuing of packets (to ensure quality of service), to lawful interception of illegal data transfers. Yet, P2P traffic classification is not an easy task, especially if addressed with traditional techniques such as port-based classification or deep packet inspection (DPI). Therefore, this work investigates behavioral traffic classification, which recognizes the pattern of traffic generated by an application (e.g., number of host contacted, with how many packets, with which periodicity and so on). First we propose a framework able to explore the space of behavioral features which can be defined over flow-level measurement, ranking them according to their usefulness for traffic classification. Guided by this analysis, we introduce Abacus, a behavioral classifier tailored for P2P live-streaming applications, whose performance we evaluate by means of an extensive experimental campaign. We show that the combination of a powerful machine learning algorithm with a careful choice of descriptive features capturing the key properties of traffic results in an extremely accurate, robust, lightweight and, furthermore, fine-grained classifier. To strengthen our claim, we present a comparison of Abacus with a sophisticated payload-based classifier: Abacus can achieve the same accuracy, consuming, however, far less resources.

In the second part, we study how the accuracy of traffic classification is affected by data reduction techniques, more and more required to shrink the huge amount of data coming from network measurement to a manageable size. First, we leverage the Abacus classifier as a test case and study the impact on its accuracy of using data from flow-level monitors (i.e., NetFlow) instead of packet-level measurement. Then, we experiment with Abacus and flow-sampling: this effect may arise when the classifier is moved towards the network core, where not all traffic directed to the target host is likely to be available due to routing. On the one hand, flow-level records proved a well-suited input for such classification techniques, despite the coarser time granularity and the reduced information. On the other hand, flow sampling may cause a severe degradation of accuracy only when it significantly changes the statistical properties of traffic used for the classifi-

cation – for instance, in Abacus, by altering the distribution of signaling and data flows. Finally, a detailed analysis of the impact of packet-level sampling on traffic monitoring and classification is conducted. Even though we show that most traffic properties are heavily distorted already at low sampling, regardless of the sampling policy adopted, we find a smart sampling technique which reveals itself well suited for statistical traffic classification, provided that the classifier is trained with data obtained at the same rate of the target data.

Finally we take an orthogonal approach: instead of adding more intelligence in the network to handle P2P traffic, we propose to modify the applications themselves. In particular, the idea is to change the transport protocols employed, to make them gentler towards both the network and other traffic. We study a new lower-than-best-effort congestion control algorithm, which is already implemented on top of UDP by BitTorrent and which is also on its way to standardization at the IETF under the name of LEDBAT (Low Extra Delay BAckground Transport Protocol). The goals of the protocol are to efficiently use all spare bandwidth, while adding only a small amount of additional delay on the forward path; additionally, by implementing a delay-based control algorithm, the protocol aims at yielding to other traffic (i.e., web or mail). In this work, after a black-box evaluation of the BitTorrent implementation, which highlighted some good properties of the protocol as well as some shortcomings, we developed an open source implementation and carried on a extensive simulation study of LEDBAT. We find that the current specification of the protocol is affected by a latecomer advantage, and hence an unfairness issue, which might heavily hit the performance. However, a careful study of the properties of the controller allowed us, first, to identify the root cause of the unfairness in the additive decrease component and, second, to propose an effective modification to the algorithm, solving the issue as well as keeping all the good properties of the original design.

# Contents

# Chapter 1

# Introduction

## 1.1 The rise and (apparent) fall of P2P applications

By now it has been more than ten years since the birth of peer-to-peer applications, with the release of Napster in 1999, and it looks like they are likely to stay much longer, further changing the landscape of the network. In fact, there is no doubt that P2P applications have played a very important role in the last decade, taking advantage of the spreading of faster network access links (ADSL) and at the same time promoting their diffusion (for many users an always-on connection was equal to an always-on P2P application). As they are the main subject of study of this thesis, in the following we will briefly review the history of P2P services, from their initial growth and fast diffusion, until their *apparent* fall in favors of video traffic, especially web-based. However, we will see that the future evolution of P2P traffic remains to date an open question, but, still, it is sure that it will not be completely ruled out from the future Internet landscape.

Presently, P2P applications are still among the major contributors to the overall amount of data exchanged over the Internet. There are several reasons at the base of their success, both technical and commercial, which we will discuss below. But, first of all, we need to better define what constitutes a P2P system.

Initially, distributed applications were built according to the traditional client-server paradigm: thin clients running on many machines connected to a central server to exploit its, hopefully sufficient, resources, be they computational power, storage or bandwidth. As powerful machines and faster network access links became more and more available, the inefficiency of relying only on the central server resources without exploiting the clients became more and more evident as well. Building upon this observation, in P2P systems each host running the application shares a portion of its resources with the others peers, directly exchanging data with them: in this way the more peers participate in the system, the more resources are available. For instance, in file-sharing applications, such as BitTorrent, the most important resource is clearly the bandwidth, for the goal is to distribute files as fast as possible. Therefore, each peer, while downloading pieces (a.k.a. chunks) of the file from other hosts, concurrently uploads data to other peers, thus devoting part of its own upload bandwidth to the system.

From a technical point of view, the strong points of such an architecture are clearly its inherent *scalability and robustness*. As for the former, it can be seen that the capacity of the system grows naturally with the number of hosts involved, as each peer actively contributes a portion of its own resources. Regarding the latter, the lack of a strong dependence from a centralized system [1]

[1] Actually some P2P systems use a centralized infrastructure to manage tasks such as user login or initial discovery of peers, however they are usually not essential to the working of the application.

guarantees that there is no single point of failure. Moreover, robustness comes from the fact that P2P applications need to be designed since the very beginning to deal with an extremely changing population of peers, which may fail or disappear without any previous notice because of network problems or user behavior (i.e., churn).

Thanks to its advantages, the P2P paradigm has proved particularly effective as well as versatile, given that it has been adopted by extremely different applications. In the beginning the first widespread P2P systems were essentially file-sharing networks (e.g., Napster, Kazaa, Gnutella), owing part of their success to the fact that they allowed users to exchange copyrighted content (illegally but for free). Most of these applications have been superseded by the diffusion of BitTorrent mainly because of its efficiency: as a matter of fact, BitTorrent has become a common way of distributing large files on the Internet, e.g. Linux distribution or software updates. More recently, the same distributed architecture has been successfully applied to very different services: for instance to VoIP with Skype, which already counts millions of users worldwide, and lately to video streaming, either live with P2P-TV applications such as PPLive, SopCast, TVAnts or on-demand with software like Joost. Among the last appeared P2P applications, two deserve to be cited: the live-streaming effort from BitTorrent 3.0, and Spotify, a music on-demand streaming service which builds a P2P overlay among users to enhance the performance of the company central servers and content delivery network.

In consequence of such a pervasive diffusion, P2P applications have become responsible for a large portion of global Internet traffic. At the end of 2007, Internet studies such as [92] indicated that P2P traffic was estimated to be in different world regions in average between 49 and 83 percent of total traffic, thus corresponding to the biggest consumer of bandwidth and exceeding even Web traffic. In that very year P2P live-streaming applications started becoming popular, beginning from China where they were originally developed. Concerns were raised by network operators, who considered such services and the related traffic as real threats to their networks; moreover, early measurement studies [86] on this technology confirmed such preoccupation, showing how the upload bandwidth consumed by peers could grow extremely large (i.e., many times the stream rate). The network community tried to address the problem with several projects such as Napawine [115] (in whose context part of this thesis was developed) or P4P [183], whose goal was basically to develop more ISP-friendly applications. At the same time, large operators independently started to deal somehow with the huge amount of P2P traffic, sometimes even by means of drastic measures like throttling it [21], thus violating network neutrality.

Nevertheless, while P2P was foretold to explode [92, 100], recent studies have somehow scaled down such a forecast [112, 169], highlighting instead a predominance of web traffic in nowadays Internet traffic breakdown, because of the diffusion of web-based video (e.g. YouTube, Megavideo) and web-based file-sharing (e.g. Megaupload, Rapidshare). The annual report and forecast of Cisco [169] further confirms the overtaking of video traffic – which has seen an astonishing increase – over P2P traffic as percentage of total amount of data exchanged over the Internet.

However, we should not rush our conclusions from this data. Although decreasing in percentage, P2P traffic is still increasing at high pace in absolute volume, which currently amounts to 4 thousand petabytes, and it is expected to double by the year 2015; furthermore, Cisco's report considers P2P live-streaming traffic as video traffic and not as P2P, thus biasing such statistics. Moreover, the decline of P2P traffic may reflect only short term effects, as pointed out by late measurement works [76]. This study shows that such a trend is changing in some regions: since web-based file-sharing services like Rapidshare enforce limits on download size and speed and operators start limiting traffic towards such websites, users are reverting to well trusted P2P applications. Besides, P2P in Adobe Flash player with RTMFP [24] and consequently in the browser, as well as new services of content distribution (e.g. Spotify) might completely change the trend of global P2P traffic in the near future. In summary, it seems like P2P traffic is not going to disappear soon and will continue to represent a consistent portion of total traffic.

## 1.2 Motivation

In such a context, this thesis has the goal of developing tools and protocols to help manage P2P traffic, which, given its volume and peculiar characteristics, continues to represent a challenge for operators. In particular, this thesis wants to provide solutions to better (i) *identify*, (ii) *measure* and (iii) *control* P2P traffic. In the following, we briefly introduce each of these topics, which, actually, correspond to the different parts of this thesis.

Whatever the strategy ISPs want to adopt to deal with P2P traffic, *the first step they need to perform is to efficiently identify it.* In fact, only by correctly telling which packets or flows belong to P2P applications, operators can (i) implement differential queuing or treatment of such traffic, (ii) comply to Quality of Service agreement and (iii) prevent P2P traffic from hurting the performance of other applications.

However, despite the huge effort [32, 34, 63, 77, 80, 83, 101, 102, 105, 119, 124, 131, 132, 132, 140, 161, 165, 184], devoted by the research community to the subject of *traffic classification*, i.e., associating an application label to packets transmitted over the network, the identification of P2P traffic is still far from being an easy task, due to a few main reasons. On the one hand, traditional techniques based on transport-layer identifiers (i.e., TCP or UDP port numbers) are fooled by modern applications, which either use random ports or hide behind well-known ports belonging to other protocols (e.g. port 80 for HTTP). On the other, more sophisticated techniques like *deep packet inspection*, which search packet payloads for evidences of the application protocol, usually require several memory accesses and computationally expensive operations, thus falling short of coping with increasing transmission speed.

For these reasons, the community has proposed novel methods for classifying network traffic. For instance, *statistical classifiers* [34, 63, 124, 130, 161, 177] characterize traffic by means of features calculated at flow-level (e.g. size and direction of first packets, total length and duration of flows). Another recent and promising family of algorithms is *behavioral classification* [101, 102, 184], which tries to capture the properties of the pattern of traffic generated by an host running an application, leveraging the fact that being the applications different, so will be the pattern of traffic generated by them (e.g. a P2P application contacting many peers versus a traditional client exchanging data with a single server). Such an approach is extremely lightweight, usually employing flow-level measurement as those provided for example by flow-level monitors like NetFlow, so being a perfect candidate for dealing with the volumes of traffic generated by P2P applications. However, before this work, behavioral classifiers only achieved coarse-grained classification of wide classes of applications (i.e., P2P vs web) and had never been applied to the emerging P2P live-streaming applications. We address the design of such a classification engine in the first part of this thesis.

While the community is busy designing more lightweight techniques like behavioral classifier, operators must already take action with respect to the large amount of traffic of nowadays networks. Cisco's report [169] estimates the total traffic transmitted over the Internet to be in the order of 20 exabytes per month in 2010 and the prevision sees it growing upto nearly 1 zettabytes in 2015: therefore, operators are obliged to adopt data reduction techniques to contain the quantity of measurement data they get from their networks. This consideration holds in particular for P2P applications, which, as mentioned above, represent a large slice of the global Internet traffic. In the second part of this thesis we will investigate *what is the impact of data reduction on the accuracy of traffic classification*.

For instance, last years have confirmed the trend in *flow-level monitoring* of operational networks: the growing use of NetFlow also standardized as IPFIX at the IETF [57], is motivated by a larger scalability with respect to packet-level measurement along with a larger expressiveness compared to the coarse-grained counters of SNMP. *Packet sampling* has become a common practice for network operators as well, reducing the loading of monitoring equipment, as only a fraction of all packets is selected to be processed, according to different criteria (e.g. systematically 1-out-of-N, at random). This kind of data is often the only one available, or at least the only one which is possible to collect. Therefore, a modern classifier cannot be exempted from dealing with sampled data. In the second part of this dissertation, we investigate this issue: whether behavioral and statistical classifiers can work with sampled or flow-level aggregated data and *what is the impact of this reduced information on classification accuracy*.

Moreover, if classifiers are deployed in the core of the network, they are likely to observe only a portion of the traffic directed to the host that is the target of the classification; yet, this was a common assumption for behavioral classifiers, which need as much information as possible to characterize the pattern of traffic of a host. Such a requirement, which is not always met, has somehow hindered the adoption of behavioral classification engines. Thus we also evaluate the performance of behavioral classification in the presence of *flow-sampling*, i.e., when only some flows are inspected at the classification point and only an incomplete vision of traffic can be gathered by the classifier. A chapter of this thesis is dedicated to understand whether behavioral classifiers are also suited for classification at the core of the network.

Besides supporting network operators with better tools to manage P2P traffic, *a complementary approach to prevent P2P traffic from badly impacting other application traffic is to provide developers with protocols specifically designed for these distributed services*. In particular such protocols should be designed to implement a *lower-than-best effort service*, which automatically yields to higher priority traffic, while efficiently exploiting the available bandwidth at the same time. This road has been recently taken by BitTorrent developers, which in late 2008 announced that the forthcoming version of the client would drop TCP as transport layer protocol in favor of a new congestion control algorithm implemented on top of UDP and named uTP. Despite the initial buzz about such an announcement, with people wrongly foretelling a soon to come congestion collapse of the Internet when all BitTorrent traffic would switch to unresponsive UDP, the intention of BitTorrent developers was sound. To prove their good intentions, they started chairing an IETF working group to openly design this new protocol under the name of LEDBAT, which stands for Low Extra Delay BAckground Transport protocol.

As stated in the IETF draft [166] documenting the algorithm, the LEDBAT protocol has three main goals: (i) to efficiently exploit the available bandwidth, (ii) to introduce a small extra-delay in the network path, (iii) to quickly yield to other traffic sharing the same bottleneck (e.g. TCP). The design of the protocol is based on the observation that, in nowadays networks, congestion is mostly self-induced and happens exclusively at the access, while the over-provisioned network core is almost congestion free. According to BitTorrent developers, the bottleneck link is found at the home gateway, where also actual queuing happens. TCP detects congestion when a loss occurs, i.e., when the buffer overflows: thus, given the large buffers usually found in home network equipment, TCP may introduce large delays (up to a few seconds [156, 164]) and latencies, to which interactive applications are particularly sensible. LEDBAT tries to prevent this situation by detecting early that queuing is building up and by adjusting the rate in such a way that only a small portion of the buffer is occupied (avoiding losses altogether). To achieve this goal, LEDBAT implements a windowed delay-based congestion control algorithm, constantly monitoring the one-way delay on the forward path and adjusting the congestion window by means of a linear controller in order to add only a small *target delay* in the bottleneck (set to 25ms by default in the draft).

Thanks to BitTorrent popularity, LEDBAT is surely going to play an important role in the Internet and measurement studies [76] have highlighted that it has already changed the distribution of traffic in large ISP networks. Therefore the evaluation of its performance is clearly a topic of large interest. Besides, given the already large literature on congestion control, both about delay-based algorithms [40, 41, 84, 114, 117] and lower-than-best-effort protocols [103, 111, 118, 175], one may tend to question whether LEDBAT is a worth addition to the Internet architecture or BitTorrent developers would have rather chosen an already well-known and tested solution. Moreover, while on the one hand chairing a IETF group clearly witnesses the good intentions of BitTorrent, on the other the application has been deploying the protocol much earlier than the draft. Actually, in the original design document [136], different parameter values are specified for the official implementation (e.g. the target delay is set to 100,ms), so that a completely different flavors of the protocol is actually implemented. Several issues have been also pointed out on the working group mailing-list, so that a rigorous evaluation of the protocol design is more than welcome by the community. In the last part of this thesis we will study LEDBAT behavior and its performance, by employing different tools (i.e., analysis, simulation, measurement) and proposing solutions where we find problems in the original specification.

## 1.3 Contributions of this thesis

This thesis is organized in three parts. In the first one we are going to explore the **behavioral classification** approach for the identification of P2P traffic. First we demonstrate that a wealth of features can be defined over data found in NetFlow flow-records, which are able to capture such distinct properties of P2P traffic that they even allow fine-grained classification of specific applications. We first introduce a framework for the definition of behavioral features, so as to systematically explore the whole feature space, combining different simple criteria to build richer attributes. We thus obtain a very general characterization of behavioral features, whereas all previous work focused on very specific techniques, which are difficult to compare between each other because of the heterogeneity of approaches and testing methodologies. Our framework is instead general enough to contain features used by previous classifiers, thus favoring their comparison. Then we evaluate the information content of single features by means of information theoretic and geometric metrics, in order to find out which are the most useful ones for our classification purposes. We employ machine learning techniques (Support Vector Machines [62] and Decision Trees [106]) to actually perform the classification, showing that our methodology achieves good performance though using simple criteria and features.

Leveraging the aforementioned findings as well as our knowledge of the internals of P2P live-streaming applications, we propose a specific signature for this kind of services, which are increasingly popular as well as bandwidth consuming. We define a simple signature based on the distribution of the rate of flows received by the target peer in small time-windows, which highlights specific properties of the P2P application implementations. Our classifiers, which is named Abacus, is based on Support Vector Machines and on a rejection criterion to classify "unknown" traffic, i.e. different from the one used for the training. Our extensive experimental campaign demonstrates that not only does this classification engine achieve extremely high accuracy and fine-grained classification, but it is also portable across different network settings and times. We also compare our solution with a state-of-the-art payload based classifier [77], tailored for the same kind of applications, in terms of classification performance and of computational complexity as well. Our analysis shows that Abacus is capable of achieving the same accuracy together with a much lighter classification process.

The second part of this thesis, instead, tackles the issue of assessing the **impact of data reduction techniques** (e.g. aggregation and sampling) on classification performance. First, we verify the very first claim of this work, that behavioral classification is feasible by employing solely flow-level data, like NetFlow flow-records. Using the previously introduced Abacus classifier as test-case, we see that classification accuracy remains high, notwithstanding the coarser time-granularities of this data (Abacus was originally designed to use a monitoring window of 5 s, whereas NetFlow timeouts are usually in the order of minutes).

Afterwards, we evaluate the performance of the Abacus classifier when it is deployed in the core of the network, where the phenomenon of *flow-sampling* might interfere with the classification. In fact, because of routing, the engine is likely to observe only a fraction of all flows directed to any given host, so gathering a limited picture of the pattern of traffic generated by the application. We test Abacus in such a situation, using both a random sample of flows and real-life routing tables. Our results show that classification is still possible and accurate even with missing data, as long as the traffic observed allows the classifier to gather a statistically representative sample of flows. Under this condition, in fact, Abacus signatures are robust because the distribution of flows is not biased by the sampling due to IP routing.

We then move on to evaluating the impact of *packet sampling* on traffic measurement and classification. We actually modified a network monitoring tool, i.e. `tstat` [16], which outputs several metrics of packet flows both in aggregated and single-flow fashion, to apply different sampling policies with arbitrary sampling rates. We run the tool on an extensive set of packet-level traces and use the data for a twofold analysis. First, we evaluate the distortion due to sampling by means of statistical metrics which measure the distance between the distribution of aggregated features over sampled and unsampled traffic. We show that sampling causes a significant degradation of features, even for low sampling rate and no matter how sophisticated the sampling policy employed. Second, we employ single-flow features to assess what is the impact of using sampled data when performing traffic classification. First we use again tools from information theory to evaluate the information content of features, showing that the degradation observed in the absolute value of the features does not necessarily imply a reduction of information content. Since diversity among classes is somehow preserved in spite of sampling, traffic classification is still possible and accurate with sampled data, provided that the classifier is trained with traffic gathered at the same sampling rate as the test traffic.

In the third part, we finally deal with **congestion control for P2P applications**, in particular with the lower-than-best-effort LEDBAT protocol, proposed by BitTorrent developers. First we conduct a measurement study of its reference implementation found in the official BitTorrent client. These experiments date back to the time before the specification of the protocol was made public: therefore, we used a black-box approach, to understand whether LEDBAT abide by its goals. We build a testbed to control network conditions (e.g. available bandwidth, delay) and look at the reaction of the protocol to such a varying scenarios. We show that LEDBAT is a promising protocol, able to efficiently exploit the bandwidth while only adding a small delay to the network path. At the same time, we argue that the definition of lower-priority service is not straightforward, as different TCP implementations, different parameter choices and different scenarios can significantly affect the results.

Although extremely useful to understand the behavior of a protocol in real life scenarios, measurement and testbed experiments are not enough to evaluate the goodness of the design of a protocol. For this reason we develop an open source implementation of LEDBAT for the packet-level network simulator `ns2`, which can also work in the Linux kernel as an optional congestion control module. By simulating the protocol in simple scenarios we are able to better grasp the dynamics of the congestion control, observing each single aspect on its own. Simulations confirm the lower-priorities properties of LEDBAT, but they also expose a *latecomer advantage* affecting the protocol. Actually, when two LEDBAT flows beginning at different times traverse the same link, the latecomer may gather an incorrect delay estimation provoking an aggressive behavior which may lead to long time starvation, potentially impacting application performances as well. Having found this unfairness issue, which was already known to afflict delay-based protocols since TCP Vegas time, we investigate the root cause of the problem and propose a few possible solutions to help relieve it.

Our first observation is that the buffer queue needs to drain to correct the delay estimation of concurrent LEDBAT flows; therefore we suggest adding a slow-start phase at the beginning of LEDBAT flows, whose steep ramp-up is likely to cause, in order, buffer overflow, packet loss, flow back-off and queue draining. Yet, this solution seems in contrast with the lower-priority nature of LEDBAT, as also normal flows may experience losses and degraded performance. Moreover, digging further, we found that the main cause on the fairness is rooted in the controller design: actually early studies from late eighties [52] already pointed out that with an *additive decrease* component the system is not able to converge to a fair share of the available resources. We then propose three increasingly better solutions, which overcome the fairness issue by restoring the multiplicative decrease component. The simpler two solutions we propose are effective, though slightly naive: the first adds a random congestion window drop to the controller specification, while the second one simply changes the algorithm replacing the addictive decrease with a multiplicative one. However, only the very last solution we propose (called fLEDBAT for fair-LEDBAT) is capable of achieving low-priority, fairness and efficiency at the same time. We provide a mathematical fluid-model of this last solution, which analytically proves the properties of the protocol, as well as an extensive simulation study considering also complex P2P-like cases to evaluate the impact of such a protocol in realistic application scenarios. We underline that this work on the LEDBAT protocol is also discussed and cited in the IETF draft [166] itself.

## 1.4 Thesis outline

An overall view of the content of this thesis is provided in Tab. 1.1. As you can see, this dissertation is divided in three parts, respectively dedicated (i) to traffic classification of P2P traffic, (ii) to traffic classification and data reduction and (iii) to congestion control for P2P applications. For each part, we listed the chapters along with the main methodologies used (for traffic classification chapters we listed the machine learning algorithm employed), a few keywords and our related publications. In the following we complete this information with a brief summary of each chapter, to help the reader find this way along this thesis.

In Chap. 2 we introduce the problem of traffic classification, clearly stating its goals and possible applications. We review the related work in this widely investigated field, proposing a taxonomy of classification algorithms based on the data exploited for the classification, to better orientate ourselves among the various solutions. Then we discuss the main challenges in developing new classifiers, which necessarily have to cope with modern applications and network technologies. We also describe the machine learning tools we exploit later on when developing our classifier, i.e. Support Vector Machines and Decisions trees. Finally we present methodologies and metrics to evaluate classification performance.

Chap. 3 contains a preliminary study on the possible traffic features for P2P traffic that can be defined over data found in NetFlow flow-records. A general framework is defined, which uses

Table 1.1: Thesis synopsis

| | | Methodology | Keywords | Publications |
|---|---|---|---|---|
| Part. I | Chap. 2 | - | DPI, statistical classification, machine learning, dataset | - |
| | Chap. 3 | Decision Trees | P2P behavioral classification, Information Gain, ReliefF | [160] |
| | Chap. 4 | Support Vector Machines | behavioral classification, Abacus, P2P-TV | [32, 173] |
| | Chap. 5 | Support Vector Machines | behavioral classification, stochastic packet inspection, computational analysis | [79] |
| Part. II | Chap. 6 | Support Vector Machines | NetFlow, flow-sampling, Abacus | [158, 159] |
| | Chap. 7 | Decision Trees | packet sampling, statistical classification, Information Gain | [142, 170] |
| Part. III | Chap. 8 | Experimental Testbed | black-box evaluation, `netem`, ADSL, BitTorrent | [156] |
| | Chap. 9 | Simulation | LEDBAT, `ns2`, latecomer advantage, slow-start, multiplicative decrease | [46, 155] |
| | Chap. 10 | Mathematical modeling, simulation | fair-LEDBAT, fairness, P2P-like simulation | [45] |

simple criteria to build a rich set of possible features. The information content of each features is evaluated by means of information theoretical and geometrical metrics, so as to determine which are the most useful for classification purposes. Finally we actually perform the classification using such features with the C4.5 decision tree algorithm.

In Chap. 4, we take advantage of the results of the previous chapter and define a set of features tailored for P2P live-streaming applications, which captures the specific characteristic of this kind of services. The overall classifier, named Abacus, feeds these features to a Support Vector Machine algorithm and also features a rejection criterion to enable the identification of “unknown” traffic. We extensively test the classifier with several experiments: we prove its portability across different networks, different access technologies, different times and different network conditions. Finally we conduct a sensitivity analysis to find out the parameter settings which achieve the best accuracy.

In Chap. 5 we compare the Abacus classifier with Kiss [77], a payload-based classification algorithm proved to be particularly accurate with P2P-TV traffic as well [144]. First we compare the raw accuracy of the two classifiers on the same set of traces; then, we contrast their requirements in term of computational resources required, i.e., CPU consumption and memory footprint. Finally we proceed with a more qualitative comparison, where we highlight pros and cons of both approaches.

We then move on to the second part of the thesis, which deals with traffic classification and data reduction. In Chap. 6, we test the Abacus classifier in more challenging conditions. First, we apply it on NetFlow flow-records data: we begin by describing how NetFlow works and how we modified the original Abacus design to cope with aggregated flow-records. We then present the accuracy achieved by the classifier in our experiments. In the same chapter, we also test Abacus in the case of flow-sampling. Initially, we analyze the distribution of flows across the IP address space, to understand whether peers are spread all over or whether they are concentrated in a few networks. Then, we evaluate the classifier both in a simulated scenario, where flows are sampled at random, and in a more realistic one, where we employ real-world routing tables from BGP core routers.

Chap. 7 is dedicated to packet sampling. After a review of related work on this topic, we introduce our methodology: sampling policies, dataset, statistical metrics, information theoretical metrics and classification algorithm employed. The chapter is then divided in three sections. The first one assesses the distortion of aggregated features, which allows us to gather a application-independent picture of the impact of sampling on traffic characterization. The second one evaluates the distortion of single-flow features, in particular the reduction of their information content. In the last section we perform traffic classification with sampled data, gauging the impact of different feature sets, datasets and training policies.

The third part of the thesis is about the LEDBAT lower-than-best-effort, delay-based protocol for P2P traffic. In Chap. 8 we present the result of our testbed experiments with the official closed-source BitTorrent client implementing the novel protocol, considering different successive versions. We observe LEDBAT behavior in different network conditions, enforcing different network capacities and delays and highlighting its positive aspects and limits.

In Chap. 9, we discuss the LEDBAT IETF draft which describes the congestion control algorithm. Then, by means of our `ns2` implementation, we perform a simulation study of the protocol, where we unveil the latecomer advantage. We then provide four possible solutions to the unfairness issue: (i) the use of random pacing of packets within an RTT, (ii) the use of slow-start, (iii) the introduction of a random window drop and (iv) the replacement of the additive decrease component of the controller with a multiplicative decrease. Those solutions that actually solve the problem are evaluated in term of fairness and network utilization.

Finally in Chap. 10 we present a more mature solution to the latecomer advantage. We introduce fair-LEDBAT (fLEDBAT), which substantially modifies the original LEDBAT design to achieve fairness and network efficiency, while still keeping the low-priority goal. The chapter contains the mathematical model of the protocol as well as the proof of its properties in simple scenarios. Finally we study more complex cases with heterogeneous RTTs and background traffic, mocking an actual swarm of BitTorrent peers.

Finally in Chap. 11, we summarize and discuss the main results of this thesis, foreseeing future research perspective and evolution of this work.

# Part I

# Traffic Classification

# Chapter 2

# Introduction to traffic classification

In this chapter we will introduce the reader to traffic classification: we define the problem, we explain the motivations behind it and we present possible applications of these techniques. This chapter contains an overview of the various solutions found in literature, from the earliest simpler classifiers to the latest, more sophisticated algorithms better suited for modern applications and network technologies. To better structure this overview, we divide the classifiers in a few categories according to the data they base the classification on.

Finally, we also present methodologies, tools and metrics commonly used to evaluate the performance of classification algorithms. We will employ such instruments in later chapters to assess the accuracy of our own classifiers and to compare them against other solutions.

## 2.1 Definitions and State of the art

Traffic classification is the task of associating network traffic with the generating application. Notice that the TCP/IP protocol stack, thanks to a clear repartition between layers, is completely agnostic with respect to the application protocol or to the data carried inside packets. This layered structure has been one of the main reasons for the success of the Internet; nevertheless, sometimes network operators, though logically at layer-3, would be happy to know to which application packets belong, in order to better manage their network and to provide additional services to their customers.

As observed by authors of several works [105, 116, 135] the information provided by traffic classification is extremely valuable, sometimes fundamental, for quite a few networking operations. For instance, it represents the first step for activities such as *anomaly detection* [140], i.e. the identification of malicious use of network resources. Also, a detailed knowledge of the composition of traffic as well as the identification of trends in application diffusion is required by operators for a better *network design and provisioning. Quality of service* (QoS) solutions [161], which prioritize and treat traffic differently according to different criteria, need first to divide the traffic in different classes: the application to which packets belongs is a very important aspect when assigning them to a class. In the same way, traffic classification enables differentiated class *charging* or Service Level Agreements (SLA) verification. Finally, some national governments expect ISPs to perform *Lawful Interception* [30] of illegal or critical traffic, thus requiring them to know exactly the type of content transmitted over their networks.

If, on one side, the applications of traffic classification are plentiful, the challenges classifiers have to face are not to be outdone. First, they must deal with an increasing amount of traffic as well as an equally increasing transmission rates: to cope with such speed and volume, researchers

Table 2.1: Taxonomy of traffic classification techniques

| **Approach** | **Properties exploited** | **Granularity** | **Timeliness** | **Computational Cost** |
|---|---|---|---|---|
| Port-based | Transport-layer port [131, 132, 140] | Fine grained | First Packet | Lightweight |
| Deep packet inspection | Signatures in payload [119, 132, 165] | Fine grained | First payload packet | Moderate, access to packet payload |
| Stochastic packet inspection | Statistical properties of payload [77, 83, 104] | Fine grained | Online, after a few packets (<100ms) | High, eventual access to payload of many packets |
| Statistical | Flow-level properties [105, 124, 132, 161] | Coarse grained | After flow termination | Lightweight |
| | Packet-level properties [34, 63] | Fine grained | After few packets ( 5) | Lightweight |
| Behavioral | Host-level properties [101, 102, 184] | Coarse grained | After flow termination | Lightweight |
| | Endpoint rate [32, 80] | Fine grained | Online after a few seconds | Lightweight |

are looking for *lightweight algorithms* with as little computational requirements as possible. The task is further exacerbated by developers of network applications doing whatever in their power to hide traffic and to elude control by operators: traffic encryption and encapsulation of data in other protocols are just the first two examples that come to mind. Therefore, researchers had to come out with novel and unexpected way for identifying traffic.

The large body of literature about traffic classification [32, 34, 63, 77, 80, 83, 101, 102, 105, 119, 124, 131, 132, 132, 140, 161, 165, 184] is a further evidence of the great interest of the research community towards this topic. In the following, we will present an overview of the different approaches and methodologies that have been proposed by researchers to solve this issue. It is important to underline that this is far from being an attempt to provide a comprehensive list of all papers in this field (which, given their number, would be particularly tedious). Such a detailed reference can be found in a few survey [105, 135] or in related community website (e.g., [3]). Our aim is rather to identify the most important research directions so far, as well as the most representative milestone works and findings, to better highlight our contribution to this already deeply investigated subject. Still, despite this huge research effort, the community has not put the last word yet on traffic classification, as a number of challenges and question still remain open.

To better structure this overview, we divide the classifiers in a few categories according to the information on which they base the classification. This widely accepted categorization, which reflects also the chronological evolution followed by research, is summarized in Tab. 2.1. The table lists the most important works in each category along with their most relevant characteristics. The most important traits of a classifier, which determine its applicability to different network tasks, are:

**Granularity** We distinguish between *coarse-grained* algorithms, which recognize only large family of protocols (e.g. P2P vs non P2P, HTTP vs Streaming) and *fine-grained* classifiers, which, instead, try to identify the specific protocol (e.g. BitTorrent vs eDonkey file-sharing), or even the specific application (e.g. PPlive vs SopCast live streaming).

**Timeliness** *Early classification* techniques are able to quickly identify the traffic, after a few packets, thus being suited for tasks requiring a prompt reaction (e.g. security). *Late classification* algorithms take longer to collect traffic properties, in some case they even have to wait for flow termination: such techniques are indicated for monitoring tasks, such as charging.

**Computational cost** The processing power needed to inspect traffic and take the classification decision is an important factor when choosing a classification algorithm. In the context of packet processing, the most expensive operation is usually packet memory access, followed by regular expression matching.

In the following we review each category in more detail.

In the first days of the Internet, identifying the application associated to some network packets was not an issue whatsoever: protocols were assigned to well-known transport-layer ports by IANA [5]. Therefore, **Port-based classification** [131, 132, 140] simply needed to extract such value from the packet header and then look it up in the list with the association port-application[1]. Unfortunately *Port-based* classification has become largely unreliable [100, 132]. In fact, in order to circumvent control by ISPs, new applications, especially P2P ones, either use non-standard ports, or pick a random port at startup. Even worse, they hide themselves behind ports of other protocols – this might enable bypassing firewalls as well.

To overcome this problem, **Payload-based classifiers** [77, 83, 119, 132, 165] were proposed. They inspect the content of packets well beyond the transport layer headers, looking for distinctive hints of an application protocol in packet payloads. We actually split this family of classification algorithms in two subcategories, *Deep packet inspection* (DPI) techniques that try to match a deterministic set of signatures or regular expressions against packet payload, and *Stochastic packet inspection*, rather looking at the statistical properties of packet content.

DPI has long provided extremely accurate results [132] and has been implemented in several commercial software products as well as in open source projects [16] and in the Linux kernel firewall implementation [8]. The payload of packets is searched for known patterns, keywords or regular expressions which are characteristic of a given protocol: the website of [8] contains a comprehensive lists of well known patterns. It is often used in Intrusion detection system [140] as a preliminary step to the identification of network anomalies. Besides being extremely accurate, DPI has been proved to be effective from the very first payload packets of a session [147], thus being particularly convenient for early classification.

Despite its numerous vantages, DPI has some important drawbacks. First the computational cost is generally high, as several accesses to packet memory are needed and memory speed is long known to represent the bottleneck of modern architectures [181]. String and regular expression matching represent an additional cost as well: although there exist several efficient algorithms and data structures for both string matching and regular expression, hardware implementation (e.g. FPGA) or ad hoc coprocessors (e.g. DFA) are often required to keep up with current transmission

[1] The list is also available in all standard Unix machines in the file `/etc/services`.

speed [110]. Moreover, such algorithms usually add further memory lookups. Second the keywords or patterns usually need to be derived manually by visual inspection of packets, implying a very cumbersome and error prone trial and error process. Last but not least, DPI fails by design in the case of encrypted or obfuscated traffic.

Stochastic packet inspection (SPI) tries to solve some of these issues, for instance by providing methods to automatically compute distinctive patterns for a given protocol. As an example, authors of [119] define Common Substring Graphs (CSG): an efficient data structure to identify a common string pattern in packets. Other works instead directly apply statistical tools to packet payload: authors of [83] directly use the values of the first payload bytes as features for machine learning algorithms; in [77], instead, a Pearson Chi-square test is used to study the randomness of the first payload bytes, to build a model of the syntax of the protocol spoken by the application (more details on this algorithm are provided in Chap. 5). Additionally, this last algorithm is able to deal with protocols with partially encrypted payload, such as Skype or P2P-TV applications.

Authors of [104], instead, propose a fast algorithm to calculate the entropy of the first payload bytes, by means of which they are able to identify the type of content: low, medium and high values of the entropy respectively correspond to text, binary and encrypted content. Authors argue that, even if this is a very rough repartition of traffic and moreover some applications are very likely to use all of these kinds of content, nonetheless such information might reveal useful to prioritize some content over the others (e.g. in enterprise environments, binary transfers corresponding to application updates to fix bugs deserve an high priority). Yet, SPI is still greedy in terms of computational resources, requiring several accesses to packet payload, though with simpler operations (i.e., no pattern matching).

**Statistical classification** [34, 63, 124, 130, 161, 177] is based on the rationale that, being the nature of the services extremely diverse (e.g., Web vs VoIP), so will be the corresponding traffic (e.g., short packets bursts of full-data packets vs long, steady throughput flows composed of small-packets). Such classifiers exploit several flow-level measurements, a.k.a. *features*, to characterize the traffic of the different applications [124, 130, 161]: a comprehensive list of a large number of possible traffic discriminators can be found in the technical report [129]. Finally, to perform the actual classification, statistical classifiers apply data mining techniques to these measurements, in particular machine learning algorithms.

Unlike payload-based techniques, these algorithms are usually very lightweight, as they do not access packet payload and can also leverage information from flow-level monitors such as [56]. Another important advantage is that they can be applied to encrypted traffic, as they simply do not care what the content of packets is. Nevertheless, these benefits are counterbalanced by a decrease in accuracy with respect to DPI techniques, which is why statistical-based algorithms have not evolved to commercial products yet. Still, researchers claim that in the near future operators will be willing to pay the cost of a few errors for a much lighter classification process.

We can further divide this class of algorithms in a few subclasses according to the data mining techniques employed and to the protocol layer of the features used. Concerning the first criterion, on one hand unsupervised clustering of traffic flows [124] (e.g. by means of the K-means algorithm) does not require training and allows to group flows with similar features together, possibly identifying novel unexpected behaviors; on the other hand, supervised machine learning techniques [105, 177] (e.g. based on Naive Bayes, C4.5 or Support Vector Machines) need to be trained with already classified flows, but are able to provide a precise labeling of traffic. Regarding the protocol layer, we have classifiers employing only flow-level features [130] (e.g., duration, total number of bytes transferred, average packet-size), as opposed to algorithms using packet-level features [34, 63] (e.g. size and direction of the very first packets of a flow). The former ones are usually capable of late (in some cases only *post-mortem*), coarse-grained classification, whereas the latter ones can achieve early, fine-grained classification.

Finally, **Behavioral classification** [101, 102, 184] moves the point of observation further upwards in the network stack, and looks at the whole traffic received by a host in the network. By the sole examination of the generated traffic patterns (e.g., how many hosts are contacted, with which transport layer protocol, on how many different ports, etc.) behavioral classifiers try to identify the application running on the target host. The idea is that different applications generate different patterns: for instance, a P2P host will contact many different peers typically using a single port for each host, whereas a Web server will be contacted by different clients with multiple parallel connections.

Some works [101, 184] characterize the pattern of traffic at different levels of detail (e.g. social, functional and application) and employ heuristics (such as the number of distinct ports contacted, or transport-layer protocols used) to recognize the class of the application running on a host (i.e. P2P vs HTTP). Works taking the behavioral approach to its extreme analyze the graph of connections between endpoints [91, 98] showing that P2P and client-server application generate extremely different connection patterns and graphs. They prove also that such information can be leveraged to classify the traffic of these classes of services even in the network core. A second group of studies [32, 80], instead, propose some clever metrics tailored for a specific target traffic, with the purpose of capturing the most relevant properties of network applications. Combining these metrics with the discriminative power of machine learning algorithms yields extremely promising results. The Abacus classifier, a major contribution of this thesis, belongs to this last family of algorithms, and it is the first algorithm able to provide a fine-grained classification of P2P applications.

Behavioral classifiers have the same advantages of statistical-based classifiers, being lightweight and avoiding access to packet payload, but are usually able to achieve the same accuracy with even less information. Such properties make them the perfect candidate for the most constrained settings. Moreover given the current tendency toward flow-level monitors such as NetFlow [56], the possibility to operate on the sole basis of behavioral characteristics is a very desirable property for classifiers.

We wrap up this overview with an overall consideration on the applicability of classifiers. With few exceptions such as [74], the wide majority of the classification algorithms proposed in literature cannot be directly applied in the network core. Limitations can be either intrinsic to the *methodology* (e.g., behavioral classification typically focuses on endpoint [184] or end-hosts [102] activity), or be tied to the *computational complexity* (e.g., DPI [77, 119, 132, 165] cannot cope with the tremendous amount of traffic in the network core), or to *state scalability* (e.g., flow-based classification [124, 130] requires to keep a prohibitive amount of per-flow state in the core), or to *path changes* (path instabilities or load balancing techniques can make early classifications techniques such as [34, 63] fail in the core). At the same time, we point out that classifying traffic at the network ingress point is a reasonable choice for ISPs: indeed, traffic can be classified and tagged at the access (e.g., DiffServ IP TOS field, MPLS, etc.), on which basis a differential treatment can then be applied by a simple, stateless and scalable core (e.g., according to the class of application.). We investigate deeper this issue in the second part of this dissertation.

Finally we must deal with a transversal aspect of traffic classification. The heterogeneity of approaches, the lack of a common dataset and of a widely approved methodology, all contribute to make the comparison of classification algorithms a daunting task [163]. In fact, to date, most of the comparison effort has addressed the investigation of different machine learning techniques [34, 71, 177], using the same set of features and the same set of traces. Only recently, a few works have specifically taken into account the comparison problem [47, 105, 116, 135]. The authors of [135] present a qualitative overview of several machine learning based classification algorithms. On the

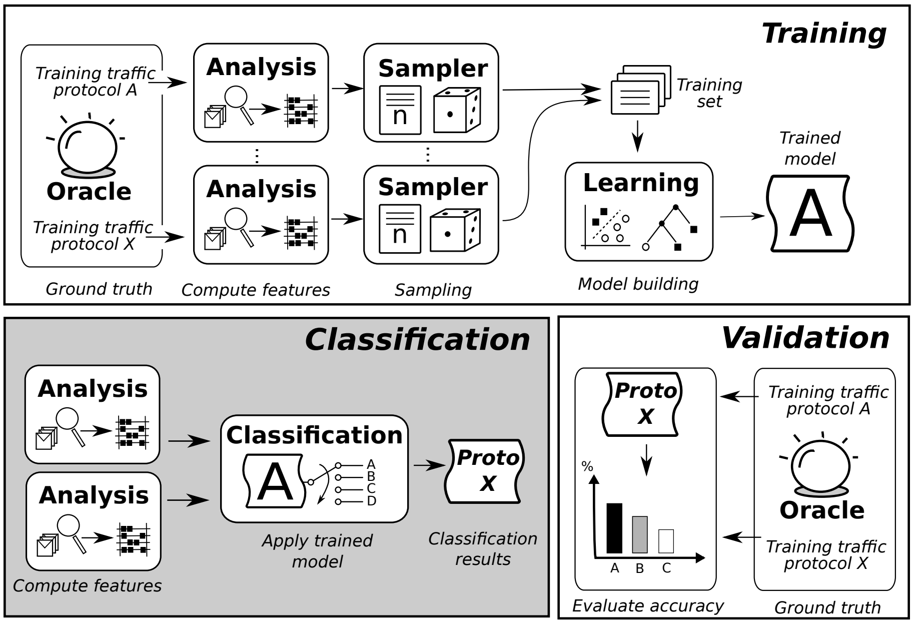


Figure 2.1: Common workflow of supervised classification.

other hand, in [105] the authors compare three different approaches (i.e., based on signatures, flow statistics and host behavior) on the same set of traces, highlighting both advantages and limitations of the examined methods. A similar study is carried also in [116], where authors evaluate spatial and temporal portability of a port-based, a DPI and a flow-based classifier. In Chap. 5 we will compare our behavioral solution, Abacus, with a packet-based classifier contrasting both their accuracy and their requirements in terms of computational power.

## 2.2 Machine learning algorithms for classification

In this section we will briefly introduce the problem of classification in machine learning theory, with a particular focus on the algorithms we actually employed in the remainder of this work, all falling in the category of *supervised classification*.

There is a whole field of research on machine learning theory which is dedicated to supervised classification [107], hence it is not possible to include a complete reference in this thesis. Moreover, instead of improving the classification algorithms themselves, we rather aim at taking advantage of our knowledge of network applications to identify good properties for their characterization. However, some basic concepts are required to correctly understand how we applied machine learning to traffic classification.

A supervised classification algorithm produces a function $f$, *the classifier*, able to associate some input data, usually a vector $\mathbf{x}$ of numerical attributes $x_i$ called *features*, to an output value $c$, the class label, taken from a list $C$ of possible ones. To build such a mapping function, which can be arbitrary complex, the machine learning algorithm needs some examples of already labeled data, the *training set*, i.e. a set of couples $(\mathbf{x}, c)$ from which it *learns* how to classify new data. In our case the features $x_i$ are distinctive properties of the traffic we want to classify, while the class label $c$ is the application associated to such traffic.

From a high-level perspective, supervised classification consists of three consecutive phases which are depicted in Fig. 2.1. During the *training phase* the algorithm is fed with the training set which contains our reference data, the already classified training points. The selection of the training points is a fundamental one, with an important impact on the classifier performance. Extra care must be taken to select enough representative points to allow the classifier to build a meaningful model; however, including too many points is known to degenerate in *overfitting*, where a model is too finely tuned and becomes “picky”, unable to recognize samples which are just slightly different from the training ones.

Notice that, preliminary to the training phase, an *oracle* is used to associate the protocol label to the traffic signatures. Oracle labels are considered accurate, thus representing the *ground truth* of the classification. Finding a reliable ground truth for traffic classification is a research topic on its own, with not trivial technical and privacy issues and was investigated by a few works [64, 82]. Sec. 2.4 is dedicated to the description of the different datasets with the related ground-truth used throughout this thesis.

The second step is the *classification phase*, where we apply the classifier to some new samples, the *test set*, which must be disjoint from the training set. Finally a third phase is needed to *validate* the results, comparing the classifiers outcome against the reference ground truth. This last phase allows to assess the expected performance when deploying the classifier in operational networks. The next section in this chapter is dedicated to describe the metrics used to express the performance of classification algorithms.

In this thesis we used mainly two supervised classification algorithms, namely *Support Vector Machines* and *Classification trees*, that we briefly describe in the following.

### 2.2.1 Support Vector Machine

Support Vector Machine (SVM), first proposed by Vapnik [60], is a binary supervised classification algorithm which transforms a non-linear classification problem in a linear one, by means of what is called a “kernel trick”. In the following we intuitively explain how SVM works and refer the reader to [62, 177] for a more formal and complete description of the algorithm.

SVM interprets the training samples as points in a multi-dimensional vector space, whose coordinates are the components of the feature vector $\mathbf{x}$. Ideally we would like to find a set of surfaces, partitioning this space and perfectly separating points belonging to different classes. However, especially if the problem is non-linear, points might be spread out in the space thus describing extremely complex surface difficult, when not impossible, to find in a reasonable time. The key idea of SVM is then to map, by means of a kernel function, the training points in a newly transformed space, usually with higher or even infinite dimensionality, where points can be separated by the easiest surface possible, an hyperplane. In the target space, SVM must basically solve the optimization problem of finding the hyperplane which (i) separates points belonging to different classes and (ii) has the maximum distance from points of either class. The training samples that fall on the margin and identify the hyperplane are called *Support Vectors*.

At the end of the training phase SVM produces a model, which is made up of the parameters of the kernel function and of a collection of the support vectors describing the partitioning of the target space. During the classification phase, SVM simply classifies new points according to the portion of space they fall into, hence classification is much less computationally expensive than training. Since natively SVM is a binary classifier, some workaround is needed to cope with multi-class classification problems. The strategy adopted along this thesis is the *one-versus-one*, where a model for each pair of classes is built and the classification decision is based on a majority voting of all binary models.

Support Vector Machines have proved to be an effective algorithm yielding good performance out-of-the-box without much tuning, especially in complex feature spaces, and has showed partic-

Table 2.2: Example of confusion matrix.

| | | Prediction outcome | |
|---|---|---|---|
| | | P | N |
| Actual | P | True Positive | False Negative |
| Value | N | False Positive | True Negative |

ularly good performance in the field of traffic classification [105, 177]. Several kernel functions are available in literature, aspect we partially explore in Chap. 4. When not explicitly stated, we employ the Gaussian kernel which usually exhibits the best accuracy. The only drawback of SVM is that models in the multidimensional space cannot be interpreted by human beings and it is not possible to really understand the reason why a model is good or bad. In our experiments, we make use of `LibSVM` implementation [50] of SVM.

### 2.2.2 Decision Trees

Decision Trees [106] represent a completely orthogonal approach to the classification problem, using a tree structure to map the observation input to a classification outcome. Again, being this a supervised classification algorithms, we have the same three phases: training, testing and validation.

During the training phase the algorithm builds the tree structure from the sample points: each intermediate node (a.k.a. split node) represents a branch based on the value of one feature, while each leaf represents a classification outcome. The classification process, instead, consists basically in traversing the tree from the root to the leaves with a new sample, choosing the path at each intermediate node according to the criteria individuated by the training phase. Like SVM, the classification process is way more lightweight than the learning phase. One big advantage of this algorithm over Support Vector Machines is that the tree can be easily read and eventually interpreted to understand how the algorithms leverages the features for the classification.

Literature on this subject contains quite a few decision tree building algorithms, which differ in the way they identify the feature and threshold value for the intermediate split nodes. In this work we employ C4.5 algorithm [106], which bases such selection on the notion of *Information Gain*. This is a metric from information theory which measures how much information about the application label is carried by each features, or, in other words, how much the knowledge of a feature tells you about the value of the label. We delay a formal definition of the information gain metric to the next chapter, where we take advantage of it for feature selection purposes. After calculating the information gain of each feature for the training set points, C4.5 picks as splitting feature for each node the one which maximizes such a score: this strategy of using the most helpful attributes at each step is particular efficient, yielding rapidly converging classification trees.

## 2.3 Evaluating classification accuracy

There are several ways of expressing the performance of classifiers, ranging from fine reporting of the classification outcome, to coarse general indexes of performance.

A common tool to represent the results of a classifier validation is the *confusion matrix*, like the example of Tab. 2.2. We will heavily use such representation in this thesis, so it is worth spending a few words to better understand it. Each row corresponds to the actual class of an test instance, while each column corresponds to a classification outcome: it is then easy to see that points falling on the diagonal represent right classification decisions, while points falling outside are classification errors.

Technically speaking, we have a specific terminology to denote the possible results of the validation process, reported in the table and explained below.

**True Positive (TP)** classifications, i.e., number of tests for which the classifier identifies the correct label of the sample point.

**False Negative (FN)** classifications, i.e., number of points of a class that are wrongly assigned to another class.

**False Positive (FP)** classifications, i.e., number of sample points that are wrongly assigned to a class, despite belonging to another one.

**True Negative (TN)** classifications, i.e., number of points correctly recognized as not belonging to a given class.

As we already mentioned above, TPs ad TNs represent correct classification decisions while FP and FN are classification errors. The goal of a good classifier is to maximize the first two and minimize the last two values. It is a common practice to normalize such metrics and write them in form of percentages, which are more intuitively interpreted and easily compared. In particular, we define a few ratios:

**True Positive Rate (TPR)** or *Recall*, the ratio of the TPs over all classification for a class.
$TPR = \frac{TP}{TP+FN}$

**True Negative Rate (TNR)** or *Specificity*, similar to the previous metric, but related to TNs.
$TPR = \frac{TN}{TN+FP}$

**Precision** is the ratio of TPs over all samples classified as a class.
$precision = \frac{TP}{TP+FP}$

**Accuracy** is the ratio of correct classifications over all classification attempts.
$accuracy = \frac{TP+TN}{TP+FP+TN+FN}$

These ratios evaluate different aspects of a classification engines. The first two indicate the ability of the classifier of associating the correct label to a sample of a given class. The third is the probability that the classification is correct provided that a given label has been selected. Finally the last metric is an overall measure of correctness of the classifier.

While in most application of classifiers these metrics are enough to describe the performance of the algorithms, in traffic classification we need to be aware of another aspect. Often classifiers, especially statistical and behavioral ones, take decisions about a network *flow*, i.e., a sequence of packets sharing some common characteristics – usually the 5-tuple composed of the two IP addresses, the two transport-layer ports and transport layer protocol type. However, since the studies of authors of [143], it is well known that in the Internet a few elephant flows carry most of the traffic, while a large number of mouse flows represent just a small portion of the overall volume of traffic. From this observation, authors of [72] demonstrate that using only flow accuracy to evaluate the performance of a traffic classification algorithm may yield to erroneous conclusions. Whenever possible it is advisable to complement this metric with *byte accuracy*, which measures the amount of bytes correctly classified. Network operators are usually far more interested in such an index, as they want to be sure that the bulk of traffic is correctly classified and, instead, do not care that much if a few mouse flows are misclassified.

## 2.4 Overview of dataset

As mentioned in the previous sections, finding good datasets for the experimental evaluation of the accuracy of traffic classifiers is quite a difficult task. Actually there is a trade-off between two factors: the representativeness of traces and the accuracy of the ground-truth. On one hand, we would like to test the algorithm on real-life traces, passively captured in operational networks, in order to gather a realistic estimation of the accuracy of a classifier when deployed in realistic scenarios. Moreover, even when considering real traffic traces, performance of the classifiers can be affected by the scenario (e.g, corporate and residential networks have very different traffic mixes), so the more variety is included in the dataset the more trustworthy results are. On the other hand, for this kind of traces it is difficult to define the ground-truth, i.e. the actual protocol to which packets belongs: either we must rely on another classifiers output (e.g. DPI) and assume it is correct, or special ad-hoc systems must be employed, for instance like the `gt` tool [82], which uses information gathered from the operating system of the machine about the process running on a given socket, in order to tag packets at the time of the capture. The opposite solution is to use active methodologies and generate controlled traffic in a testbed: in this way there is no doubt on the application generating the packets since you control the machine, yet this traffic is hardly representative of real world traffic, unless particular cautions are taken.

Moreover, there are also non technical issues [27], which further complicate collection and sharing across the community of datasets for traffic classification. The most prominent issue is privacy: not only do IP addresses and host names already convey important information, but above all packet payloads may contain sensible data (e.g. email content). While anonymizing techniques can easily solve the problem of network identifiers, dealing with the payload is much more difficult: on the one hand, removing it altogether is not a solution, as it will prevent payload based techniques from working; on the other, scanning the content for all sensible data is a very cumbersome task. The second issue impacts operators instead of users. In fact packet traces may allow to deduce much information on the network infrastructure, which the provider may not be so incline to share with others. Consequently researchers are required to sign non disclosure agreement (NDA) to access traces from operators, and usually cannot share such traces with others.

In this thesis we use an extremely heterogeneous set of traces, gathered in different environments and times, whose ground-truth has been determined using all of the aforementioned methods. Actually our dataset has evolved during the thesis, as novel tools and traces were made available to us (e.g the `gt` tool and a set of traces captured with this methodology from University of Brescia [19] appeared in 2009, halfway through the period of this work). As we proceeded in this work and our methods evolved, so did our dataset, to which we tried to add data whenever possible to improve its representativeness. For this reason it was not possible to uniform the dataset all over this thesis, considering also that the order in which our experiments are presented along this dissertation does not necessarily coincide with the order in which they were performed.

Therefore we decided to give an overview of the whole dataset in the following. For each trace we will report the most important characteristics, in particular the way it was collected and how the ground-truth was determined. However in each chapter we also punctually precise the actual composition of the dataset used for the experiments, along with the most important features for the evaluation taking place (e.g. in Chap. 6 where we classify using NetFlow flow-records, we are interested in the number of flow-records for each trace, while in all other chapters this information is irrelevant).

**P2P-TV testbed traces**

To capture P2P live-streaming traffic, we resorted to a large testbed involving multiple measure-

Table 2.3: Overview of the different traces used for the experiments presented in this thesis.

| **Name** | Napawine Testbed | Italian ISP | Campus | University of Brescia | Auckland |
|---|---|---|---|---|---|
| **Year** | 2008 | 2006,2007 | 2008 | 2009 | 2001 |
| **Methodology** | Active | Passive | Passive | Passive | Passive |
| **Public** | Yes | No | No | Yes | Yes |
| **Ground-truth** | Testbed | DPI | DPI | `gt` | port-based |
| **Applications** | P2P-TV (PPLive, TVAnts, SopCast, Joost) | Web, Mail, P2P (eDonkey, Skype) | Web, Mail, Skype | Web, Mail, P2P (Bittorrent, eDonkey, Skype) | Web, Mail |

ments points. The testbed was setup in May 2008 in the context of NAPA-WINE, a 7th Framework Programme project funded by the EU [115]. The testbed involved more than 30 controlled peers, hosted at 7 different Institutions, scattered over 4 European countries Details concerning the experiments are reported in Tab. 2.4. During the experiments, each PC ran the same application for one hour, during which all involved peers were forced to watch the same channel at the same time. SopCast, TVAnts, PPLive and Joost were run: in all (but Joost) cases, the nominal stream rate was 384 kbps ($\sim$550 kbps) and Windows Media 9 Encoder was used. PCs were synchronized via NTP and MS Windows scheduler was used to automatically start the application and tune to the selected channel. Each PC captured the packet-level traces for the whole duration of the experiment (no packet sampling or loss occurred, and broadcast/multicast packets were filtered out).[2] All details about these traces are provided in [54].

Moreover, as different network setups (e.g., access technologies, security policies, private/public addresses, etc.), different content (popular vs unpopular channels) and peers configurations (hardware, OS) were all part of the testbed, we are confident that the heterogeneity of the dataset is representative of a wide range of scenarios.

Being composed by testbed traces with a very reliable ground-truth, this dataset will be used throughout this work, especially in Chap. 4, where we present the Abacus classifier, specifically targeting this kind of applications. In the other chapters we normally use a subset of this large dataset, to avoid using an unbalanced mix of traffic (i.e., to much data for P2P live streaming application with respect to other services).

In Chap. 4, we also use two other datasets containing P2P-TV traffic actively collected in an ad hoc testbed by other researchers, and kindly made available to the community. Traces of [167] were collected in July 2006 during the Fifa World Cup. Traces of [26] were instead collected in 2008 in a testbed, where changing network conditions were artificially enforced. In particular, in these experiments, a Linux router was used to emulate some network conditions: (i) bandwidth, (ii) delay and (iii) packet losses were imposed on the downlink path to the PC running the P2P-TV application. To avoid going out of topic, we refer the reader to [26] for a complete description of the testbed: we only point out that impairments ranged from mild to very tough conditions (e.g., 200 Kbps of available downlink bandwidth, delay up to 2 s and packet losses up to 40%).

### Italian ISP traces

This set is composed by two 1-hour long traces collected in 2006 and 2007 from one of the main broadband ISP in Italy (which we cannot cite due to NDA) which offers triple-play services

[2] Traces differ because during the experiment some application could not successfully run, e.g., due to peer failure, or bad network condition.

Table 2.4: Summary of the hosts, sites, countries (CC) and access types (NAT=Network Address Translation, FW=Firewall) of the peers involved in the testbed.

| Host | Site | CC | Access | NAT | FW |
|---|---|---|---|---|---|
| 1-4 | BME | HU | high-bw | - | - |
| 5 | | | DSL 6/0.512 | - | - |
| 1-9 | PoliTO | IT | high-bw | - | - |
| 10 | | | DSL 4/0.384 | - | - |
| 11-12 | | | DSL 8/0.384 | Y | - |
| 1-4 | MT | HU | high-bw | - | - |
| 1-3 | FT | FR | high-bw | - | - |
| 1-4 | ENST | FR | high-bw | - | Y |
| 5 | | | DSL 22/1.8 | Y | - |
| 1-5 | UniTN | IT | high-bw | - | - |
| 6-7 | | | high-bw | Y | - |
| 8 | | | DSL 2.5/0.384 | Y | Y |
| 1-8 | WUT | PL | high-bw | - | - |
| 9 | | | CATV 6/0.512 | - | - |

over an all-IP architecture to more than 5 millions of users. The ISP network is representative of a very heterogeneous and uncontrolled scenario, in which customers are free to use the network without restriction. Traffic is sniffed at a PoP level, to which around 500 users are connected, using more than 2000 different IP addresses considering VoIP phones, set-top-boxes and PCs. This set is representative of a very heterogeneous scenario, in which no traffic restriction are applied to customers. Ground-truth for these traces has been determined by means of an DPI tool [16]. Relevant protocols we extract from this trace are eDonkey for file-sharing, Skype for VoIP and DNS for traditional client-server traffic. However, in the following chapters we mostly use this trace for background traffic, i.e. to test whether a classifier is able to recognize traffic for which it was not originally trained.

**Campus traces**

This is a 1-days long trace, collected during one working week at the edge router of Politecnico di Torino LAN, which is representative of a typical data connection to the Internet. The LAN contains about 7000 hosts, whose users can be administrative people, faculty members and students. Most of the traffic is due to TCP data flows carrying web, email and bulk traffic, since a firewall blocks all P2P file sharing applications. Again ground-truth for this scenario has been determined by means of an DPI tool [16]. Similarly to the ISP trace, we use this set to evaluate the ability of the classifier to identify "unknown" traffic. All details about these traces are available in [125].

**University of Brescia traces**

This is a set of 3 traces captured during 3 working-days in 2009 by colleagues at University of Brescia on the 100Mb/s link connecting their campus network to the Internet. This dataset, of which we only use the largest trace, is publicly available in anonymized form [18]. The ground-truth is extremely reliable for this set as it was collected using the `gt` tool [82], which uses information gathered from the machine operating system to label packets during capture.

**Auckland traces**

This is a public available trace [2], collected during 4.5 days in 2001 at the University of Auckland, of which we extract the initial 8hr busy-day period only. This trace is used in Chap. 7, to evaluate the performance of classification under packet-sampling. In this case we want to have as a heterogeneous dataset as possible, to gather robust results. The ground-truth was determined using a simple port-based classifier, given that in 2001 such a classification method was still reliable.

# Chapter 3

# A general framework for behavioral P2P traffic classification

In the previous chapter, we have seen that the research community has recently proposed behavioral classifiers to face the new challenges of traffic classification, such as high traffic volumes, encryption and obfuscation. Since they base the classification on the peculiar pattern of traffic generated by a host, such classifiers are extremely lightweight and accurate even though using only flow-level traffic properties, like the ones usually provided by NetFlow [57].

However, the research community has produced a rather fragmentary work so far: a few very specific approaches [80, 97, 158, 173, 180] have been proposed, moreover evaluated on specific datasets. Therefore, the community lacks a broad view of the relative importance, in the context of traffic classification, of any feature that can be defined over IPFIX flow-level data. Similarly, as considering a single dataset can bias the evaluation, a careful analysis should explicitly take into account the relative stability of the feature expressiveness over multiple network scenarios.

This chapter, containing results in part published in [160], proposes a comprehensive framework for the definition of behavioral features which can be defined over IPFIX-records. Our aim is to provide the community with a reference as complete as possible of all behavioral features suited for P2P traffic classification, similarly to what has been done in [129] with flow-level features, but with the additional constraint that such features should be fully compliant with IPFIX records. By clearly stating the criteria that guide our definition, we are able to thoroughly explore the space of features and define a long list of potentially expressive characteristics. The resulting framework is general enough to include features of existing classifiers [80, 141, 180, 184, 184], enabling their evaluation as well.

Moreover, we quantify the amount of information contained in the defined features and exploitable for the classification of P2P traffic. Our analysis is close to what has been done by the authors of [75], which focused on the stability of the information carried by traffic flows at the *packet-level*. However, we want rather to assess the stability of behavioral features computed at the *flow-level*.

In our experiments we compute the behavioral features over packet-level traces containing traffic from different kinds of P2P applications (P2P-TV, VoIP, file-sharing) and representative of diverse network scenarios. Then we employ two different metrics to evaluate their usefulness for classification purposes, i.e. *Information Gain and ReliefF*. After ranking the feature according to their relevance for the classifier, we actually perform the classification employing C4.5 decision trees.

The remainder of this chapter is organized as follows. Sec. 3.1 introduces the framework for the definition of behavioral features and clearly defines the feature we use in the experimental part. Sec. 3.2 describes the methodology and metrics used to evaluate the information content of features, while Sec. 3.3 presents the results of our experiments.

## 3.1 Defining behavioral features

In this section we will describe our framework for the definition of behavioral features for traffic classification. In order to perform a systematic exploration of the feature space, we first introduce a series of criteria, described in the following, to guide the feature definition. We find a good mapping between features used by existing classifiers and our framework, which proves the generality of our approach. While we want to keep our framework as general as possible in its definition, in the experimental part we actually restrict our attention on a smaller subset of the possible features, which is listed in Tab. 3.1 and detailed at the end of this section.

Our classification target is a peer, identified as a socket (or an aggregate of sockets) running on a host. As this peer contacts other peers and exchanges information with them, we suppose that an IPFIX monitor at the edge of the network produces records for all the traffic related to the host. Our framework takes this data as input and derives the features to be used for the classification.

### 3.1.1 Timescale

This criterion refers to the temporal duration of the periodical statistics collection, thus dividing time in subsequent time-slots in which features are computed. Observation timescale is subject to the following tradeoff. On the one hand, we would like $T$ to be as small as possible, to support *early classification* for tasks like QoS verification, security and lawful interception. On the other hand, we would like $T$ to be as large as possible for lightweight operation, which would however limit possible applications of the classification to *post-mortem analysis* (e.g accounting, monitoring). Coherently with this requirements, values used in literature range from $T = 5\,s$ in [32] up to $T = 5$ minutes in [180].

Current IPFIX implementations impose further constraints on the choice of $T$, as they dump statistics on active flows every $T = 30$ minutes, with a configurable minimum of $T = 1$ minute. To have a finer timescale, however, one could use custom implementations on dedicated high-profile device such as Endace [4] or AITIA [1] cards.

### 3.1.2 Entities

We can define the entities involved in a P2P system at different network layers, which in their turn correspond to different levels of traffic aggregation. In fact, a peer can be identified either at L3 by its IP address, at L4 by its port number, or at the endpoint-level by the combination of IP and port.

This is better explained with the help of a simple example. Consider an application running on a host $IP_x$, receiving all traffic on a single socket on port $p_x$ of L4-protocol type $PT \in \{TCP, UDP\}$. By focusing on different network layers, we can identify the following different entities $E$

- At the endpoint-level, $E(y) = IP_y : p_y$, by aggregating all IPFIX records $PT : IP_y : p_y : IP_x : p_x$
- At the L3 host-level, $E(y) = IP_y$, by aggregating all IPFIX records $PT : IP_y : * : IP_x : p_x$
- At the L4 port-level, $E(y) = p_y$, by aggregating all IPFIX records $PT : * : p_y : IP_x : p_x$

Basically *endpoint-level* entities correspond to single flows, and have been used in [32, 158, 184]. The other entities, instead, decouple L3 from L4. *Host-level* aggregation, found for instance in [180, 184], may be useful in cases where an application runs multiple sockets (e.g., aggregating several client TCP connections using ephemeral ports, or several UDP sockets with different functions, such as data or signaling). *Port-level* aggregation, instead, might help in evaluating how an application uses the port space (e.g., by using several different random ports, or a single deterministic port). This has been shown to be a good discriminator in [154, 184]. Notice also that, as recently underlined in [105], the port number itself may still be a helpful feature.

### 3.1.3 Granularity and Direction

Since IPFIX records provide counters with different granularities, a trivial criterion regards the level of coarseness of the statics: features can be computed entity-wise $E$, packet-wise $P$ and byte-wise $B$.

Another intuitive criterion consists in discriminating *incoming* versus *outgoing* traffic, or aggregating both directions together. Notice that the adopted type of transport layer protocol can cause significant difference in the pattern of traffic observed in the two directions. For example, an application using a connectionless service (i.e. a UDP datagram socket) can easily multiplex all incoming and outgoing traffic over the same endpoint $IP_y : p_y$. Conversely an application employing a connection oriented service (i.e., a TCP stream socket) is likely to receive traffic on a single TCP port ($p_y$), but it surely spreads the outgoing traffic on different ephemeral ports, whose allocation is controlled by the OS.

### 3.1.4 Categories

The entities involved in the communication with the target peer can be further categorized according to different properties. In more detail, we define some rules $C$ to partition the set of entities $\mathcal{S} = \mathcal{S}_C \cup \overline{\mathcal{S}}_C$. Although in principle the subsets do not need to be disjoint, we believe that requiring $\mathcal{S}_C \cap \overline{\mathcal{S}}_C = \emptyset$ induces more clarity and simplifies the collection of the statistics. We can envisage a number of different properties, related to either the *spatial* or *temporal* domain.

Let us focus on the *spatial* category first. P2P applications offer services built on top of an overlay network which needs to be continuously maintained to handle peers churn. Thus, traffic can roughly be divided in either data or signaling traffic. We consider *contributing* or *data* entities $E_d$, peers sending or receiving a number of bytes larger than a given threshold. More formally, indicating with $B_y$ the amount of bytes exchanged with entity $E(y)$, we have $\mathcal{S}_d = \{E(y) : B_y > \beta\}$. Unfortunately, the choice of a proper threshold is not trivial, as it has been shown that good values might be application dependent [154]. However, in our experiments, consistently with [86], we use a value $\beta = 12\,KB$.

We now move to the *temporal* properties. Consider the set $\mathcal{S}_i$ of entities observed at the $i$-th slot, i.e., $t \in [iT, (i+1)T))$. By comparing $\mathcal{S}_i$ with the previous slot $\mathcal{S}_{i-1}$, we can define the set of *new* entities as $S_n = \mathcal{S}_i \backslash \mathcal{S}_{i-1}$, i.e. the set of peers discovered in the current timeslot. A similar distinction can be found in many works on P2P traffic analysis [54, 154], or P2P traffic classification [180].

Although for our evaluation we just consider the above two rules, it is worth mentioning a few other partitions related to the temporal domain. For instance we can define the set of $k$-persistent neighbors as the set of peers that have been seen in (at least) $k$ consecutive rounds $\mathcal{S}_{k-per} = \cap_{j=i}^{i-k} \mathcal{S}_j$. Symmetrically we could define the set of $k$-recurring peers, i.e., the peers that are to be found in the current and in the $k$-th previous slot, as $\mathcal{S}_{k-rec} = \mathcal{S}_i \cap \mathcal{S}_{i-k}$. Besides, many

Table 3.1: List of P2P traffic features used in the experiments, for a single direction and a single timescale.

| | **Categories** | | |
|---|---|---|---|
| **Operation** | All | New Peers | Data Peers |
| $O(\cdot)$ | $\mathcal{S}$ | $\mathcal{S}_n$ | $\mathcal{S}_d$ |
| **None** | $E$ (entities) | $E_n$ | $E_d$ |
| $O(x) = x$ | $P$ (packets) | $P_n$ | $P_d$ |
| | $B$ (bytes) | $B_n$ | $B_d$ |
| **Difference** | $\Delta_t(E)$ | $\Delta_t(E_n)$ | $\Delta_t(E_d)$ |
| $O(x,t) = x_{t-1} - x_t$ | $\Delta_t(P)$ | $\Delta_t(P_n)$ | $\Delta_t(P_d)$ |
| | $\Delta_t(B)$ | $\Delta_t(B_n)$ | $\Delta_t(B_d)$ |
| **Breakdown** | - | $E_n/E$ | $E_d/E$ |
| $O(x_{cat}, x) = x_{cat}/x$ | - | $P_n/P$ | $P_d/P$ |
| | - | $B_n/B$ | $B_d/B$ |
| **Ratio** | $P/E$ | $P_n/E_n$ | $P_d/E_d$ |
| $O(x,y) = x/y$ | $B/E$ | $B_n/E_n$ | $B_d/E_d$ |
| | $B/P$ | $B_n/P_n$ | $B_d/P_d$ |
| **Average** | $\mathrm{E}[\mathbf{P}]$ | $\mathrm{E}[\mathbf{P}_n]$ | $\mathrm{E}[\mathbf{P}_d]$ |
| $O(\mathbf{x}) = \mathrm{E}[\mathbf{x}]$ | $\mathrm{E}[\mathbf{B}]$ | $\mathrm{E}[\mathbf{B}_n]$ | $\mathrm{E}[\mathbf{B}_d]$ |
| | $\mathrm{E}[\mathbf{B}/\mathbf{P}]$ | $\mathrm{E}[\mathbf{B}_n/\mathbf{P}_n]$ | $\mathrm{E}[\mathbf{B}_d/\mathbf{P}_d]$ |
| **Standard Deviation** | $\mathrm{Std}[\mathbf{P}]$ | $\mathrm{Std}[\mathbf{P}_n]$ | $\mathrm{Std}[\mathbf{P}_d]$ |
| $O(\mathbf{x}) = \mathrm{Std}[\mathbf{x}]$ | $\mathrm{Std}[\mathbf{B}]$ | $\mathrm{Std}[\mathbf{B}_n]$ | $\mathrm{Std}[\mathbf{B}_d]$ |
| | $\mathrm{Std}[\mathbf{B}/\mathbf{P}]$ | $\mathrm{Std}[\mathbf{B}_n/\mathbf{P}_n]$ | $\mathrm{Std}[\mathbf{B}_d/\mathbf{P}_d]$ |

more complex presence indicator such as those defined in [153] could be evaluated.

### 3.1.5 Operations

Finally, a variety of computations can be performed on the gathered counters, ranging from very simple to rather complex operations. As examples of the latter, in [32] and in the next chapter of this thesis, a probability mass function is built starting from the counts of packets and bytes exchanged by the target peer with the other entities. In [80], instead, the Autocorrelation function (ACT) and the discrete Fourier transform (DFT) is applied to the time series of entity counts, of data rates exchanged with a given entities, and of start and end time of flows.

However, in this preliminary analysis, we limit ourselves too the following few simple operations $f(\cdot) : \mathbb{N}^m \to \mathbb{R}$, that produce a single scalar value.

- *None*, use the raw count as feature.
- *Temporal difference* with respect to the previous slot (e.g., the rate at which the number of packets received is changing $\Delta_t(P) = P_t - P_{t-1}$)
- *Category breakdown* per-category breakdown (e.g., the percentage of *new* entities $E_{new}/E$).
- *Ratio* of different counters for a given entity (e.g., $B_y/P_y$ the mean packets size of a given entity $y$).
- *Spatial mean and standard deviation* of a counter over a set of entities (e.g., mean number of packets per peer $\mathrm{E}[\mathbf{P}]$)

Table 3.2: Summary of the dataset

| Category | Application | Packets | Bytes |
|---|---|---|---|
| P2P TV | PPlive | 7,3 M | 1,13 G |
| | Sopcat | 3,2 M | 0,45 G |
| | TVAnts | 2,4 M | 2,56 G |
| | Joost | 3,4 M | 2,14 G |
| P2P File-sharing | eDonkey | 22,4 M | 6,93 G |
| | BitTorrent | 1,4 M | 0,74 G |
| P2P VoIP | Skype | 6,1 M | 2,91 G |

Despite their simplicity, quite a few of these operations have already been successfully employed for traffic classification, for example in [101, 141, 184]. We point out that this list is clearly not exhaustive (e.g., temporal means and other statistics can naturally be defined). However, we believe that the merit of the framework is not weakened by considering, for the time being, a small but well-defined list. Moreover we argue that the features we define explores a significant portion of the space, providing valuable advice about which properties discriminate best among P2P applications, thus deserving a deeper investigation, as we do in the next chapter.

Let us clearly state here the set of features we are going to analyze in the experimental part. First of all, we decided to consider two timescales $T \in \{5, 120\}\, s$, outgoing and incoming traffic separately, and endpoint-level entities. Finally, operations and categories are summarized in Tab. 3.1. Features are organized in columns according to the category they pertain to (i.e. all, new and data entities), and in rows according to operation performed to calculate them. Notice that we use counters of all possible levels of granularity (entity, packets, bytes), along with all their meaningful combinations in computing ratios and statistical indexes. Overall the final set is composed of 102 features.

## 3.2 Methodology

### 3.2.1 Dataset

We validate our classification engine on a large set of traces of heterogeneous P2P applications, collected in different environments and whose main characteristics are reported in Tab. 3.2.

The traces containing P2P live-streaming traffic correspond to a subset of the traffic collected in the context of the experiments of the NAPA-WINE project [115], which was introduced in Sec. 2.4. BitTorrent traces were instead collected by running the official client connected to the Internet through an ADSL access. Using traces from controlled active scenarios has the advantage of providing a reliable ground truth, as there is no doubt on the application generating the traffic.

For the remaining applications, we resorted to the trace passively collected at the POP of a large Italian Internet provider, whose details were presented as well in Sec. 2.4. By running a traditional DPI classifier we were able to extract flows pertaining to eDonkey, Skype representative of P2P file-sharing and P2P VoIP. We also used the labeled trace made public by University of Brescia [19], which contains the traffic generated by faculty members and students captured at the link between the campus and the Internet. This trace has a extremely reliable ground-truth as it was capture by means of the automatic tool `gt` [82].

It can be seen that, in order to gather robust results, we tried to include in our dataset a representative sample of the whole spectrum of P2P applications. In detail, the dataset is made up of traffic from four *P2P-TV* applications (PPLive, TVAnts, SopCast, Joost [1]), two *file-sharing* applications (Edonkey, BitTorrent ) and a *VoIP* application (Skype).

### 3.2.2 Metrics

In this section we describe the procedure followed to evaluate the importance of each feature for traffic classification. We started by extracting IPFIX-records from the packet traces. Then we aggregated the records related to the relevant endpoints and compute the features. The result is a list of couples $(Y, \mathbf{X})$, where $Y$ is the application label, and $\mathbf{X}$ is the vector of features listed in Tab. 3.1, computed on both incoming and outgoing traffic. Afterwards, similarly to [75], we extracted a random subset of all data, in such a way that it finally contains the same number of samples for each application–corresponding to about 6 hours worth of traffic for each application. Besides facilitating our analysis, this process removes any bias deriving from unbalanced traffic mixture.

We proceeded with ranking our features according to their information content related to the application label. Our aim is to understand which attributes are the most useful for our classification purposes, and which ones we can neglect. Moreover when fewer features are fed to a machine learning algorithms, the classification process becomes more lightweight and fast, whereas too many features can confuse the classifier and cause the *overfitting* phenomenon.

In the following we introduce the two metrics used to evaluate the usefulness of features for our classification purposes. To calculate them we employed the `weka` [20] toolkit, which allows us not only to apply different feature selection algorithms, but also to later perform the classification and to evaluate the classifier performance.

#### 3.2.2.1 Information Gain

Information Gain [61] $I(X, Y)$ measures the reduction of the uncertainty of the class $Y$ (in our case the application label) when the value of feature $X$ is known; in other words, it evaluates how much the knowledge of $X$ tells you about the value of $Y$. Information Gain is based on the concept of *entropy* $H(Y)$ of a random variable $Y$, i.e., the amount of randomness of its distribution. In case of a discrete random value with a distribution $p(y)$, the entropy is given by the expression

$$H(Y) = -\sum_{y \in Y} p(y) \log_2 p(y)$$

The uncertainty of $Y$, when the value of $X$ is known, is given by the conditional entropy

$$H(Y|X) = -\sum_{x \in X} p(x) H(Y|X = x)$$

Finally the information Gain is the difference of the above quantities

$$I(X, Y) = H(Y) - H(Y|X)$$

This metric is commonly used for feature selection and it is also employed by the C4.5 algorithm during the tree building phase to choose the attributes for the splits at intermediate nodes. This quantity is always positive and usually measured in bit. If the two variables $X$ and $Y$ are completely unrelated, their mutual information is zero. Conversely, a feature $X$ is a perfect discriminator if the condition $I(X, Y) = H(Y)$ is met, where $H(Y)$ is the entropy of the protocol label (i.e. the number of bits needed to perfectly describe $Y$). In our case of 7 applications represented by the same number of samples we have $H(Y) = \log_2(N) \sim 2.8$.

[1] Traces date back May 2008, when Joost was still exploiting P2P for video distribution

**Input**: data set $\mathcal{D}$ of randomly selected instances, $|\mathcal{D}| = m$, attribute set $\mathcal{X}$, class label $Y$
**Output**: vector $\mathbf{W}$ of estimations of the quality of attributes

```
set all weights $W_i := 0$
**for all** instances $d \in \mathcal{D}$ **do**
  find $k$ nearest hits $H_j$
  **for all** classes $Y \neq \text{class}(d)$ **do**
    from class $Y$ find $k$ nearest misses $M_j(Y)$
  **end for**
  **for all** attributes $X \in \mathcal{X}$ **do**
    $W[X] = W[X] - \sum_{j=i}^{k} \text{diff}(X, d, H_j)/(m \cdot k) +$
        $\sum_{Y \neq \text{class}(d)} [\frac{P(Y)}{1-P(\text{class}(d))} \sum_{j=1}^{k} \text{diff}(X, d, M_j(Y))]/(m \cdot k)$
  **end for**
**end for**
```

where $\text{diff}(X, a, b) = \begin{cases} 0 & \text{if } a_X = b_X \\ 1 & \text{otherwise} \end{cases}$ for binary attributes
and $\text{diff}(X, a, b) = \frac{|a_X - b_X|}{max(X) - min(X)}$ for numerical attributes.

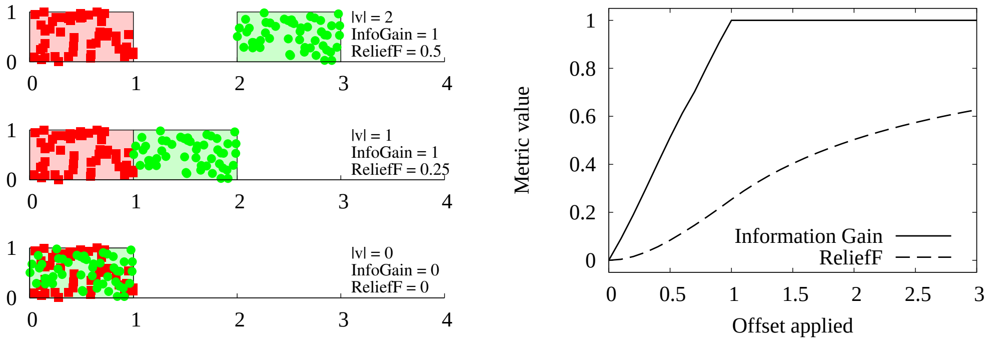
Figure 3.1: Pseudo-code for the ReliefF metric.

#### 3.2.2.2 ReliefF

The ReliefF [149] is another widely used metric for feature selection. It is a more robust generalization of the Relief metric to handle multi-class classification problems. The key idea of the algorithm is to evaluate how well an attribute can distinguish among instances close to each other. The actual algorithm is reported in Fig. 3.1 and explained in the following.

Basically for each instance $d$ in the dataset the algorithm searches the $k$ nearest misses ($M_j$) related to all other classes (i.e. the closest neighbor points with a different label), and the $k$ nearest hits $H_j$ (i.e. the closest neighbors belonging to the same class). Then, it updates the estimation of the weights of the attributes based on the values taken for the current instance $d$ and the nearest hits and misses. The rationale is the following. If an attribute assumes very different values for two close points belonging to different classes, then it is a good one, so its weight is increased proportionally to the difference between the attribute values. On the contrary, if an attribute takes very different values for two close points belonging to the same class, then it is a bad discriminator and its weight is decreased accordingly. You can see that the formula that updates the weights takes care of averaging over all the possible classes. The function diff is used to measure how much the values of a feature for two distinct instances differ between each other and it is normalized on the whole range of variation of the feature. It comes in different flavors according to the type of attribute (i.e., binary of numeric) and it is also used to evaluate the overall distance of two instances (i.e., by taking the sum of distances over each attribute).

Actually, the weights of the attributes can be interpreted as the difference of the estimated



(a) (b)

Figure 3.2: Empirical comparison of Information Gain and ReliefF metrics.

probabilities

$$\begin{aligned} W(X) = & P(\text{different value of} X | \text{nearest instance from different class}) \\ & - P(\text{different value of} X | \text{nearest instance from same class}) \end{aligned}$$

or, in other words, the probability that $X$ takes a different value when the closest point belongs to a different class minus the probability that $X$ takes a different value when the closest point belongs to the same class. As you can see, ReliefF uses a more geometric approach towards feature selection, with respect to Information Gain which draws more from information theory: we will see how this reflects in our results.

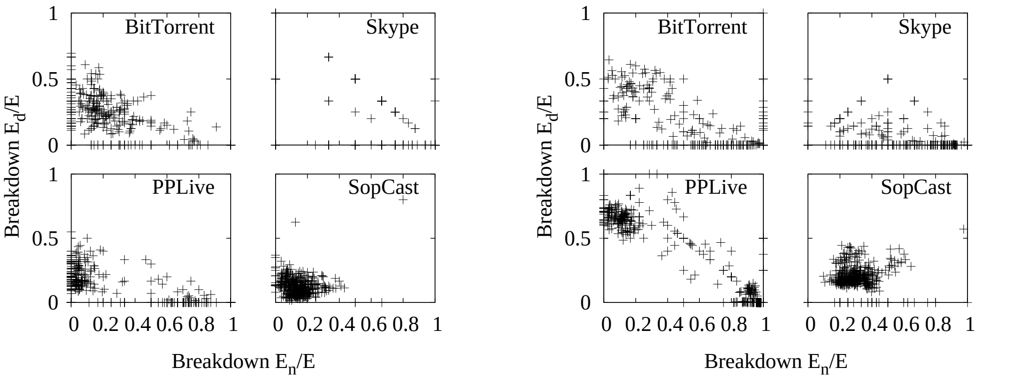
### 3.2.3 Preliminary Examples

Before presenting the results over all features, in this section we use two examples to better illustrate some properties of the features defined by our framework as well as of the metrics we used to characterize their information content.

#### 3.2.3.1 Metrics

First, we present a very simple experiment with the two metrics on a artificial dataset. We generate two sets of points on a bi-dimensional plain, with $x$ and $y$ coordinates selected at random in the interval $[0, 1]$, i.e. we pick random points in a square with a side of length one; each set of points is given a different label. Then we apply a rigid geometrical transformation to one class of points, adding an offset $\vec{v}$ to their x-coordinate. We construct several dataset, increasing the offset or, in other words, separating more and more the two classes on the x-axis: the more the points are separated the more powerful becomes the abscissa as a discriminator. In Fig. 3.2 we visually show three cases: in the bottom plot, no offset is applied and the two classes of points overlap; in the middle plot, the two squares are adjacent but no longer overlap; in the top plot, instead, points are distant and very well separated.

For each dataset we compute the Information Gain and the ReliefF for the x-coordinate, to better understand the meaning of the metrics. In Fig. 3.2-(b) we plot their values in function of the offset applied to the points. The Information Gain attains its maximum value of 1 (in this case with two classes one bit is enough to distinguish between them), as soon as two squared regions do not overlap: this makes sense as in this case the x-coordinate is enough to perfectly tell the class



(a) $T = 5\,s, R = (0.08, 0.07), I = (0.84, 0.82)$ (b) $T = 120\,s, R = (0.14, 0.16), I = (0.70, 0.46)$

Figure 3.3: Scatter plots of the breakdown of new entities versus the breakdown of data entities for four applications and two timescales. Captions report the values of Information Gain ($I$) and ReliefF ($R$).

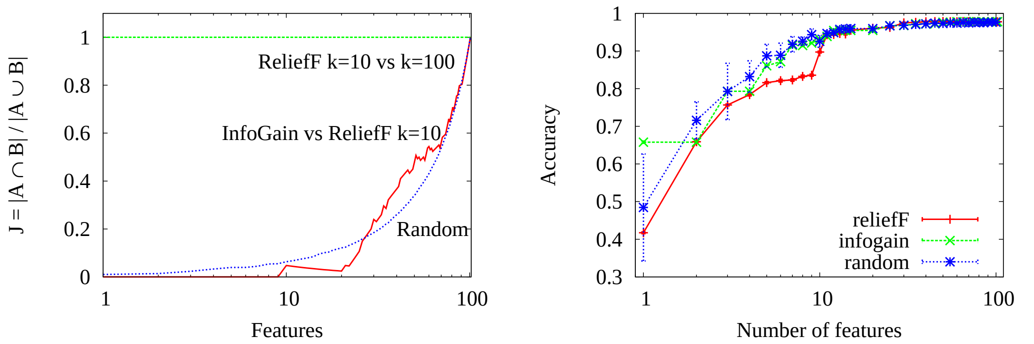
a point belongs to. The ReliefF instead shows a different behavior with a slower increase towards the value of 1, which is actually the asymptote of the curve. This comes from the geometric nature of the metric: for instance, considering the case where the two regions are adjacent to each other, it can be seen that there are still points which are very close (e.g., on the shared border of the regions) and yet belong to different classes.

#### 3.2.3.2 Features

In Fig. 3.3 we report the scatter plots of two features, namely the breakdown of new entities ($E_n/E$) on the x-axis, and the breakdown of data entities ($E_d/E$) on the y-axis, considering four applications and both timescales $T \in 5, 120\,s$. Although for clarity sake we do not represent all the 7 applications, nevertheless pictures include at least one example for each service (P2P-TV, filesharing, VoIP).

At first glance, the impression is that each application generates a distinct pattern, as points cluster in different regions of the plane. The larger time scale seems to yield better results as clouds of points appear to be very well differentiated, with few regions of intersection between each other. Notably, a bimodal behaviors of PPLive is highlighted, with two distinct clouds of points corresponding exactly to the two typical activities of a P2P application: discovering new peers (i.e. low percentage of data peers, but high percentage of new peers), and transferring data (i.e. high percentage of data peers, but low percentage of new peers). Notice, instead, that for $T = 5\,s$ the region of space near the origin, i.e. low percentage of both new and data peers, is more or less common to all applications.

We report in the captions the values of Information Gain and ReliefF for the two features. As you can see in a complex space like the one defined by our 102 features, the two metrics disagree between each other. While the ReliefF gets an higher score for $T = 120$, the Information Gain identify $T = 5$ as the best. This stems from the different nature of the two metrics, the Information Gain coming from information theory as opposed to ReliefF with a more geometrical interpretation. In three cases out of four, however, the metrics identify the feature on the x-axis (the breakdown of new entities over all entities) as the most useful, which confirms the first impression one may gather with naked eye, i.e. that points are better separated along the horizontal rather than on the vertical axis.



(a) (b)

Figure 3.4: (a) Comparison of the rankings of the features and (b) accuracy of classification in function of the number of features used

## 3.3 Experimental Results

### 3.3.1 Comparing feature ranking

Let us now extend our analysis to the complete set of features. Since results are very similar for the two timescales, we report only those gathered for $T = 5\,s$ to avoid repeating ourselves. First of all we are going to look at the ranking of features produced by the two metrics. As we have seen in the preliminary example above, the two metrics can yield almost opposite results. We first apply them separately to the whole set of features and then we compare their ranking, in order to see whether we just picked a specific feature or if the two metrics disagree in the general case.

We actually compare three rankings. First the one provided by the Information Gain metric; then, the two provided by the ReliefF metric with two different choices of the parameter $k$ (i.e., the number of neighbor hits and misses), in order to assess its eventual impact on the ordering. We used the Jaccard Index [172] $J$ as a simple metric to quantify the level of similarity of the two rankings. Let $A$ and $B$ be the sets composed of the first $n$ features as ranked by two different metrics: we calculate the ratio of the cardinality of the intersection of the two sets over the cardinality of their union

$$J = \frac{|A \cap B|}{|A \cup B|}$$

Basically, in this way, we obtain the percentage of common features over the total number of features included in the two sets. We start considering the sets containing only the first metric of each ranking, and then proceed adding one feature at a time.

We report our results in Fig. 3.4-(a). First we compare the two ReliefF rankings between each other: as the similarity score is always one, this means that they are exactly the same and that this metrics, in our case, is insensitive to the number of neighbors considered. Then we compare the Information Gain and ReliefF: the low score obtained especially for small cardinalities of the sets means that the two rankings are very unlike one another. As a reference we report the comparison of two random orderings of features (the line is actually the mean over 10 runs): in this case the score is greater than zero even for low cardinalities. Naturally, as the number of features considered increases, so does the similarity score as the probability of having features in common increases as well. In the next section we observe how this dissimilarity between the rankings influences the classification accuracy.

### 3.3.2 Classification results

In this section we finally perform the classification, focusing again on the shorter timescale of 5 s. To evaluate the contribution of each feature we used a simple strategy: as we did to compare the ranking, we perform several runs of classification, including an increasing number of features in descendant order of metric score. We repeat this procedure for each ranking separately, as well as for a random selection of features. Results in terms of overall accuracy of the classification are reported in Fig. 3.4-(b) in function of the number of features included in the set. Each point and its vertical bar correspond respectively to the average and standard deviation of ten repetitions, each time with a randomly sampled training set, which corresponds to approximately one tenth of the overall dataset.

We used the C4.5 decision (see previous chapter) tree algorithm which gave us faster execution times and, hence, the possibility of performing more experiments. Moreover, given the way the training phase operates (i.e., creating the splits on the base of the Information Gain of features), this algorithm is much more resilient to the phenomenon of overfitting, which allows us to safely add features without loosing too much accuracy. On the other hand we should pay attention when evaluating the Information Gain ranking, because our results may be biased by the fact that the same metric is used inside the classification algorithm itself.

At first glance, we see that after including ten features, whatever their original order, the classifier accuracy saturates and adding more information has no beneficial effect. Our features are particularly good despite their simplicity, given that they are able to guarantee a very good accuracy, exceeding the 98% of true positives. As expect, we see no overfitting whatsoever, for the classification algorithm cleverly discards redundant, eventually misleading, information provided by superfluous features. Yet, the most interesting behaviors can be observed for feature sets with small cardinality. Information Gain seems a better metric for our case with respect to ReliefF, since it provides better results. This is particularly evident for a single-feature set, in which case the features picked by Information Gain already gives us an accuracy of 65%.

Interestingly, it seems that selecting a random set of features yields, in average, an higher accuracy than using the sets of features provided by the two rankings. The higher standard deviation of the random curve means, however, that there is an high variability of the accuracy in function of the actual random set of chosen features. We provide our interpretation of this counterintuitive phenomenon. First we recall that both the Information Gain and the ReliefF evaluate the goodness of each feature on its own, without considering the correlation with other features. Therefore, naively selecting the top scoring features may not be a good strategy, for they are likely to be highly correlated between each other and thus redundant: hence, adding them to the training set does not improve the information available to the classifier. On the contrary, in a random selection of features it is likely to find uncorrelated features, that, though singularly carrying lower information, when combined together might well become a good discriminator, overall providing more information to the classifier.

After seeing their classification performance, let us have a closer look to the top ten features selected by both rankings, which are listed in Tab. 3.3, with a star denoting features related to the outgoing direction of traffic. Confirming our previous analysis the two rankings have no feature in common; however, they do give us some interesting insight. First of all about directionality: apparently features related to the incoming direction (i.e., traffic downloaded by the target endpoint) are more telling, as they represent the majority. Information Gain seems more interested in the temporal dimensions as in its top features we can see quite a few ones related to the “new” category, whereas ReliefF selects mostly features related to the “data” category, thus being more sensitive to the spatial dimension of traffic. Most important, overall it seems like the distribution of

Table 3.3: Top ten features according to Information Gain and ReliefF metrics.

| Information Gain | | ReliefF | |
|---|---|---|---|
| Score | Feature | Score | Feature |
| 1.13 | $\mathrm{E}[\mathbf{P}]$ | 0.26 | $B_d/P_d$ |
| 1.13 | $P/E$ | 0.26 | $B/P$ |
| 1.05 | $\mathrm{Std}[\mathbf{B}/\mathbf{P}]\star$ | 0.26 | $\mathrm{E}[\mathbf{B}_d]$ |
| 1.03 | $\mathrm{Std}[\mathbf{B}/\mathbf{P}]$ | 0.25 | $\mathrm{Std}[\mathbf{B}/\mathbf{P}]$ |
| 1.03 | $P/E\star$ | 0.25 | $P_d/P\star$ |
| 1.03 | $\mathrm{E}[\mathbf{P}]\star$ | 0.24 | $B_d/B\star$ |
| 1.01 | $P_n/P$ | 0.22 | $P_d/P$ |
| 1.01 | $\mathrm{E}[\mathbf{P}_n]$ | 0.21 | $\mathrm{Std}[\mathbf{B_d}/\mathbf{P_d}]$ |
| 0.95 | $\mathrm{E}[\mathbf{P}_n]\star$ | 0.20 | $B_d/B$ |
| 0.95 | $P_n/E_n$ | 0.19 | $\mathrm{E}[\mathbf{B}]$ |

packets and bytes across peers is a good discriminator, as both rankings includes either the ratios, the breakdowns or the statistical indexes related to such quantities.

## 3.4 Summary

In this chapter we presented two main contributions. First we introduced a coherent and comprehensive framework for the definition of behavioral features for the classification of P2P traffic, in whose definition we assumed that only flow-level measurements are available. By clearly stating the criteria at the base of feature definition, which take into account both the nature of input data (IPFIX flow counters), and of the target traffic (meshed P2P systems), we obtained an extremely general framework, which, in our opinion, could represent a valuable reference for the research community.

In the second part, we performed an analysis of the amount of information carried by the defined features, using two metrics: the Information Gain and ReliefF. We considered a large set of about 100 features, with two different timescales for statistic collections, over a large dataset of traces generated with active and passive methodologies and including 7 P2P applications. The most important takeaway of this experiment is that the features we defined provide a rather good amount of information about the application label. In fact by employing a set of ten features we are able to obtain a very good accuracy exceeding the 90%, despite the simplicity of the attributes we defined.

Moreover, our analysis actually highlights a few properties which appear to be valuable discriminators. Even though the natural next step would have been to run some more sophisticated algorithms taking into account the correlation among features (e.g. correlation based filter [97]), nevertheless in this thesis we decided to take another approach. Instead of further combining these features, in the next chapter, starting from the findings of this chapter, we develop some more sophisticated features, which try to better capture the information that seems to be most useful for P2P traffic classification.

# Chapter 4

# Abacus - Behavioral classification of P2P-TV Traffic

In this chapter, whose results have been published in [32, 173], we present our behavioral classification solution, Abacus, which bases the classification on the simple counts of packets and bytes exchanged by peers in small-time windows. The results of the previous chapter have also shown these features to be among the ones most correlated with the application label. This careful choice of features makes the Abacus classifier accurate as well as extremely lightweight. Moreover, though originally designed for P2P-TV traffic, Abacus proved itself effective with P2P traffic in general, as we will show in Chap. 6.

Let us begin by clearly stating the context and motivation of this work.

## 4.1 Introduction

The Internet has proved to have an awesome capability of adapting to new services, migrating from the initial pure datagram paradigm to a real multi-service infrastructure. One of the most recent step of this evolution is P2P-TV, i.e., large-scale real-time video-streaming services exploiting the peer-to-peer communication paradigm. There are several currently deployed P2P-TV systems [7, 11, 13, 17], which feature low-quality and low-bitrate streaming, with high-quality systems just beyond the corner. In P2P-TV systems, hosts running the application, called *peers*, form an *overlay topology* by setting up virtual links over which information is transmitted and received. A source peer injects the video stream, by chopping it into data units of a few kilobytes, called *chunks*, which are then sent to a few other peers, called *neighbors*. Each peer contributes to the video diffusion process by retransmitting chunks to its neighbors following a swarming-like behavior, somehow inspired to file sharing P2P systems like BitTorrent.

P2P-TV systems are candidates for becoming the next Internet killer application as testified by the growing success of commercial systems (such as PPLive, SopCast, TVAnts and many others) which already attract millions of users every day [86]. Also, Cisco estimates that, globally, P2P-TV traffic is now over 280 petabytes per month [169], and is projected to increase further over the next year. As a consequence, P2P-TV systems gathered the attention of the research community, interested in understanding their behavior and improve their performance, while the Internet Service Providers (ISP) community have raised some concerns about them. Indeed, P2P-TV traffic may potentially grow without control, causing a degradation of quality of service perceived by Internet users or even the network collapse [115]. In fact while the downlink rate of peers is limited by the video stream rate, the uplink rate may grow unbounded as observed in [86]. Unfortunately, most successful P2P-TV systems follow a closed and proprietary design, so the algorithms and protocols they adopt are unknown.

The vast amount of works on P2P live streaming systems is a further witness of the popularity of such applications. We mention a few references on other relevant works on P2P-TV system, limiting to works focusing on existing and popular system, which will be the target of our classification algorithm. Indeed, after the first pioneering works [185, 187] presenting this innovative way of streaming content across thousands of hosts using swarm-like unstructured system (somehow inspired by Bittorrent [58]), the main reason of the research community interest on P2P-TV is clearly the success of commercial software as [11, 13]. With this regard, measurement of P2P-TV applications are the focus of [26, 51, 54, 86, 167, 179]. Specifically, [86] focuses on PPLive, [179] on UUSee, [51] on Zattoo, while [26, 54, 167] perform a comparison of several popular applications (the first considers PPlive, SopCast and TVAnts, the second adds Joost and the latter also adds TVUplayer).

Given their diffusion, the identification of P2P-TV applications is a topic of growing interest. For instance, ISPs can be interested in blocking application $A$ and at the same time explicitly supporting application $B$, because the ISP itself provides a service relying on $B$. In a similar way, an operator could be forced to block the traffic of an application for some infringements (e.g., copyright), while still protecting the traffic of the application used by other broadcasters. More possible uses of P2P-TV application classification can be found in the field of network *monitoring* (e.g., ranking applications according to their popularity), security (blocking a given application which is exploited for DDoS attacks or worm diffusion) and charging. We think that, with the growing diffusion of P2P-TV applications, ISP will soon be asking for tools enabling this kind of activities.

However, we found that, despite the valuable effort devoted to traffic classification we reviewed in Chap. 2, the community was only partially addressing the identification of P2P-TV traffic by means of payload-based mechanism [77]. Moreover the characteristics of P2P-TV application made them a perfect target for behavioral classification [184], which, we recall, aims at identifying the traffic by the sole examination of transport-layer traffic patterns, not requiring neither packet-payload inspection nor per-packet operation. However, behavioral classification has usually achieved only coarse-grained classification of Internet applications, identifying broad application *classes* (e.g., interactive, P2P, Web, etc.) rather than discriminating different applications within the same class.

In this chapter, we tackle precisely this issue by designing a novel behavioral classification framework, tailored for P2P-TV applications, which is able to achieve fine-grained classification (i.e., distinguish among applications). Our framework uses simple application signatures gathered from the count of packets and bytes that peers exchange during small time windows. To validate the proposed classification engine, we carry out a thorough experimental campaign using both testbed traces and passive measurements collected from real networks. We consider four P2P-TV applications, chosen for their popularity among the large number of available ones. Our results show that the percentage of correctly classified traffic is above 95% of bytes. Moreover, the engine correctly labels as "unknown" the traffic generated by non P2P-TV applications, keeping the false positive rate (i.e., wrong classification of non P2P-TV traffic as such) below 0.1% in the worst case.

The rest of this chapter is organized as follows. Sec. 4.2 defines the application signatures and describes the classification framework, Sec. 4.3 thoroughly describes the workflow, methodology and datasets used to validate the classification engine. Sec. 4.5 then illustrates baseline classification results, providing an extensive study of the signature portability, to show that the proposed technique works in rather different network environments. A careful sensitivity and robustness analysis of the method to internal parameters is reported in Sec. 4.6. Afterwards, we show in Sec. 4.7 that an extended signature definition can further improve the classification accuracy. Finally Sec. 4.8 summarizes our main contributions.

## 4.2 Classification Framework

### 4.2.1 The Rationale

Our aim is to classify P2P-TV *end-points*, which can be identified by IP address and transport layer port pair $(IP, port)$. Typically, P2P-TV applications rely on UDP as the transport protocol. During installation, a single UDP port is selected at random, over which all the signaling and video traffic exchanged with other peers is multiplexed. Therefore, all the traffic going to/coming from a given *(IP,UDP-port)* end-point is actually destined to/sourced from the same P2P-TV application running on the host. This holds true for P2P-TV applications like PPLive[11], SopCast[13], TVAnts[17] and Joost[7][1], which we take as examples. Because of the continuous development of new applications, the choice of a representative set is definitely a difficult one. We decided to use the most popular applications at the time of experiments.

As mentioned before, we design a P2P-TV classification methodology that relies only on the evaluation of the *amount of information*, such as packets and bytes, exchanged by peers during small time-windows. The rationale is that a raw count of exchanged data conveys useful information concerning several aspects of P2P-TV applications.

A human analogy may help in clarifying the intuition. Suppose peers in the network are people in a party room: people generally have different behavior, e.g., they will be more or less talkative. As such, somebody may prefer lengthy talks with a few other people, whereas somebody else may prefer very brief exchanges with a lot of people. This is similar to what happens with P2P applications: some applications continuously perform peer discovery by sending few packets to a previously not-contacted peer; others tend to keep exchanging most of packets with the same peers.

Additionally, most P2P-TV applications have been designed around the concept of "chunks" of video, i.e., small units of information whose size is a typical parameter of each application.[2] Download of video content is thus performed using several chunks, and the size of flows carrying the video content is roughly a multiple of the chunk size. Moreover, P2P-TV video service has an almost constant downlink throughput, due to the nature of the video stream. By tracking the *breakdown* between the different contributors it is possible to highlight different policies that a particular application can adopt, namely, fetching chunks from many neighbors, or downloading from a restricted list of preferential peers. Yet, while any P2P-TV peer downloads an equal quantity of data, the amount of uploaded data can be significantly different from peer to peer, due to different configuration, such as upload capacity. For example, in [86], it is shown that uplink to downlink throughput ratio for PPLive varies in the $[0, 10]$ Mbps range, for the same downlink throughput of about 400 Kbps. In reason of the above observation, we assume that the classifier is located at the *edge* of the network (where all traffic exchanged by a given end-point transits), and consider only the *downlink* direction, i.e., traffic coming from the Internet and crossing the edge of the network into the end-point direction. Notice that once an endpoint has been identified by means of downlink traffic, the uplink traffic is classified as well. Also the evaluation of informa-

[1] Joost became a Web-based application in October 2008, but at the time when the experiments were performed it offered VoD and live-streaming by P2P.

[2] Note that while *frames* are the typical unit of data generated by video encoders, the segmentation in *chunks* is instead imposed by the P2P application and is typically independent from the codec.

tion content carried in the previous chapter showed that the downlink direction is the most relevant for our classification purposes.

In the following, we restrict our attention to UDP traffic, although endpoint identification can be extended to applications relying on TCP at the transport layer as well. In case TCP is used, the client TCP port is ephemeral, i.e., randomly selected by the Operating System for each TCP connection. The TCP case thus requires more complex algorithms in case of traffic *generated* from a specific peer, since ephemeral ports differ among flows generated by the same peer. However, we point out that the ephemeral port problem vanishes if we focus on the downlink direction as we do in this chapter (i.e., since we need in this case to aggregate all traffic received by a TCP server port, that remains the same for all flows of any given peer).

### 4.2.2 Behavioral P2P-TV Signatures

Let us consider the traffic received by an arbitrary end-point $\mathcal{P} = (IP, port)$ during an interval of duration $\Delta T$. We evaluate the amount of information received by $\mathcal{P}$ simply as the number of received packets. In Sec. 4.7 we extend this concept to account also for the amount of bytes, which we will show to further improve classification performance.

We partition the space $\mathbb{N}$ of the possible number of packets sent to $\mathcal{P}$ by another peer into $B_n + 1$ bins of exponential-size with base 2: $I_0 = (0, 1]$, $I_i = (2^{i-1}, 2^i]$ for $i = 1, \ldots, B_{n-1}$ and $I_{B_n} = (2^{B_n - 1}, \infty]$. For each $\Delta T$ interval, we count the number $N_i$ of peers that sent to $\mathcal{P}$ a number of packets $n \in I_i$; i.e., $N_0$ counts the number of peers that sent exactly 1 packet to $\mathcal{P}$ during $\Delta T$; $N_1$ counts the number of peers that sent 2 packets; $N_2$ the number of peers that sent 3 or 4 packets and, finally, $N_{B_n}$ is equal to the number of peers that sent at least $2^{B_n - 1} + 1$ packets to $\mathcal{P}$. Let $K$ denote the total number of peers that contacted $\mathcal{P}$ in the interval. The behavioral signature is then defined as $\underline{n} = (n_0, \ldots, n_{B_n}) \in \mathbb{R}^{B_n + 1}$, where:

$$n_i = \frac{N_i}{\sum_{j=0}^{B_n} N_j} = \frac{N_i}{K} \tag{4.1}$$

The signature $\underline{n}$ is the observed probability mass function (pmf) of the number of peers that sent a given number of packets to $\mathcal{P}$ in a time interval of duration $\Delta T$; this function is discretized according to the exponential bins described above. The choice of exponential width bins reduces the size of the signature, while keeping the most significant information that can be provided by the pmf.

In fact, since low order bins are much finer, short flows are likely to end up in different bins, even though the difference in their counts is small (e.g. flows composed by a single packet, two packets and three packets are counted respectively in the components $n_0$, $n_1$ and $n_2$). On the contrary, longer flows are coarsely grouped together in the higher bins. Intuitively, having a finer characterization of short flows can provide much information (e.g., distinguishing between single-packet probes versus short signaling exchanges spanning several packets), while there is no gain in having an extreme accuracy when considering long flows (e.g., distinguishing between 500 or 501 packet long flows). This intuition is discussed in Sec. 4.6, where we examine the impact of different binning strategies.

Since $\underline{n}$ has been derived from the pure count of exchanged packets, we name our classifier "Abacus", which is also a shorthand for "Automated Behavioral Application Classification Using Signatures". Before describing the whole classification process, let us show the expressiveness of the Abacus signatures, by presenting a few examples.

In Fig. 4.1-(a) we show an example of temporal evolution of the signature $\underline{n}$ for each of the four applications we consider in this chapter (from left to right, Joost, SopCast, TVAnts and PPLive).

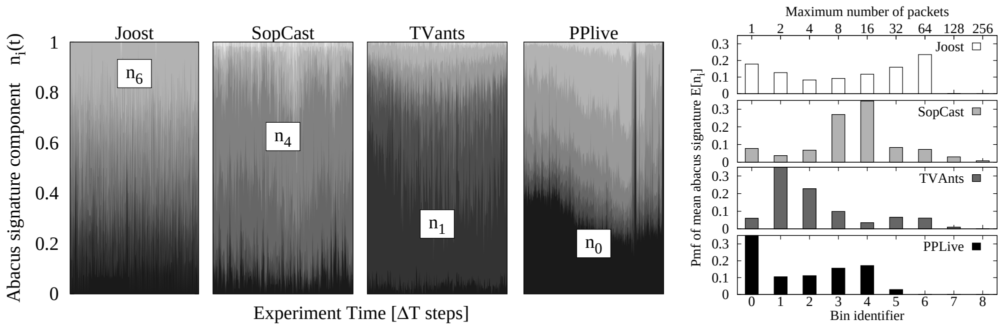


Figure 4.1: Temporal evolution of P2P-TV applications signature with a mark for the widest bin in (a), and mean value for each component of Abacus signatures in (b)

Each plot is built by running a single application on a controlled peer for an hour and capturing the received packets. Then we process this traffic and compute a signature $\underline{n}$ for each interval $\Delta T = 5\,s$. Each graph has the time on the x-axis, while on the y-axis it is reported the value of each component $n_i$. Each component is represented with a shaded area of a particular level of gray (with darker colors corresponding to low-order components, and lighter ones to high-order components). Moreover the components are staggered one above the others, so that $n_0$, the darkest component, extends from 0 to $n_0$, while $n_1$ extends from $n_0$ to $n_0 + n_1$ and so on. Clearly, as the overall signature is itself a pmf, the sum of all components is equal to 1.

Each application has its own characteristic distribution, which is extremely different from the others. The most probable bin (i.e. the one which exhibits the highest values for most of the intervals $\Delta T$) is highlighted in the figure, showing that it is different for each application. Interestingly, the most probable bin remains the same during most of the application lifetime, despite its actual width varies over time. Notice that the breakdown is not stationary over time for all applications: this is for instance the case of PPLive, as it emerges from the rightmost plot of Fig. 4.1-(a), which hints to transient or possibly "multi modal" behaviors. The dark vertical line towards the end of PPLive experiment corresponds to a sudden massive increase of $n_0$, due to a 10-seconds long blackout period (i.e. two $\Delta T$ intervals), where the end-point under observation was essentially receiving single-packet probes, and likely no video chunks.

To better highlight how Abacus signatures capture the differences between applications, we have computed the *average* of each single signature component $n_i$ over all the intervals represented in Fig. 4.1-(a), and reported it in the histograms of Fig. 4.1-(b). Bin identifier $i$ is reported on the x-axis, with top x-axis showing the maximum number of packets within the bin.

Interesting behaviors stand out from the picture. For instance, Joost peers preferentially receive either a single or several $(32, 64]$ packets from any given peer. SopCast instead prefers middle-sized burst of $(5, 16]$ packets, while TVAnts prefers lower order bins $(2, 7]$ packets. Finally, PPLive highly prefers single packet exchanges. This confirms that different P2P-TV applications have remarkably different behaviors, just like humans at a party: in the next sections, we exploit this evidence for classification purpose. Obviously if two applications employ the same signaling and diffusion algorithms, they are characterized by a similar behavior and, thus, hardly recognizable. Under these assumptions, a fine-grained classification is no longer possible. Notice that this is a common problem with all classification methodologies: as far as the features are the same, the classifier is confused. For example, traditional Deep-Packet-Inspection (DPI) classifiers cannot distinguish two VoIP applications relying on the same protocol at the session level, e.g., RTP. Similarly, behavioral classifier that use packet size and inter-packet-gap as features cannot distinguish VoIP applications that use the same Codec.

## 4.3 Methodology

Classification of P2P-TV traffic can be performed by exploiting the Abacus signatures so far described through any supervised learning machine. We resort to Support Vector Machines (SVM) (see also Sec. 2.2.1), well known for their discriminative power [62]. In this section, we describe the workflow we follow in the evaluation, as well as the dataset used for our experimental campaign.

### 4.3.1 Workflow overview

Employing a supervised machine learning algorithm, we follow the same steps described in Sec. 2.2, i.e., training, classification and validation. However, it is worth recalling the classical workflow here because, first, we modified it it to suit our goal of classifying P2P-TV application, and, second, it gives us the opportunity of discussing the applicability of such a methodology to different scenarios.

The first step consist in generating the labeled signatures of known P2P-TV applications, using the testbed traces described later in this section: specifically, for each trace, we build an Abacus signature $\underline{n}$ every interval of $\Delta T$ seconds. SVM, like all supervised algorithms, needs to be trained as illustrated in the top part of Fig. 4.2 with some sample Abacus signatures, with the associated labels that specify the generating P2P-TV applications. Therefore, each signature in our dataset is possibly chosen, at random, to be included in the training set: impact of training set size and selection policy on the classification performance is discussed in details in Sec. 4.6.

The output of this phase is a *trained model*, which is basically a careful selection of samples from the original training set, called Support Vectors. Such points define a partition of a vectorial space obtained by applying a transformation (based on a Gaussian kernel unless otherwise stated) to the original space defined by Abacus signatures. We examine the impact of other kernels later on in Sec. 4.6.

During the classification phase, in the bottom part of Fig. 4.2, SVM, based on the trained model, associates a label to a previously unseen signature, by identifying in which portion of space it falls into. Normally, the process ends here, but we needed an additional phase to handle unknown traffic: the classification is accepted provided it passes a *rejection criterion*, to correctly discard non P2P-TV traffic. In fact, as SVM partitions the space into regions, it will *always* label any new sample as belonging to one region specified during the training phase. We thus devise a rejection criterion, which is extensively explained in Sec. 4.3.2, based on a measure of distance between signatures in the probability distribution space. We defer all details to the next section: now, we just mention that the this threshold-based criterion is able to reject SVM classifications and label sample non-P2P traffic as “Unknown”.

Finally, it is worth remarking that, apart from the training phase, the overall framework is well suited for live classification, yielding a classification result every $\Delta T$ s, (which is the parameter that defines the *reactivity* of our classifier). Notice as well that all operations performed on observed traffic are extremely lightweight so that our classifier can cope also with high rates of traffic (a brief analysis of the scalability of our Python implementation is reported at the end of Sec. 4.6 and a more formal analysis is also performed in the next chapter, which compares Abacus with a payload based classifier). As a side note, we have also released an open-source demo of our classifier, available online [174] and briefly described in Sec. 5.4, that allows the user to run Abacus in real-time, on live traffic captured on a real network interface. At the same time, we

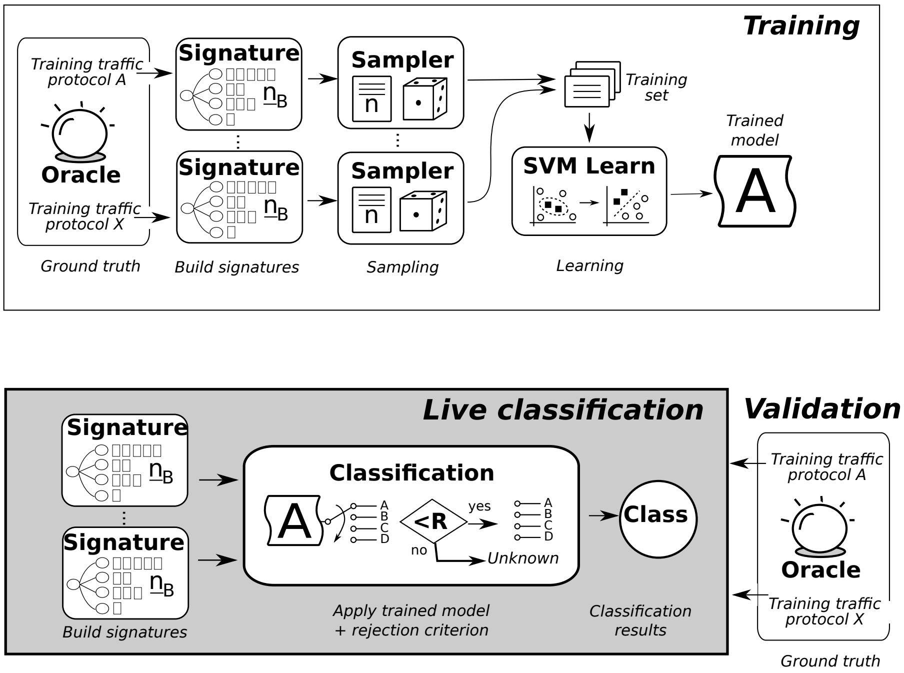


Figure 4.2: Classification framework: Model training (top) and validation (bottom). Live classification is performed as in validation, except that no ground truth is available as in the validation case.

point out that in all our experiments we used Abacus as an *offline* classifier on pre-recorded traces with associated ground-truth, in order to perform multiple experiments on the same dataset, which is needed to have a trustworthy validation of the classification performance. However, from the above discussion it follows that results are representative of live classification as well.

The only offline operation is constituted by the training phase. Concerning this point, we anticipate that the results of our portability analysis show that a well chosen training set can be used to successfully classify traffic in a variety of network conditions, so that offline retraining is rarely required. Moreover, the collection of the training set is a very simple process, which can be easily automated. This is not the case for many other classifiers, usually presenting a much more complicated training phase, which can even involve manual inspection of the packet traces (e.g. DPI requires analysis of the traffic to extract keywords or regular expressions that identify the target protocol).

### 4.3.2 Rejection Criterion

SVM is a powerful classification algorithm, but for our purpose of network traffic classification it presents one simple shortcoming. Recall that a SVM trained model is composed of two parts: first a mapping from the original features space to a multidimensional space; second a set of hyperplanes individuated by Support Vectors, which defines a *partition* of the target space into regions, each corresponding to a possible classification outcome. The problem is that, in this partitioned space, a new point is always deemed to fall into a region, hence it will always be associated to one of the label represented in the training set. Unfortunately in traffic classification, we also need to deal with “other” traffic, generated by different applications. To overcome this issue, we define a rejection criterion, whose aim is basically recognize to traffic belonging to *none* of the target training classes.

Given that Abacus signatures are probability mass functions, we use a measurement index suitable to quantify distribution similarity. Given two pmfs, there exist several indexes to evaluate their degree of similarity. The Bhattacharyya distance (BD) [36] is a measure of *divergence* of two probability density (or mass) functions. Given two pmfs $p$ and $q$ over $n$ discrete values, the Bhattacharyya distance $BD(p,q)$ is defined by:

$$BD(p,q) = \sqrt{1-B} \quad \text{where} \quad B = \sum_{i=1}^{n} \sqrt{p_i q_i} \tag{4.2}$$

Bhattacharyya distance, which is a particular case of the Chernoff distance, has several properties. First, it verifies the triangular inequality. Values of $BD$ close to zero indicates strong similarity (if $p_i = q_i \ \forall i$, $B = 1$ and $BD = 0$) whereas values close to one indicates weak similarity. The Bhattacharyya coefficient $B \in [0,1]$ is the scalar product between the two vectors $p' = (\sqrt{p_1}, \ldots, \sqrt{p_n})$ and $q' = (\sqrt{q_1}, \ldots, \sqrt{q_n})$, which leads to a geometric interpretation of the coefficient B. In fact it can be seen as the cosine of the angle between $p'$ and $q'$. The Bhattacharyya distance has been successfully applied in different contexts such as signal selection [99], or classification [123].

In our context, we use $BD$ to measure the *separability* of two traffic classes. In particular, we *reject* the SVM label $C$ of a signature $\underline{n}$ whenever the distance $BD(\underline{n}, E[\underline{n}(C)])$ exceeds a given threshold $R$, i.e., the sample will be labeled as *unknown*. $E[\underline{n}(C)]$ is the average signature computed on the training samples of application $C$. Notice that the average signature $E[\underline{n}(C)]$ identifies the center of the cluster formed by all training set signatures of application $C$. In other words, we accept SVM decision conditionally to the fact that the observed traffic signature $\underline{n}$ lies within a radius $R$ from the center of the SVM training set for that class. The selection of the threshold value is simple but delicate, as it heavily influences the performance of the classification in terms of both True Positive Rate and False Positive Rate, hence we will carefully provide a sensitivity analysis on this value.

However, there exist some cases where no false alarm can be raised (i.e., non P2P-TV traffic will be always classify as unknown), which makes Abacus robust by design. Let us consider the case when traffic is received from one peer only. Then, the Abacus signature $\underline{n}$ is a vector containing a single 1 at the bin $i^*$. In this case, the distance from the center $E[\underline{n}(C)]$ ($C$ for short) of the cluster of an arbitrary application will be $BD(\underline{n}, C) = \sqrt{1 - \sqrt{C_{i^*}}}$. Suppose we choose a threshold $R$, then we reject the classification if:

$$BD(\underline{n}, C) = \sqrt{1 - \sqrt{C_{i^*}}} < R$$

from which we can derive an acceptance condition on the value of $C_{i^*}$:

$$C_{i^*} < (1 - R^2)^2$$

In case we set $R = 0.5$, as we will do in Sec. 4.5, we have that a signature is rejected whenever its most likely bin $C_{i^*}$ exceeds $(1-0.5^2)^2 = 0.5625$: notice from Fig. 4.1 this is never the case for any P2P-TV applications, whose most likely bins remain below $0.3$. In other words, the criterion is robust with respect to P2P-TV applications (whose signatures are not rejected) and with client-server applications as well (since any signature containing a single bin has forcibly $C_{i^*} = 1$ and is thus rejected). Similarly, consider the case when traffic is received by only two peers: any such signature is a linear combination of two unit vectors and it can simply be proved that it will be rejected too. Therefore, also in this case, the signature is always rejected (classified as “unknown”)

Table 4.1: Details about the Testbed Traces

| Application | Hr | UDP Signatures | UDP Packets | UDP Bytes | UDP% |
|---|---|---|---|---|---|
| SopCast | 36 | 26k | 17.2M | 7.5G | 92.5 |
| TVAnts | 36 | 26k | 14.2M | 7.1G | 33.0 |
| PPLive | 26 | 19k | 11.7M | 5.1G | 70.7 |
| Joost | 30 | 22k | 6.1M | 6.4G | 99.5 |
| Total | 128 | 93k | 48.2M | 26.1G | 73.7 |

and therefore no false alarm is raised. To summarize, the criterion rejects any peer contacting two or fewer other peers during a given time interval $\Delta T$ – which basically means that client-server traffic will never raise any false alarm, but will rather be correctly classified by the engine as “Unknown”.

## 4.4 Dataset

Assessing classification performance represents a well-known problem, as we already mentioned in Chap. 2, because of the difficulty of finding traces with reliable ground-truth.

To gather robust results we adopted a mixed approach: we use both (i) traces actively gathered in a large scale testbed, and (ii) passive traces collected from different real operational networks. Testbed traces contain P2P-TV traffic only and allow us to evaluate the engine capability to correctly discriminate P2P-TV applications and correctly label all P2P-TV traffic. Conversely, real network traces do not contain any P2P-TV traffic and allow us to verify that the engine correctly handles unknown applications as well (i.e., does not label other traffic as P2P-TV).

### 4.4.1 Testbed Traces

We extensively use the dataset captured during the testbed experiments in context of the NAPA-WINE project, which was describe in Sec. 2.4. Overall, the testbed is representative of about 130 hours worth of video streaming, $93$k signatures samples, $48$M packets and $26\,$GBytes of data. Since our classifier operates only on downlink traffic, that is the traffic directed to the target endpoint, we first extracted the relevant UDP packets from the traces. The overall duration, number of signatures (i.e. snapshot of $\Delta T = 5\,s$) as well as number of UDP packet and UDP bytes finally included in our dataset per application are reported in Tab. 4.1. Notice that the table also reports the percentage of UDP traffic relatively to the overall amount of IP traffic in the captured testbed traces. With the exception of TVAnts, we gather confirmation that UDP traffic is prevalent, accounting to about 3/4 of the total traffic volume, and more than 90% in case of SopCast and Joost. Notice that this changes with respect to previous work [167] in which TCP was found to be responsible for the bulk of the exchanges. As the version of the software that we used in our experiments is more recent than that used in [167], the data confirms that P2P-TV sofware is continuously evolving[90], and that in this evolution UDP is preferred over TCP. In the TVAnts case, instead, the software version did not evolve from 2006, in which case UDP/TCP ratio is in agreement with [167].

### 4.4.2 Real Traces

Real traffic traces are collected from two different networks in Italy: (i) CAMPUS (C) is the trace from our university and (ii) ISP (I) is the trace of 2006 from a large Italian ISP. For all the details

Table 4.2: Details about Real Traces. To perform a worst-case analysis, only end-points that can lead to false positive classification are considered (28% of the CAMPUS and 15% of the ISP overall traffic volume).

| Network | Traffic | Signatures | Packets | Bytes |
|---|---|---|---|---|
| CAMPUS | UDP | 1.9M | 73.6M | 10.6G |
| | Skype | 0.5M | 11.9M | 2.2G |
| | DNS | 0.2M | 5.0M | 0.7G |
| ISP | UDP | 0.7M | 28.5M | 24.9G |
| | eDonkey | 0.3M | 9.8M | 1.4G |
| | DNS | 24.4k | 0.6M | 37.8M |

about these trace please refer to Sec. 2.4.

In both cases, traces were collected during May 2006, when P2P-TV applications were not popular in such networks. As such, they are instrumental to assess the amount of false positive classification (i.e., non-P2P-TV traffic classified as P2P-TV).

As detailed in Sec. 4.3.2, the rejection criterion is very effective in avoiding false alarms, since all signatures related to hosts contacting two or less peers (i.e., not running any P2P application) is rejected by design. For this traces, the percentage of signature that cannot be classified amounts to 62% and 82% in CAMPUS and ISP respectively, which corresponds to 72% and 85% of the ISP traffic volume. However, our aim is to gather conservative performance bounds: therefore, we build a *worst-case scenario* for the comparison, so that the actual performance in operational networks can be expected to be much more robust. To devise the worst-case scenario, we do not consider unknown traffic that Abacus would reject by design (e.g., such as single-flow client-server traffic) , but instead take into account only the subset of traffic that Abacus could actually misclassify (i.e., accepting unknown traffic as P2P-TV and generating thus a false alarm), and that constitutes merely the remaining 28% of the CAMPUS and 15% of the ISP traffic traces.

As reported in Tab. 4.2, we consider both the aggregated UDP traffic volume (that Abacus can misclassify) produced by all applications in the CAMPUS and ISP traces, as well as relevant UDP traffic subsets, representative of both P2P and client-server applications. The rationale of this choice is that we want to test whether false-positive classification is more likely to arise when considering P2P applications or traditional client-server services. Specifically, we consider Skype and eDonkey traffic as examples of voice and file-sharing P2P applications, and DNS as an example of traditional client-server service. To reliably identify eDonkey, we develop and implement a DPI classifier, based on [6, 108], while we classify Skype with out previous work [39], and rely on Tshark DPI protocol inspection capabilities to isolate DNS traffic.

## 4.5 Experimental Results

This section reports the results of our experimental campaign.

We start by considering signatures that are defined on the number of packets exchanged, providing first some baseline results in a general enough scenario. Then we investigate the *signature portability*, across different space, time and network conditions: this is done to assess if a classifier trained with signatures gathered under a given set of conditions, is able to correctly identify traffic generated in completely different settings (e.g., different ISPs, access technologies, networks conditions, different TV channels, different times, etc.).

Table 4.3: P2P-TV Classification Performance: Confusion Matrix of Testbed and Real Traces (Signatures)

| | PPLive | TVAnts | SopCast | Joost | Unk |
|---|---|---|---|---|---|
| PPLive | **81.66** | 0.58 | 9.55 | 2.32 | 5.90 |
| TVAnts | 0.49 | **98.51** | 0.18 | 0.77 | 0.04 |
| SopCast | 3.76 | 0.11 | **89.62** | 0.32 | 6.19 |
| Joost | 2.84 | 0.55 | 0.28 | **89.47** | 6.86 |

| | PPLive | TVAnts | SopCast | Joost | TNR |
|---|---|---|---|---|---|
| CAMPUS | 2.42 | 2.23 | 0.01 | 0.02 | **95.3** |
| ISP | 0.66 | 0.13 | 0.43 | 0.10 | **98.7** |

### 4.5.1 Baseline results

In this first set of experiments, we report results considering the following parameters: for each application, the training set includes samples extracted considering 2 peers at random from each group of $N = 7$ networks taking part to the experiment. From all signatures they generate, 4000 signatures are randomly extracted to define the training set, which corresponds to about 17% of all signatures. Experiments are then repeated 10 times, randomly changing the training set and so the validation set at each run. Finally, average classification results are computed. We consider signatures generated using $\Delta T = 5$s intervals. Classification is performed using SVM with a Gaussian kernel and exponential bins $B_n = 8$, with a rejection threshold $R = 0.5$. Parameters sensitivity and optimization is later discussed in the remaining part of this section.

The top part of Tab. 4.3 reports the classification performance of on the testbed traces using the traditional “confusion matrix”, already explained in Sec. 2.3. Performance are expressed for the time being in terms of signatures (i.e., groups of packets received during a $\Delta T$ interval) but we will also consider classification performance in terms of packets, bytes and peers later on. It can be seen that, in the worst case, about 81% of individual signatures are correctly classified. The most difficult application to identify appears to be PPLive, which is confused with SopCast (9.55%) or Joost (2.32%). Other applications show higher TPR, with TVAnts showing almost perfect match. On average, about 4.5% of P2P-TV signatures are rejected, therefore being labeled as Unknown.

Bottom part of Tab. 4.3 reports results considering the real traces dataset. Since no P2P-TV traffic is present in this dataset, True Negative Ratio (TNR) is the main index to be considered (boldface, rightmost table column). Results show that the rejection criterion adopted is very robust, so that less than 5% (see Sec. 4.3.2 for further details) so that the overall TNR is actually much higher: namely, considering all the UDP traffic traces of the real networks, more than 99% of the signatures do not raise any false alarm. Left part of the Table details the breakdown of False Positives: PPLive and TVAnts are the cause of most misclassification, while Joost practically causes no False Positives.

### 4.5.2 Signatures Portability

We now evaluate *network portability* of Abacus signatures. The objective is to answer the question: how generic is a training performed considering traces collected in a network? Our testbed dataset is different enough to see what happens when, for example, the classifier is trained considering a trace collected in a University Campus network, and then used in a totally different network, like a ADSL scenario. Moreover, both the access type and the channel popularity could impact the accuracy of the training set, which we deal with in the following. Besides, we are interested in testing how often the signatures have to be redefined, considering P2P-TV traces gathered in different years. Finally, we also test how robust the classifier is in presence of high packet loss or limited bandwidth.

For the sake of simplicity, we consider only packet-wise Abacus signatures and testbed traces, and no longer apply the rejection criterion. Results are summarized in Tab. 4.4: the first column reports the experiment label, the second column (Train) states which training set was used, while the third column (Test) reports the dataset using for the classification process. TPR for each application are reported in the subsequent columns. To ease the comparison, the first row (labeled *Ref*) reports the baseline results: notice that TPR is slightly higher with respect to Tab. 4.3, since we do not apply the rejection criterion.

#### 4.5.2.1 Portability across Network Sites (NS)

In the first scenario, we consider traffic captured from PCs running at different institutions, i.e., in different Countries, networks, etc. (see Tab. 2.4). We start by considering peers that are all placed in corporate or campus networks, with high-bandwidth connections to the Internet. There are 7 of such sites. For each application, we select 4 sites, and used traffic collected there for the training. Then, traces collected in the remaining 3 networks are classified to evaluate TPR. To gather robust results, we consider every possible combination $\binom{7}{4} = 35$ of training and validation subsets. For each combination, 3 tests are performed with different random training samples.

Results are reported in the raw labeled as *NS* in Tab. 4.4, which shows that signatures are network-portable under homogeneous settings: indeed, the largest performance drop is 4%, which corresponds to the PPLive case.

#### 4.5.2.2 Portability across Access Technologies (AT)

We now test to what extent signatures are portable across different access technologies, e.g., ADSL versus High Bandwidth (HB) access. As noted in [86], nodes with high-bandwidth access can act as “amplifiers”, providing content to possibly several peers; conversely, ADSL peers may only act as “forwarder” due to the limited uplink capacity. Despite we consider only the downlink traffic, such different behaviors can impact the Abacus signatures, e.g., due to a different fraction of signaling packets a peer receives. For example, an amplifier peer can receive many small sized acknowledgments, while a low-capacity peer mainly receives large packets containing video data. We therefore split the testbed dataset into two parts: the first contains traces collected from all High Bandwidth PCs, while the second contains ADSL PCs. Three tests are performed: (i) classifying ADSL traces using ADSL training set, (ii) classifying ADSL traces using the HB training set and (iii) classifying HB traces using ADSL training set. Each test has been repeated 10 times, and average results are reported.

Results are reported in rows labeled *AT* in Tab. 4.4. Overall, Abacus signatures confirm their portability even across different access networks: for TVAnts, SopCast and Joost, results are modestly impacted by train/test combination (being 8% of reduced TPR the worst case). In case of PPLive, the TPR drops to 58% when HB training is used to classify ADSL traffic. This is likely due to the fact that PPLive is very aggressive in exploiting HB peers upload capacity, so that the number of peers sending acknowledgments shifts the signature toward low bins, i.e., few acknowledgment packets are received from a given peer. ADSL peers, on the contrary, contribute with little upload bandwidth, so that the incoming traffic is mainly due to video chunks received as trains of packets, i.e., groups of large data packets that are received from contributing peers.

Table 4.4: Signature portability: TPR evaluation

| | Train | Test | PP | TV | SO | JO |
|---|---|---|---|---|---|---|
| *Ref* | ALL | ALL | **84.84** | **98.51** | **92.63** | **91.50** |
| *NS* | 4/7 | 3/7 | 78.90 | 97.61 | 90.30 | 88.61 |
| *AT* | ADSL | ADSL | 83.48 | 97.86 | 95.61 | 91.36 |
| | ADSL | HB | 79.63 | 93.73 | 87.30 | 90.61 |
| | HB | ADSL | 58.28 | 98.15 | 93.70 | 81.55 |
| *CP* | POP | POP | 95.88 | - | - | - |
| | POP | UNP | 48.59 | - | - | - |
| | UNP | POP | 94.79 | - | - | - |
| *TI* | 2008 | 2006 | 18.81 | 98.44 | 51.06 | - |
| *EI* | HB | Bw | 91.14 | 76.80 | 75.76 | - |
| | HB | Delay | 88.19 | 84.62 | 77.80 | - |
| | HB | Loss | 75.22 | 91.77 | 84.31 | - |

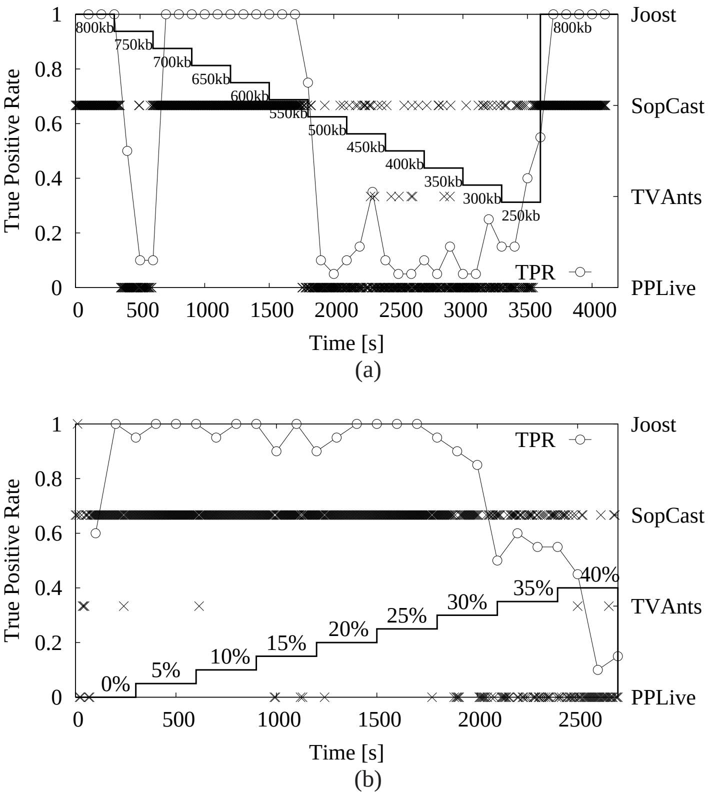
#### 4.5.2.3 Portability across Channel Popularity (CP)

We now consider what is the impact of channels with different popularity. Channel popularity indeed may significantly influence the P2P-TV application behavior: for example, considering popular channels, a large number of peers are available, while for unpopular channel few peers can be used to exchange the video content. We performed a second experiment considering a very popular (POP) channel using PPLive. We selected PPLive since it is the P2P-TV application for which Abacus showed the worst performance so far. The total number of peers observed during this experiment was larger than 200000, while in the previous dataset less than 56000 peers were observed. We refer to this dataset as a unpopular channel (UNP). As before, we evaluate the portability over all combination of train/test sets, repeating the experiments 10 times.

Results are reported in the rows labeled *CP* in Tab. 4.4. Few considerations hold: first, PPLive classification performance improves when it comes to the classification of popular channels (i.e., TPR in POP/POP and UNP/POP cases is about 95% versus the about 85% of the UNP/UNP case used as reference). Nonetheless, we observe that the classification of UNP dataset when training has been done considering the POP dataset leads to poor performance (TPR drops to less than 50%). This partly limits the portability across channels. A simple solution consists in building a training set containing a mixture of signatures from both traces, which raises the TPR again to about 85%. This result suggests that channel popularity should be explicitly taken into account when building the training set, by including samples that are representative of different channel popularity.

#### 4.5.2.4 Portability over Time (TI)

We now focus on the signatures portability over different periods of time. From a practical point of view, this allows to know how often classifiers should be retrained. We resort to the traffic traces used in [167], that authors kindly made available to the scientific community. Traces of [167] were collected in July 2006 during the Fifa World Cup: the study focused on the same applications we examined in this chapter, with the exception of Joost which was not available at that time. Overall, the time-portability measurements account for 14 hours of video, $14M$ packets and 2.3 Gbytes of data. We classify this old dataset using the Abacus classifier trained with the dataset collected in 2008. (same training set of Sec. 4.5.1). Notice that the network environment was also different, so that we are *jointly* evaluating time and network portability. Results are reported



(a)

(b)

Figure 4.3: Portability over Emulated Impairment: example of temporal evolution of SopCast classification for decreasing bottleneck bandwidth (a) and increasing packet loss rate (b)

in the row labeled *TI* of Tab. 4.4, which shows that TVAnts is correctly classified, SopCast has a TPR of 51%, while PPLive is almost completely misclassified. This suggests that some applications changed drastically their behavior from July 2006 to March 2008. Notice that TVAnts was at version 1.0.58 in [167] and it is now at 1.0.59, which suggests that little changes have been implemented. On the other hand, SopCast moved from version as explained in [89, 90]. Thus, in case applications do not change their internal algorithms, the Abacus signatures are extremely portable across time –even across years– as we see in the case of TVAnts. On the other hand, if an application implements new algorithms which result in new behavior, then Abacus should undergo a new training phase. However, similar considerations are valid for any kind of classifier, from port-based ones, to DPI or behavioral classifiers. In fact, whenever the features change, all classifiers must be trained again (e.g. by changing the port number, updating the DPI signature or re-training the behavioral/statistical features).

#### 4.5.2.5 Portability over Emulated Impairments (EI)

As a final case, we consider whether Abacus signatures are portable across different network conditions. We consider the traces gathered in an active testbed [26], where changing network conditions were artificially enforced. In particular, in these experiments, a Linux router was used to emulate some network conditions: (i) bandwidth, (ii) delay and (iii) packet losses were imposed on the downlink path to the PC running the P2P-TV application. To avoid going out of topic, we refer the reader to [26] for a complete description of the testbed: we only point out that impairments range from mild to very tough conditions (e.g., 200 Kbps of available downlink bandwidth, delay up to 2 s and packet losses up to 40%). Traces gathered in this testbed are classified considering the HB training set, and results are reported in the last lines of Tab. 4.4 labeled *EI*. Even in these extreme conditions, Abacus still exhibits very high TPR, which can still exceed 90% for some applications, with a worst case of about 75%. Reported results are averaged over all the time varying conditions, including very distorted scenarios. Classification results are differently impaired by different network conditions. For example, PPLive is mostly affected by loss increase, while TVAnts classification results are more sensitive to bandwidth change. SopCast results are mostly affected by bandwidth and delay changes.

Interestingly, results ameliorate considering PPLive classification in the case of bandwidth limitations. While this seems counter intuitive, it can be explained considering that most False Negatives obtained from other applications are actually misclassified as PPLive. This suggests that PPLive signatures are more variable and spread out, avoiding FN classification for PPLive but possibly causing more FP classification for other applications.

As an example, Fig. 4.3 reports the time evolution of two different experiments of SopCast classification, considering a scenario in which the available bandwidth is decreasing (top plot), or the packet loss rate is increasing (bottom plot). Every 5 minutes network conditions are artificially worsened by either reducing the available bandwidth by 50 Kbps, or by increasing the packet loss rate by 5%. The resulting impairment profile is reported in the picture.

Fig. 4.3 plots individual classification decisions, taken each $\Delta T = 5$s: these are represented with crosses, referring to the right y-axis, and allow to see when and how the application has been eventually misclassified. The picture also reports the True Positive Rate, evaluated over 20 consecutive signatures (i.e., 100 seconds), represented as a continuous dotted line referring to the left y-axis. Considering the top plot, which refers to bandwidth limited scenario, it can be seen that as soon as the bottleneck bandwidth kicks in, SopCast is misclassified as PPLive during a brief period, possibly hinting to a sudden reaction of the application to the anomalous conditions. Then, SopCast is correctly classified until the available bandwidth drops too low: afterward, SopCast TPR drops quickly, being most of the time misclassified as PPLive and seldom with TVAnts. At the end of the experiment, when the bottleneck bandwidth is removed, SopCast is again correctly classified. Similar considerations hold for the loss scenario depicted in the bottom plot of Fig. 4.3, in which samples are misclassified only when loss rate exceeds 30%.

## 4.6 Sensitivity Analysis

After evaluating the effect of external conditions on the classifier performance, in this section we rather focus on its internals. In fact we present the results of the experiment carried on to investigate the *sensitivity* of the classification to parameter changes, so as to select the settings which guarantee the best performance.

### 4.6.1 Impact of the Rejection Threshold $R$

Irrespectively of the precise distance metric used in the rejection criterion, the selection of the rejection threshold $R$ is guided by the following trade-off: $R$ should be large to maximize the TPR (i.e., avoid classifying P2P-TV as Unknown), while $R$ should be small to minimize the FPR (i.e., avoid classifying unknown traffic as P2P-TV).

We evaluate the TPR and FPR as a function of $R$ in Fig. 4.4, where a solid vertical line at $R = 0.5$ represents the threshold used so far. It can be seen that TPR of P2P-TV applications quickly saturates to an asymptotic value for $R \geq 0.5$. Conversely, the FPR of non-P2P-TV traffic

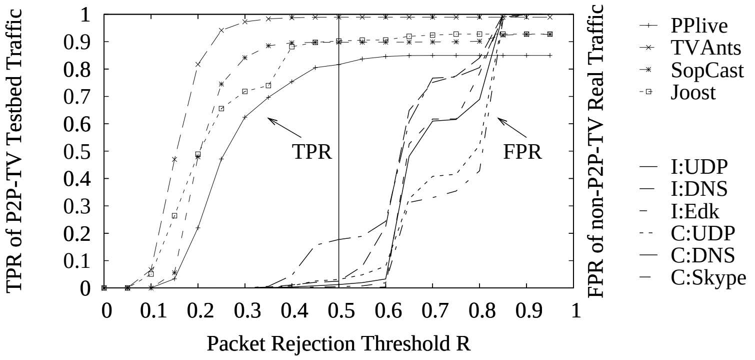


Figure 4.4: TPR and FPR as a function of the rejection threshold $R$ evaluated on packet feature.

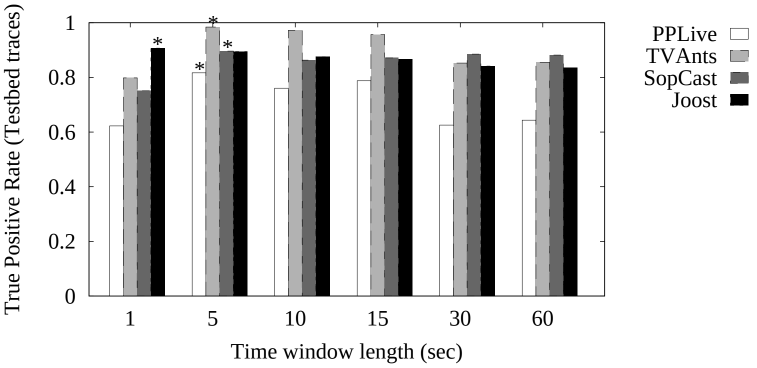


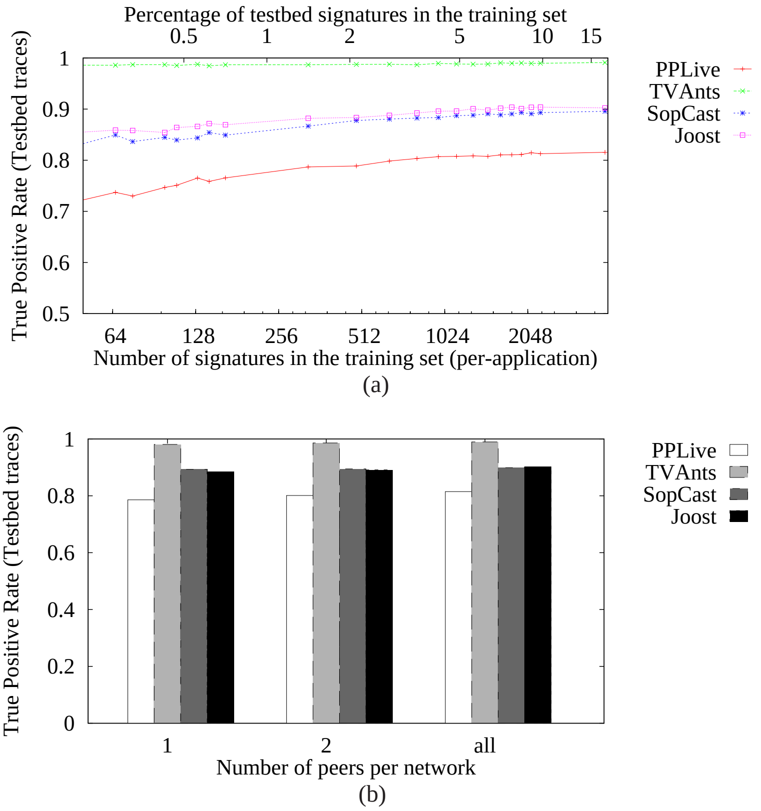
Figure 4.5: P2P-TV TPR for different values of the time interval $\Delta T$. Best-case for each application is labeled with a star $*$ sign.

increases only for large values of $R$, and for low values of $R \leq 0.5$ almost no false alarm is raised. Among the various traffic, only DNS traffic is sometimes misclassified as P2P-TV traffic. This confirms $R = 0.5$ to be a good choice of the threshold.

### 4.6.2 Impact of Time Interval $\Delta T$

The choice of the value of the $\Delta T$ parameter is driven by the following trade off. On the one hand, timely detection of P2P-TV traffic needs $\Delta T$ to be small. On the other hand, sufficiently large time intervals must be considered to estimate the signature. Moreover, to limit computational complexity and the generated amount of information, network monitoring entities (such as Netflow [56] probes) typically operate on larger timescales.

Results are reported in Fig. 4.5, where the best results for each application is labeled with a star. As expected, medium-duration window (e.g., $\Delta T = 5\,$s) yields higher TPR for most applications, while providing a more timely classification. Smaller values of $\Delta T$ limit the estimation of the bin distribution, impairing classification accuracy. Interestingly, for large windows (e.g., $\Delta T = 60\,$s) the discriminative power of the Abacus signatures only mildly degrades for three out of four applications. Only for PPLive we observe a decrease of 20% for the TPR, which is due manly to the rejection criterion being too aggressive and discarding correct classifications, suggesting that for longer $\Delta T$ the rejection criterion should be more carefully tuned.



(a)

(b)

Figure 4.6: Impact of size (a) and diversity (b) of the training set

### 4.6.3 Impact of Training Set Size

We now assess the classification sensitivity to variations on the training set *size*, i.e., the impact of the number of samples that form the training set. Indeed, the training set should be large enough to be representative of the application behavior under a large range of conditions. On the other hand, the SVM training and classification computational costs benefit of a smaller set. Moreover, too large a training set could result in the well-known phenomenon of *over-fitting*, resulting in poor classification performance. Fig. 4.6-(a) reports the TPR for each application, as a function of the number of signatures used in the training phase per each application. For each value of the training set size, we run 10 independent experiments over which results are averaged. The bottom x-axis reports the number of signatures used for training on a logarithmic scale, while the upper x-axis reports the percentage of training samples versus the total testbed dataset. Training set size extends up to 4000 signatures per application, which corresponds to the 17% early used early used in Sec. 4.5.

Results show that no over-fitting phenomenon is experienced, since the TPR increases with the increase of the training set size. Best results are obtained considering 4000 signatures per application, which validates the choice made in previous section. Notice that even by drastically decreasing the training set size to about 300 signatures per application, the corresponding decrease in TPR is only modest, e.g., 8% in the worst case of PPLive, while TVAnts shows excellent results even with an extremely reduced training set. This interesting performance is the result of both the discriminative power of SVM, and the descriptive expressiveness of Abacus signatures. Clearly, a better characterization of each application behavior is achieved including more signatures, as reflected by the improved performance.

### 4.6.4 Impact of Training Set Diversity

We now fix the training set size and focus on the training set *diversity*, i.e., the number of different peers from which signatures are selected. Our aim is to roughly assess whether it is sufficient to observe a single peer in a given network to gather an adequate description of the application behavior in that network, or whether the observation of several peers is necessary. To answer this question, we fix the overall training set size to 4000 signatures per application and vary the number of peers selected as reference in each network (see Tab. 2.4 for details on the number of peers). Each experiment is repeated 10 times to collect average results. Fig. 4.6-(b) shows the TPR obtained considering a reference set of *one*, *two* or *all* peers for each network in the testbed. Results show that the increase of the number of peers only provides a very limited gain on the classification performance. From a practical perspective, this is a very desirable property: even a single trace is sufficient to build expressive signatures.

### 4.6.5 Impact of SVM Kernel and Binning Strategy

Since the core of our classification framework exploits SVM, all parameters that are susceptible of affecting its performance need to be investigated as well. Therefore, we focus on two main choices concerning SVM: (i) the *kernel* function and (ii) *binning* strategy.

The kernel function is used to map the training points to an hyper-space where they can be separated by hyper-planes. The SVM literature is very rich of kernel functions, which are more or less indicated for different kinds of data. In our study we evaluate three well-known kernels: the general-purpose *gaussian* kernel ($K_G$), the *linear* kernel ($K_L$) and the Bhattacharyya kernel ($K_B$). The linear kernel (4.4) is simply the dot product of two feature vectors, while the Bhattacharyya kernel (4.5) can be obtained by substituting each features with its square root [96]. As we used the Bhattacharyya distance [36] between probability mass functions as a core tool for the rejection criterion to quantify the separability of two classes, a natural question is whether the kernel function (4.5) can be helpful to better separate the different applications also from the SVM standpoint.

$$K_G(\underline{x}_i, \underline{x}_j) = e^{-\gamma||\underline{x}_i - \underline{x}_y||^2} \tag{4.3}$$

$$K_L(\underline{x}_i, \underline{x}_j) = \underline{x}_i \cdot \underline{x}_j \tag{4.4}$$

$$K_B(\underline{x}_i, \underline{x}_j) = \sqrt{\underline{x}_i \cdot \underline{x}_j} \tag{4.5}$$

As far as *bin distribution* is concerned, we use either $B_{exp} = 9$ exponential-width bins (base 2), or $B_{fix} = 255$ constant-width bins (1-packet steps), both spanning over the $[0, 255]$ packets range. Recall that the number of bins impacts both memory requirement and computational complexity, so that exponential binning should be preferred in case of comparable classification performance.

Results are shown in Tab. 4.5, which reports classification results in terms of the TPR of P2P-TV applications and the number of Support Vectors (SV) of the trained model. The latter is a measure of the classification computational cost, since the number of operations that has to be performed to classify each signature grows linearly with the number of SVs. The cost of the training phase is not considered, since it is an offline operation rarely performed. Notice also that, due to the type of operations in (4.3), (4.4) and (4.5), the selected kernel has impact on the computational cost – with the Linear kernel being light-weighted, the Gaussian kernel the most expensive and the Bhattacharyya kernel in between the other two.

Table 4.5: Classification performance and cost for different binning strategies, SVM kernels. In bold the best results.

| | | Recall (TPR) | | | | Support Vectors (SV) | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Bins | Kernel | PPLive | TVAnts | SopCast | Joost | PPLive | TVAnts | SopCast | Joost | Total |
| Exponential | Gaussian | **81.66** | 98.51 | **89.62** | **89.47** | **1015** | **106** | **845** | **415** | **2381** |
| | Bhattacharyya | 77.73 | 98.52 | 88.58 | 87.99 | 1759 | 110 | 1185 | 798 | 3852 |
| | Linear | 73.44 | **98.54** | 88.55 | 87.42 | 2062 | 219 | 1348 | 956 | 4585 |
| Constant | Gaussian | **67.12** | **97.86** | **89.76** | **69.66** | **853** | **81** | **654** | **635** | **2223** |
| | Bhattacharyya | 65.27 | 97.14 | 89.58 | 68.27 | 1215 | 113 | 902 | 755 | 2985 |
| | Linear | 64.90 | 97.70 | 89.45 | 68.88 | 1382 | 316 | 911 | 1091 | 3700 |

Tab. 4.5 collects results highlighting the best choices using bold font. Irrespectively of the binning strategy, the Gaussian kernel yields consistently better results for both TPR and number of SVs. An important decrease in the performance is observed when considering constant binning, where the TPR for PPLive and Joost falls below the 70%. This is mostly due to the rejection criterion, which wrongly identifies as unknown a conspicuous number of signatures: indeed, the Bhattacharyya distance is less effective with this longer signature containing many zero values, which result in bigger distances from the class center. Results obtained with the Bhattacharyya kernel are almost equal to the linear kernel, with the advantage that the number of SV is smaller. Finally it must be noted that TVAnts requires a very small number of SV to obtain very good performance, irrespectively of the binning and kernel choice. In contrast PPLive, the most difficult application to classify, requires a number of SV that is ten times the number of TVAnts for its best choice of binning and kernel.

With respect to the bin distribution choice, the use of exponential binning reduces the memory consumption and the number of operations to be performed by $B_{fix}/B_{exp}$, i.e., almost a factor of 30. For example, assuming 1 GBytes of RAM, $B_{exp} = 9$ exponential bins would allow to compute about $15M$ end-points considering 64bit floating point notation. With the same amount of memory, using $B_{fix}$ linearly spaced bins allows to track roughly $0.5M$ end-points. Considering CPU time, a server equipped with an Intel Xeon E5345 clocked at 2.33GHz reaches 3000 classifications per second using exponentially distributed bins. Given that a signature is produced every $\Delta T = 5$ s, about 15000 end-points could be classified in real-time even by our non optimized Python code. Considering linearly distributed bins, only 126 classifications per second are computed, allowing to classify no more than 630 end-points.

Results from this analysis reinforces the selection of an exponential binning strategy in combination with the Gaussian kernel.

## 4.7 Improving the Accuracy: Extending the Signature

In this section we augment the Abacus signature to include not only the number of packets received by each peer, but also the number of received bytes. Following the same procedure, we consider a $\Delta T$ time interval in which the endpoint $\mathcal{P}$ receives $b_1, \ldots, b_K$ bytes from $K$ peers. $B_b + 1$ exponential-width classes are identified, according to the number of bytes received from $\mathcal{P}$, and counting the occurrences of each class in $B_i$. The byte-wise signature $\underline{b}$ is then obtained by normalizing the count $B_i$ over the total number of received bytes. The tuple $\underline{b}$ is a pmf, whose component $b_i$ can be interpreted as the probability that an arbitrary peer sends between $(2^{i-1}, 2^i]$ bytes to $\mathcal{P}$. For byte-wise signatures, we set the number of bins to $B_b = 14$.

We define the application signature by concatenating the packet-wise $\underline{n}$ and byte-wise $\underline{b}$ signatures in a single vector $\underline{a} = (\underline{n}, \underline{b})$. Since the extended signature $\underline{a} = (\underline{n}, \underline{b})$ is composed of

Table 4.6: Extended Abacus Signatures: Confusion Matrix of P2P-TV Application

| | Signatures: Confusion Matrix | | | | |
|---|---|---|---|---|---|
| | PPLive | TVAnts | SopCast | Joost | Unk |
| PPLive | **95.42** | 0.22 | 1.86 | 0.36 | 2.14 |
| TVAnts | 0.06 | **99.84** | 0.10 | 0.00 | 0.00 |
| SopCast | 0.98 | 0.15 | **97.55** | 0.03 | 1.29 |
| Joost | 0.21 | 0.01 | 0.01 | **94.97** | 4.80 |

Table 4.7: Extended Abacus Signatures: Classification Results per Signature, Packets, Bytes and End-Point

| | Signatures | | | Packets | | | Bytes | | | Peer | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | TP | Mis | Unk | TP | Mis | Unk | TP | Mis | Unk | TP | Unk (n) |
| PPLive | 95.42 | 2.44 | 2.14 | 98.11 | 1.60 | 0.29 | 98.32 | 1.54 | 0.14 | 100.0 | 0.0 (0) |
| TVAnts | 99.84 | 0.16 | 0.00 | 99.77 | 0.23 | 0.00 | 99.82 | 0.17 | 0.01 | 100.0 | 0.0 (0) |
| SopCast | 97.55 | 1.17 | 1.29 | 99.18 | 0.78 | 0.04 | 98.96 | 0.98 | 0.06 | 97.06 | 2.94 (1) |
| Joost | 94.97 | 0.23 | 4.80 | 99.50 | 0.25 | 0.25 | 99.62 | 0.23 | 0.15 | 93.33 | 6.67 (2) |

two parts, we can define two rejection thresholds, considering $\underline{n}$ or $\underline{b}$ only. We therefore report in Fig. 4.7 the TPR and FPR as a function of the rejection threshold $R$ applied to byte signatures. Contrasting Fig. 4.7 with Fig. 4.4, we observe that the bytes signatures exhibit a better behavior also with respect to the rejection criterion. In fact, on the one hand, TPR curves saturate much faster, which means that points of the same application are better clustered; on the other hand, the FPR curves start showing up for larger values of the threshold, which is even better because we can safely adopt a larger value for $R$, obtaining at the same time lower FPR and higher TPR. Given these considerations, we decided to apply the rejection criterion only to the byte-wise signatures $\underline{b}$ with a threshold $R = 0.6$.

We perform the classification based on the extended signature $\underline{a}$ with a byte-wise rejection criterion and a rejection threshold $R = 0.6$. Results reported in Tab. 4.6 are gathered for $\Delta T = 5$s, with a training set of 4000 signatures extracted at random. Compared to previous results of Tab. 4.3, the extended signature leads to significant performance improvement, so that TPR is now about 95% in the worst case, and misclassification probability is reduced to few percentage points.

To better appreciate results, Tab. 4.7 reports performance considering correctly classified *packets*, *bytes* and *peers*. Packet-wise and byte-wise performance can be directly gathered by taking into account the number of packets and bytes carried by each signature; the peer classification is instead evaluated considering a *majority* criterion, so that a peer is classified as running application $X$ if the majority of time such peer samples have been classified as $X$. Tab. 4.7 reports the percentage of correct classification (TP), of misclassification (Mis, corresponding to the sum by rows of non diagonal values in the confusion matrix) and rejection (Unk) for all the above metrics. Notice that FN=Mis+Unk. Interestingly, performance improves when the number of correctly classified *packets* and *bytes* is considered, suggesting that misclassification occurs when signatures carry few data, e.g., when the application is possibly malfunctioning. In case of *peer* classification, reliability of end-points identification increases as well. Only 3 hosts are classified as not running any P2P-TV application, and notably there is no misclassification. Investigating further, we found that rejected cases correspond to peers that received a small amount of traffic, and, thus, possibly were not playing any video.

We now assess the benefits of the extended signatures on the effectiveness of the rejection criterion. We again consider real traffic collected from operational networks, considering only the

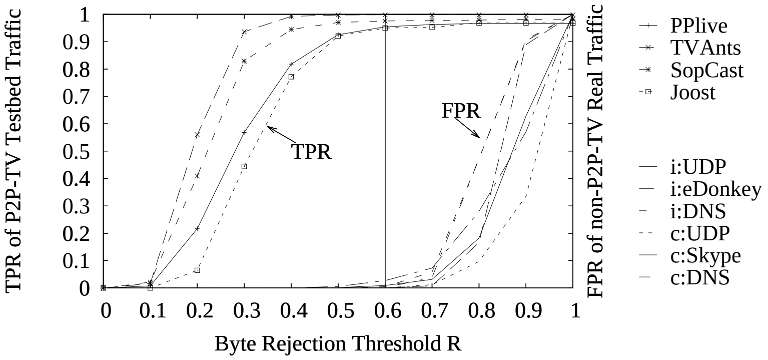


Figure 4.7: TPR and FPR as a function of the rejection threshold $R$ evaluated on bytes feature.

Table 4.8: Non P2P-TV Traffic in Campus and ISP traces: False Positive Ratio FPR and FP Confusion Matrix. To perform a worst-case analysis, only end-points that can lead to false positive classification are considered (28% of the CAMPUS and 15% of the ISP overall traffic volume)

| | | | FP Confusion Matrix | | | |
|---|---|---|---|---|---|---|
| Traffic | | FPR | PP | TV | SO | JO |
| C | UDP | **2.70** | 0.57 | 1.00 | 1.13 | - |
| | Skype | 0.04 | - | 0.03 | 0.01 | - |
| | DNS | 0.17 | 0.02 | 0.10 | 0.05 | - |
| I | UDP | **0.90** | 0.61 | 0.14 | 0.15 | - |
| | eDonkey | 0.09 | 0.02 | 0.04 | 0.03 | - |
| | DNS | 0.44 | 0.03 | 0.33 | 0.08 | - |

worst-case traffic portion for which possible false positives may be triggered. Tab. 4.8 reports results referring to the extended signatures, showing the false positive rate (FPR) and the breakdown of false alarms between the different P2P-TV applications. First, notice that the number of false alarms is very limited, being only 2.7% in the worst-case traffic subset: however, if all UDP traffic is considered, FPR drops to less than 0.1%. This negligible number of false alarms confirms the reliability of the classification engine. Moreover, false positive rate is low for individual applications too: indeed, it is very rare that eDonkey or Skype traffic is confused with any P2P-TV application (0.09% and 0.04% of false positives).

## 4.8 Summary

In this chapter we have proposed Abacus, a novel behavioral approach for fine-grained classification of P2P-TV applications. Our methodology relies only on the simple count of packets and bytes exchanged amongst peers during small time-windows. Our classification engine, which makes use of Support Vector Machines during the decision process, correctly classifies about 95% of packets, bytes and peers in the worst case. Moreover, the classification engine raises very few false alarms, well below 0.1% in the worst case. Such astonishing performance is the result, on the one hand, of the discriminative power of SVM, and, on the other hand, of the descriptive expressiveness of Abacus signatures.

A large set of experiments has been carried over to assess Abacus performance, both considering parameter sensitivity, and signature portability: results prove that the proposed approach is very robust to both. Training the Abacus classifier is simple, as signatures can be generated automatically using a very small number of traces. In terms of both memory requirements and computational complexity, Abacus is also very lightweight. Moreover, the fact that behavioral data used by Abacus are directly available from commonly deployed NetFlow monitors makes it apt to be deployed in real network environments.

These results, though extremely promising, also raise a number of interesting point which we will evaluate in the following chapters .First of all, while we proved Abacus to be extremely reliable on P2P-TV traffic, it would be interesting to investigate whether Abacus can classify applications of other P2P classes as well, as we will prove possible in Chap. 6. Second, the behavioral statistics on which Abacus decisions are taken are extremely reliable when the classification is applied to *all* the traffic observed by an endpoint. In Chap. 6, we will also investigate what happens when the classification engine is moved from the access deeper inside the aggregation network (e.g., at the first or second IP router), where only a *subset* of all flows can be observed. We will show that since Abacus signature are normalized, if the subset of observed flows contains both long and short flows, not biasing the overall distribution, then classification is still possible with just a minor reduction in accuracy.

In the next chapter we will compare the performance of Abacus with a packet-based algorithm, also tailored for P2P live streaming applications.

# Chapter 5

# Comparing behavioral and payload based classification algorithms

In this chapter, whose results have been published in [79], we compare the Abacus behavioral classifier with a stochastic payload based classifier, namely Kiss [77]. This classifier bases the classification on the examination of the first bytes of the application-layer payload. It has already been compared with other classifiers in [47], proving to be the best one for this specific class of traffic.

The motivation for this comparison lies in the skepticisms of the operational community towards behavioral classifiers. Though recognizing their advantages – less computational requirements and the ability to deal with encrypted/closed protocols – the lack of trustworthy comparison with traditional techniques discourages their adoption. In fact, since each behavioral classifier is tested on a different set of traces, under different conditions and often using different metrics, it is really difficult for a network operator to identify which methods could best fit its needs.

We test Abacus and Kiss on an common set of traces, evaluating their accuracy in terms of both true positives (i.e., correct classification of P2P-TV traffic) and true negatives (i.e., correct identification of traffic other than P2P-TV). We also provide a detailed comparison of theirs features, focusing mostly on the differences which stem from the undertaken approaches. Moreover, we formally investigate the computational complexity by comparing the memory occupation and the computational costs.

Results show that Abacus achieves practically the same performance of Kiss and both classifiers exceed 99% of correctly classified bytes for P2P-TV traffic. Abacus exhibits some problems in terms of flow accuracy for one specific application, for which it still has a high bytewise accuracy. The two algorithms are also very effective when dealing with non P2P-TV traffic, raising a negligible number of false negatives. Finally we found that Abacus outperforms Kiss in terms of computation complexity, while Kiss is a much more general classifier, able to work with a wider range of protocols and network conditions.

The remainder of this chapter is organized as follows. In Sec. 5.1 we briefly present the two techniques under exam, then in Sec. 5.2 we test them on a common set of traces and compare their performance. We proceed with a more qualitative comparison of the classifiers in Sec. 5.3 as well as an evaluation of their computational cost. In Sec. 5.4, we briefly describe the demo software we developed, which implements the two classifiers and allows to compare their performance when running on live traffic. Finally Sec. 5.5 summarizes the chapter.

## 5.1 Classification algorithms

In this section we introduce the two classifiers. Since Abacus was thoroughly presented in Chap. 4, here we just recall a few relevant aspects for the comparison. Likewise, for Kiss, we provide only those detail needed to fully understand the results of our experiments, while we refer the reader to [77] for more information.

Both Kiss and Abacus employ supervised machine learning as their decision process, in particular Support Vector Machine - SVM [62], which has already been proved particularly suited for traffic classification [105]. In the SVM context, entities to be classified are described by an ordered set of *features*, which can be interpreted as coordinates of points in a multidimensional space. Kiss and Abacus differ for the choice of the features. The SVM must be trained with a set of previously labeled points, commonly referred to as the *training set*. During the training phase, the SVM basically defines a mapping between the original feature space and a new space, usually characterized by an higher dimensionality, where the training points could be separated by hyperplanes. In this way, the target space is subdivided in areas, each associated to a specific class. During the classification phase, a point can be classified simply looking for the region which best fits it.

Before proceeding with the description of the classifiers, it is worth analyzing their common assumption. First of all, they both classify *endpoints*, i.e., couples (IP address, transport-layer port) on which a given application is running. Second, they were originally designed for on UDP traffic, since this is the transport-layer protocol generally chosen by P2P-TV applications. Finally, given that they rely on a machine learning process, namely Support Vector machines, they follow the usual workflow of such a family of algorithm, explained in Sec. 2.2.1. As a first step, the engines derive a signature vector from the analysis of traffic relative to the target endpoint. Then, they feed the vector to the trained SVM, which in turn gives the classification result. Once an endpoint has been identified, all the flows which have that endpoint as source or destination are labeled as being generated by the identified application.

### 5.1.1 Abacus

A full description of the Abacus classifier as well as its key idea are provided in Chap. 4. Here we just recall the main procedure followed by the algorithm to build the signatures, which we reported as pseudo-code in Tab. 5.4.

As you may remember, Abacus signatures are based on the number of contacted peers and the amount of exchanged information among them, measured in number of packets and bytes. This simple measure highlights the distinct behaviors of the different P2P-TV applications. Indeed, an application which implements an aggressive peer-discovering strategy will receive many single-packet probes, consequently showing large values for low order bins. Conversely, an application which downloads the video stream using chunks of, say, 64 packets will exhibit a large value of the 6-th bin.

Let us focus on the packet counters. We first define a partition of $\mathbb{N}$ in $B$ exponential-sized bins $I_i$, i.e. $I_0 = [0, 1]$, $I_i = [2^{i-1} + 1, 2^i]$ and $I_B = [2^B, \infty)$. Then, we order the observed peers in bins according to the number of packets they have sent to the given endpoint. In the pseudo-code we see that we can assign a peer to a bin by simply calculating the logarithm of the associated number of packets. We proceed in the same way also for the byte counters (except that we use a different set of bins), finally obtaining two vectors of frequencies, namely $p$ and $b$. The concatenation of the two vectors is the Abacus signature which is fed to the SVM for the actual decision process.

Table 5.1: Datasets used for the comparison

| Dataset | Duration | Flows | Bytes | Endpoints |
|---|---|---|---|---|
| Napa-WUT | 180 min | 73k | 7Gb | 25k |
| Operator 2006 (op06) | 45 min | 785k | 4Gb | 135k |
| Operator 2007 (op07) | 30 min | 319k | 2Gb | 114k |

Finally, Abacus provides a simple mechanism (see Sec. 4.3.2) to identify applications which are “unknown” to the SVM (i.e., not present in the training set), which in our case means non P2P-TV applications. Basically, for each class we define a centroid based on the training points, and we label a signature as unknown if its distance from the centroid of the associated class exceeds a given threshold. To evaluate this distance we use the Bhattacharyya distance, which is specific for probability mass functions.

### 5.1.2 Kiss

The Kiss classifier [77] is instead based on a statistical analysis of the packets payload. In particular, it exploits a *Chi-Square* like test to extract statistical features from the first application-layer payload bytes. Considering a window of $C$ segments sent (or received) by an endpoint, the first $k$ bytes of each packet payload are split into $G$ groups of $b$ bits. Then, the empirical distributions $O_i$ of values taken by the $G$ groups over the $C$ segments are compared to a uniform distribution $E_i = C/2^b$ by means of the Chi-Square like test:

$$X_g = \sum_{i=1}^{2^b} \frac{(O_i^g - E)^2}{E} \qquad g \in [1, G] \tag{5.1}$$

This allows to measure the randomness of each group of bits and to discriminate among constant/random values, counters, etc. as the Chi-Square test assumes different values for each of them. The array of the $G$ Chi-Square values defines the application signature. In this chapter, we use the first $k = 12$ bytes of the payload divided into groups of 4 bits (i.e., $G = 24$ features per vector) and $C = 80$ segments to compute each Chi-Square.

The generated signatures are then fed to a multi-class SVM machine, similarly to Abacus. As previously stated, a training set is used to characterize each target class, but for Kiss an additional class must be defined to represent the remaining traffic, i.e., the *unknown* class. In fact, a multi-class SVM machine always assigns a sample to one of the known classes, in particular to the best fitting class found during the decision process. Therefore, in this case a trace containing only traffic other than P2P-TV is needed to characterize the unknown class. We already mentioned that in Abacus this problem is solved by means of a threshold criterion using the distance of a sample from the centroid of the class. We refer the reader to [77] for a detailed discussion about Kiss parameter settings and about the selection of traffic to represent the unknown class in the training set.

Table 5.2: Classification results

(a) Flows

Abacus

| | pp | tv | sp | jo | un |
|---|---|---|---|---|---|
| pp | **13.35** | 0.32 | - | 0.06 | 86.27 |
| tv | 0.86 | **95.67** | 0.15 | - | 3.32 |
| sp | 0.33 | 0.03 | **98.04** | 0.1 | 1.5 |
| jo | 0.06 | 2.21 | - | **81.53** | 16.2 |
| op06 | 0.1 | 0.1 | 1.03 | 0.06 | **98.71** |
| op07 | 0.21 | 0.03 | 0.87 | 0.05 | **98.84** |

Kiss

| | pp | tv | sp | jo | un | nc |
|---|---|---|---|---|---|---|
| pp | **98.8** | - | - | - | 0.2 | 1 |
| tv | - | **97.3** | - | 0.01 | 0.69 | 2 |
| sp | - | - | **98.82** | - | 0.21 | 0.97 |
| jo | - | - | - | **86.37** | 3.63 | 10 |
| op06 | - | 0.44 | 0.08 | 0.55 | **92.68** | 6.25 |
| op07 | - | 2.13 | 0.09 | 1.21 | **84.07** | 12.5 |

(b) Bytes

Abacus

| | pp | tv | sp | jo | un |
|---|---|---|---|---|---|
| pp | **99.33** | - | - | 0.11 | 0.56 |
| tv | 0.01 | **99.95** | - | - | 0.04 |
| sp | 0.01 | 0.09 | **99.85** | 0.02 | 0.03 |
| jo | | - | - | **99.98** | 0.02 |
| op06 | 1.02 | - | 0.58 | 0.55 | **97.85** |
| op07 | 3.03 | - | 0.71 | 0.25 | **96.01** |

Kiss

| | pp | tv | sp | jo | un | nc |
|---|---|---|---|---|---|---|
| pp | **99.97** | - | - | - | 0.01 | 0.02 |
| tv | - | **99.96** | - | - | 0.03 | 0.01 |
| sp | - | - | **99.98** | - | 0.01 | 0.01 |
| jo | - | - | - | **99.98** | 0.01 | 0.01 |
| op06 | - | 0.07 | - | 0.08 | **98.45** | 1.4 |
| op07 | - | 0.08 | 0.74 | 0.05 | **96.26** | 2.87 |

pp=PPLive, tv=Tvants, sp=Sopcast, jo=Joost, un=Unknown, nc=not-classified

## 5.2 Experimental Results

### 5.2.1 Methodology and Datasets

To evaluate the two classifier we used a subset of the traces already used to evaluate the Abacus classifier in the previous chapter (cfr. Sec. 4.4). Again we use two distinct sets of traces, both described in Sec. 2.4, passively and actively collected, to asses two different aspects of our classifiers.

First we used the traces coming from a single vantage point of the European testbed described in Sec. 2.4, in particular those coming from the Warsaw University of Technology. We consider the same four applications previously studied, namely PPLive, TVAnts, SopCast and Joost. This set is used both to train the classifiers and to evaluate their performance in identifying the different P2P-TV applications.

The second dataset consists of two real-traffic traces collected in 2006 and 2007 on the network of a large Italian ISP. Given the extremely rich set of channels available through the ISP streaming services, customers are not inclined to use P2P-TV applications and actually no such traffic is present in the traces. We verified this by means of a classic DPI classifier as well as by manual inspection of the traces. This set has the purpose of assessing the number of false alarms raised by the classifiers when dealing with non P2P-TV traffic. We report in Tab. 5.1 the main characteristics of the traces.

To compare the classification results, we employ the `diffinder` tool [148], as already done in [47] . This simple software takes as input the logs from different classifiers with the list of flows and the associated classification outcome. Then, it calculates as output several aggregate metrics, such as the percentage of agreement of the classifiers in terms of both flows and bytes, as well as a detailed list of the differently classified flows, so eventually enabling further analysis.

### 5.2.2 Classification results

Tab. 5.2 reports the accuracy achieved by the two classifiers on the test traces, employing the usual confusion matrix representation. For each table, the upper part is related to the Napa-Wine traces

Table 5.3: Main characteristics of Abacus and Kiss

| Characteristic | Abacus | Kiss |
|---|---|---|
| Technique | Behavioral | Stocastic Payload Inspection |
| Entity | Endpoint | Endpoint/Flow |
| Input Format | Netflow-like | Packet trace |
| Grain | Fine grained | Fine grained |
| Protocol Family | P2P-TV | Any |
| Rejection Criterion | Threshold | Train-based |
| Train set size | Big *(4000 smp.)* | Small *(300 smp.)* |
| Time Responsiveness | Deterministic *(5sec)* | Stochastic *(early 80pkts)* |
| Network Deploy | Edge | Edge/Backbone |

while the lower part is dedicated to the operator traces. The values in bold on the main diagonal of the tables express the *recall* (see also Sec. 2.3), defined as the ratio of true positives over the sum of true positives and false negatives. The “unknown” column counts the percentage of traffic which was recognized as not being P2P-TV traffic, while the column “not classified” accounts for the percentage of traffic that Kiss cannot classify as it needs at least 80 packets for each endpoint.

At first glance, both classifiers are extremely accurate in terms of bytes. For the Napa-Wine traces the percentage of true positives exceeds 99% for all the considered applications. For the operator traces, again the percentage of true negatives exceeds 96% for all traces, with Kiss showing a overall slightly better performance. These results demonstrate that even an extremely lightweight behavioral classification mechanism, such as the one adopted in Abacus, can achieve the same precision of an accurate payload based classifier.

If we consider flow accuracy, we see that for three out of four applications the performance of the two classifiers is comparable. Yet Abacus presents a very low percentage of 13.35% true positives for PPLive, with a rather large number of flows falling in the unknown class. By examining the classification logs, we found that PPLive actually uses more ports on the same host to perform different functions (e.g. one for video transfer, one for overlay maintenance). In particular, from one port it generates many single-packet flows all directed to different peers, apparently to perform peer discovery. All these flows, which account for a negligible portion of the overall bytes, fall in the first bin of the abacus signature, which is always classified as unknown. However, from the byte-wise results we can conclude that the video endpoint is always correctly classified.

Finally, we observe that Kiss has a lower flow accuracy for the operator traces. In fact, the great percentage of flows falling in the “not classified” class means that many flows are shorter than 80 packets. Again, this is only a minor issue since Kiss byte accuracy is however very high.

## 5.3 Comparison

### 5.3.1 Functional Comparison

In the previous section we have shown that the classifiers have similar performance for the identification of the target applications as well as the “unknown” traffic. Nevertheless, they are based on very different approaches, both presenting pros and cons, which need to be all carefully taken into account.

Tab. 5.3 summarizes the main characteristics of the classifiers, which are reviewed in the following. The most important difference is the classification technique used. Even if both classifiers are statistical, they work at different levels and clearly belong to different families of classification algorithms. Abacus is a behavioral classifier since it builds a statistical representation of the pattern of traffic generated by an endpoint, starting from transport-level data. Conversely, Kiss derives a statistical description of the application protocol by inspecting packet-level data, so it is a payload-based classifier.

The first consequence of this different approach lies in the type and volume of information needed for the classification. In particular, Abacus takes as input just a measurement of the traffic rate of the flows directed to an endpoint, in terms of both bytes and packets. Not only this represents an extremely small amount of information, but it could also be gathered by a Netflow monitor, so that no packet trace has to be inspected by the classification engine itself. On the other hand, Kiss requires to access packet payload to compute its features. This constitutes a more expensive operation, even if only the first 12 bytes are enough to achieve a high classification accuracy.

Despite the different input data, both classifiers work at a fine-grained level, i.e., they can identify the specific application related to each flow and not just the class of applications (e.g., P2P-TV). This consideration may appear obvious for a payload-based classifier such as Kiss, but it is one of the strength of Abacus over other behavioral classifiers which are usually capable only of a coarse grained classification.

Clearly, Abacus pays the simplicity of its approach in terms of possible target traffic. In fact its classification process relies on some specific properties of P2P-TV traffic (i.e., the steady download rate required by the application to provide a smooth video playback), which are really tied to this particular service. For this reason Abacus currently cannot be applied to applications other than P2P-TV applications. On the contrary, Kiss is more general, it makes no particular assumptions on its target traffic and can be applied to any protocol. Indeed, it successfully classifies other kinds of P2P applications, from file-sharing (e.g., eDonkey) to P2P VoIP (e.g., Skype), as well as traditional client-server applications (e.g., DNS).

Another important distinguishing element is the rejection criterion. Abacus defines an hypersphere for each target class and measures the distance of each classified point from the center of the associated hypersphere by means of the Bhattacharyya formula. Then, by employing a threshold-based rejection criterion, a point is label as “unknown” when its distance from the center exceeds a given value. Instead Kiss exploits a multi-class SVM model where all the classes, included the unknown, are represented in the training set. If this approach makes Kiss very flexible, the characterization of the classes can be critical especially for the unknown since it is important that the training set contains samples from all possible protocols other than the target ones.

We also notice that there is an order of magnitude of difference in the size of the training set used by the classifiers. In fact, we trained Abacus with 4000 samples per class (although in some tests we experimented the same performance even with smaller sets) while Kiss, thanks to the combination of the discriminative power of both the ChiSquare signatures and the SVM decision process, needs only 300 samples per class.

On the other hand, Kiss needs at least 80 packets generated from (or directed to) an endpoint in order to classify it. This may seem a difficult requirement to meet: yet results reported in Sec. 5.2 actually show that the percentage of not supported traffic is negligible, at least in terms of bytes. This is due to the adoption of the endpoint-to-flow label propagation scheme, i.e. the propagation of the label of an “elephant” flow to all the “mice” flows of the same endpoint. With the exception of particular traffic conditions, this labeling technique can effectively bypass the constraint on the number of packets.

Finally, for what concerns the network deployment, Abacus needs all the traffic received by the

Table 5.4: Analytical comparison of the resource requirements of the classifiers

| | Abacus | Kiss |
|---|---|---|
| Memory allocation | 2F *counters* | $2^b$G *counters* |
| Packet processing | EP_state = hash($IP_d$, $port_d$)<br>FL_state = EP_state.hash($IP_s$, $port_s$)<br>FL_state.pkts ++<br>FL_state.bytes += pkt_size | EP_state = hash($IP_d$, $port_d$)<br>**for** g = 1 to G **do**<br>  $P_g$ = payload[g]<br>  EP_state.O[g][$P_g$]++<br>**end for** |
| *Tot. op.* | 2 $lup$ + 2 $sim$ | (2G+1) $lup$ + G $sim$ |
| Feature extraction | EP_state = hash($IP_d$, $port_d$)<br>**for all** FL_state in EP_state.hash **do**<br>  p[$log_2$(FL_state.pkts )]+=incr<br>  b[$log_2$(FL_state.bytes)]+=incr<br>**end for**<br>EP_state = hash($IP_d$, $port_d$)<br>**for all** FL_state in EP_state.hash **do**<br>  p[ $log_2$(FL_state.pkts )] += 1<br>  b[ $log_2$(FL_state.bytes)] += 1<br>**end for**<br>N = count(keys(EP_state.hash))<br>**for all** i = 0 to B **do**<br>  p[i] /= N<br>  b[i] /= N<br>**end for** | E = C/$2^b$ *(precomputed)*<br>**for** g = 1 to G **do**<br>  Chi[g] = 0<br>  **for** i = 0 to $2^b$ **do**<br>    Chi[g] += (EP_state.O[g][i]-E)$^2$<br>  **end for**<br>  Chi[g] /= E<br>**end for** |
| *Tot. op.* | (4F+2B+1) $lup$ + 2(F+B) $com$ + 3F $sim$ | $2^{b+1}$G $lup$ + G $com$ + (3·$2^b$+1)G $sim$ |

lup=lookup, com=complex operation, sim=simple operation

endpoint to characterize its behavior. Therefore, it is only effective when placed at the edge of the network, where all traffic directed to a host transits. Conversely, in the network core Abacus would likely see only a portion of this traffic, so gathering an incomplete representation of an endpoint behavior, which in turn could result in an inaccurate classification. Kiss, instead, is more robust with respect to the deployment position. In fact, by inspecting packet payload, it can operate even on a limited portion of the traffic generated by an endpoint, provided that the requirement on the minimum number of packets is satisfied.

### 5.3.2 Computational Cost

To complete the classifiers comparison, we provide an analysis of the requirements in terms of both memory occupation and computational cost. We follow a theoretical approach and calculate these metrics from the formal algorithm specification. In this way, our evaluation is independent from specific hardware platforms or code optimizations. Tab. 5.4 compares the costs from an analytical point of view while in Tab. 5.5 there is a numerical comparison based on a case study.

Memory footprint is mainly related to the data structures used to compute the statistics. Kiss requires a table of $G \cdot 2^b$ counters for each endpoint to collect the observed frequencies employed in the chi-square computation. For the default parameters, i.e. $G = 24$ chunks of $b = 4$ bits, each endpoint requires 384 counters. Abacus, instead, requires two counters for each flow related to an

Table 5.5: Numerical case study of the resource requirements of the classifiers

| | Abacus | Kiss |
|---|---|---|
| Memory allocation | 320 bytes | 384 bytes |
| Packet processing | 2 $lup$ + 2 $sim$ | 49 $lup$ + 24 $sim$ |
| Feature extraction | 177 $lup$ + 96 $com$ + 120 $sim$ | 768 $lup$ + 24 $com$ + 1176 $sim$ |
| *Params values* | *B=8, F=40* | *G=24, b=4* |

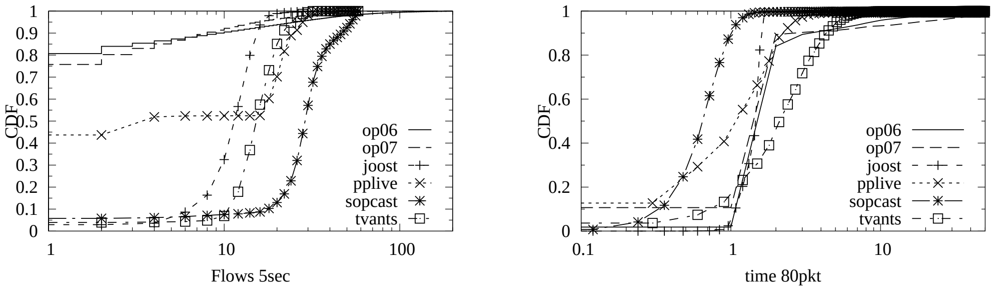
endpoint, so the total amount of memory is not fixed but it depends on the number of flows per endpoint. As an example, Fig. 5.1-(a) reports, for the two operator traces, the CDF of the number of flows seen by each endpoint in consecutive windows of 5 seconds, the default duration of the Abacus time-window. It can be observed that the 90th percentile in the worst case is nearly 40 flows. By using this value as a worst case estimate of the number of flows for a generic endpoint, we can say that $2 \cdot \#Flows = 80$ counters are required for each endpoint. This value is very small compared to Kiss requirements but for a complete comparison we also need to consider the counters dimension. As Kiss uses windows of 80 packets, its counters assume values in the interval $[0, 80]$ so single byte counters are sufficient. Using the default parameters, this means 384 bytes for each endpoint. Instead, the counters of Abacus do not have a specific interval so, using a worst case scenario of 4 bytes for each counter, we can say that 320 bytes are associated to each endpoint. In conclusion, in the worst case, the two classifiers require a comparable amount of memory but on average Abacus requires less memory than Kiss.

Computational cost can be evaluated comparing three tasks: the operations performed on each packet, the operations needed to compute the signatures and the operations needed to classify them. Tab. 5.4 reports the pseudo code of the first two tasks for both classifiers, specifying also the total amount of operations needed for each task. The operations are divided in three categories and considered separately as they have different costs: *lup* for memory lookup operations, *com* for complex operations (i.e., floating point operations), *sim* for simple operations (i.e., integer operations).

Let us first focus on the packet processing part, which presents many constraints from a practical point of view, as it should operate at line speed. In this phase, Abacus needs 2 memory lookup operations, to access its internal structures, and 2 integer increments per packet. Kiss, instead, needs $2G + 1 = 49$ lookup operations, half of which are accesses to packet payload. Then, Kiss must compute $G$ integer increments. Since memory read operations are the most time consuming, from our estimation we can conclude that Abacus should be approximately 20 times faster than Kiss in the packet processing phase.

The evaluation of the signature extraction process instead is more complex. First of all, since the number of flows associated to an endpoint is not fixed, the Abacus cost is not deterministic but, like in the memory occupation case, we can consider 40 flows as a worst case scenario. For the lookup operations, Considering $B = 8$, Abacus requires a total of 177 operations, while Kiss needs 768 operations, i.e., nearly four times as many. For the arithmetic operations, Abacus needs 96 floating point and 120 integer operations, while Kiss needs 24 floating point and 1176 integer operations.

Abacus produces one signature every 5 seconds, while Kiss signatures are processed every 80 packets. To estimate the frequency of the Kiss calculation, in Fig. 5.1(b) we show the CDF of the amount of time needed to collect 80 packets for an endpoint. It can be observed that, on average, a new signature is computed every 2 seconds. This means that Kiss performs the feature calculation more frequently, i.e., it is more reactive and possibly more accurate than Abacus but obviously



(a) (b)

Figure 5.1: Cumulative distribution function of (a) number of flows per endpoint and (b) duration of a 80 packet snapshot for the operator traces

also more resource consuming.

Finally, the complexity of the classification task depends on the number of features per signature, since both classifiers are based on a SVM decision process. The Kiss signature is composed, by default, of $G = 24$ features, while the Abacus signature contains 16 features: also from this point of view Abacus appears lighter than Kiss.

## 5.4 Demo software

We here describe the demo software [78] presented at IEEE Globecom 2010 [1], which shows the Abacus classifier and the Kiss classifier in action, allowing to perform live traffic classification with either engine as well as to compare their performance on the same traffic.

A full-fledged implementation of both classifiers is included in the demo software. Basically the demo runs the two classification engines concurrently on the same traffic, which is either read from a recorded trace or directly captured live from a network interface. The data used internally by the classification algorithms is exported at different points of execution, before being used for the final classification. A user-friendly GUI (i) displays these data in a number of graphs updated in real time and (ii) allows the user to interact with the classification processes.

At the beginning the user is presented with three windows, one for each classification algorithm, which is run independently from the other one, and one to show the resource consumption of the two classification processes. The standard configuration of the demo is showed in Fig. 5.2-(a): a host runs the P2P-TV applications targeted by our classifiers, the traffic is captured at the access point to the Internet and is inspected by the classifiers. During the demo at the conference, given that the busy wifi network did not allow to run the P2P live streaming smoothly, we rather used a prerecorded trace; therefore we were able to report the transport-layer ports used by each application, but such information is not used during the classification.

When running the demo, the user first selects the socket (i.e. IP address and UDP port) which will be the target of the classification: traffic exchanged by this socket is then processed and classified. As it can be seen in the screenshots in Fig. 5.2, the result of the classification is constantly displayed and updated in the bottom part of the windows, as a percentage of correctly classified signatures and bytes.

[1] The code of the demo is publicly available at `http://perso.telecom-paristech.fr/~drossi/index.php?n=Software.ClassificationDemo`

Fig. 5.2 shows two examples of the kind of data displayed by the demo software. In particular Fig. 5.2-(d) is related to the Kiss classifier, and displays the evolution over time of the Chi-Square metric calculated for the first 24 groups of 4 bits in the packet payload. The top part plot is related to traffic received by the target endpoint, while the bottom part is related to traffic sent by the endpoint. Color is used to denote the level of randomness of the chunk: red chunks correspond to constant fields, whereas purple, blue, sky blue and yellow represent increasing levels of randomness. In the picture you can clearly distinguish two different pattern in both the top and the bottom plots: in fact we purposely capture the moment when the user has changed the socket to inspect, so the two patterns are related to two different application (namely TVAnts and SopCast). The difference between the two patterns is pretty evident even with naked eye and it is then exploited by the Support Vector Machine to actually classify the traffic.

Fig. 5.2-(c) shows, instead, the temporal evolution of the Abacus signature, in particular the packet-wise portion. We recall that the signature is actually a p.d.f., and, more specifically, the distribution of peers according to the number of packets sent to the target socket. In our representation, each bin of the distribution is associated with a different color and bins are stacked one over the other, building a well-defined and rather stable pattern. As in the Kiss picture, we captured the moment after the user selected a new socket, to show the pattern of two different application (again TVAnts and SopCast). Again, the pattern are extremely unlike one another, which means that the distribution is characteristic of each single application and, hence, good discriminators as well.

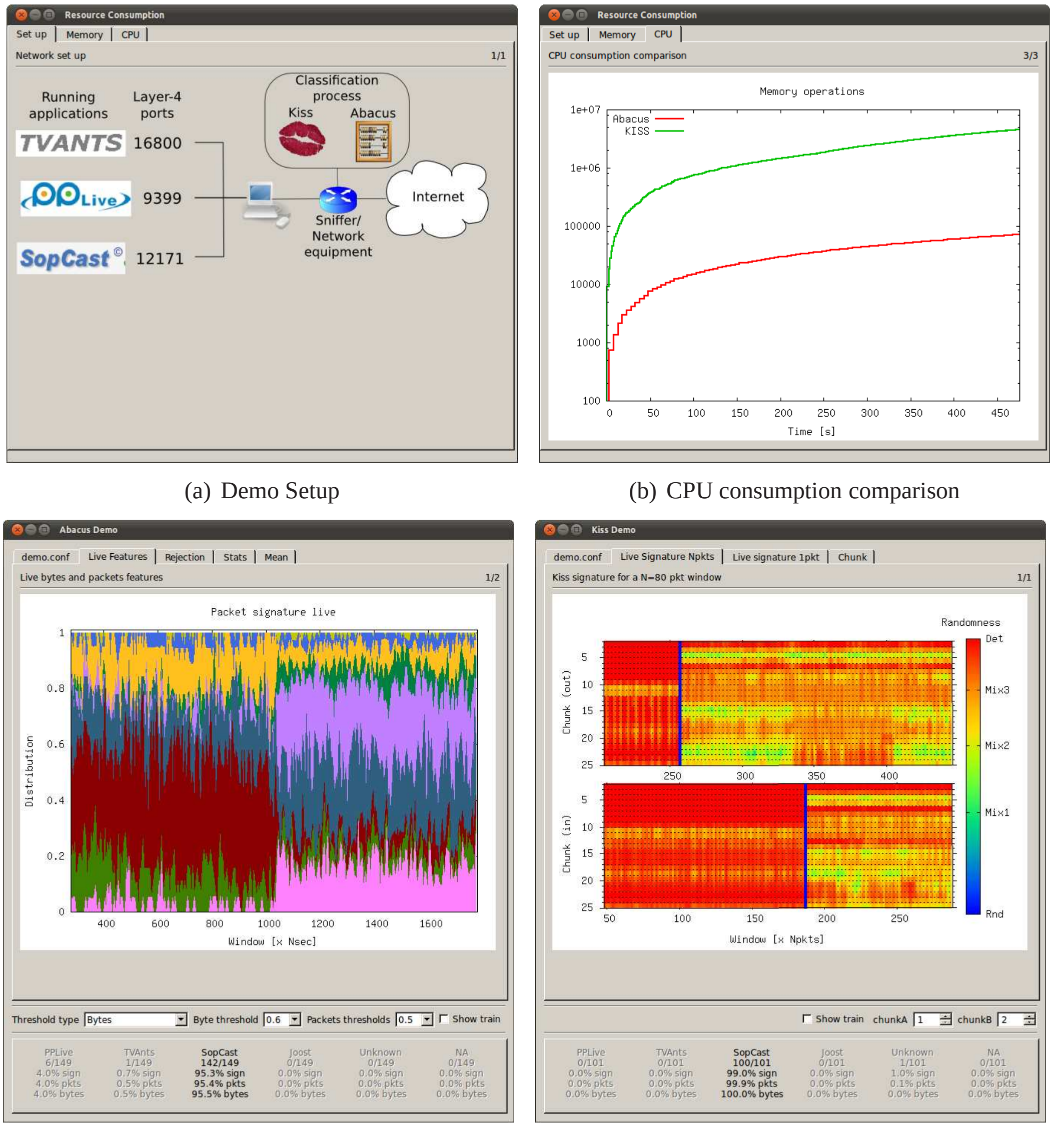
Notice also that both windows contain some controls, just above the classification results. By changing their setting the user can tweak the parameters of the classifiers (e.g. the rejection criterion threshold for Abacus), or select the information showed by the demo (e.g., to toggle the display of an additional window, which provides a view on the training set used by the classification engines).

Finally, Fig. 5.2-(b) is the window where the demo shows the information about the computational requirements of the classifiers. We show both the memory occupied by the classifier to keep track of the flows and compute the signatures (i.e., the chi-square values for Kiss, and the packet and byte counters for Abacus) and the number of operations required, divided in integer, floating point and memory access operations. As an example, in the picture we actually show the number of memory accesses performed by the two classifiers over time: in this run you can see that Abacus requires two orders of magnitude less accesses to classify the traffic.

## 5.5 Summary

In this chapter we compared two approaches to the classification of P2P-TV traffic. We provided not only a quantitative evaluation of the algorithm performance by testing them on a common set of traces, but also a more insightful discussion of the differences deriving from the two followed paradigms.

The algorithms proved to be comparable in terms of accuracy in classifying P2P-TV applications, at least regarding the percentage of correctly classified bytes. Differences emerged, though, when we compared the computational cost of the classifiers. With this respect, Abacus outperforms Kiss, because of the simplicity of the features employed to characterize the traffic. For this reason Abacus is a perfect candidate for scenarios with hard constraints in term of computational resources. Conversely, Kiss, though more costly from a computational point of view, is much more general, as it can classify other types of applications as well.



Threshold type Bytes   Byte threshold 0.6   Packets thresholds 0.5   □ Show train

| PPLive | TVAnts | SopCast | Joost | Unknown | NA |
|---|---|---|---|---|---|
| 6/149 | 1/149 | **142/149** | 0/149 | 0/149 | 0/149 |
| 4.0% sign | 0.7% sign | **95.3% sign** | 0.0% sign | 0.0% sign | 0.0% sign |
| 4.0% pkts | 0.5% pkts | **95.4% pkts** | 0.0% pkts | 0.0% pkts | 0.0% pkts |
| 4.0% bytes | 0.5% bytes | **95.5% bytes** | 0.0% bytes | 0.0% bytes | 0.0% bytes |

□ Show train   chunkA 1   chunkB 2

| PPLive | TVAnts | SopCast | Joost | Unknown | NA |
|---|---|---|---|---|---|
| 0/101 | 0/101 | **100/101** | 0/101 | 1/101 | 0/101 |
| 0.0% sign | 0.0% sign | **99.0% sign** | 0.0% sign | 1.0% sign | 0.0% sign |
| 0.0% pkts | 0.0% pkts | **99.9% pkts** | 0.0% pkts | 0.1% pkts | 0.0% pkts |
| 0.0% bytes | 0.0% bytes | **100.0% bytes** | 0.0% bytes | 0.0% bytes | 0.0% bytes |

(a) Demo Setup

(b) CPU consumption comparison

(c) Live Abacus signatures

(d) Live Kiss signatures

Figure 5.2: Screenshots from the running demo software.

# Part II

# Traffic Classification and data reduction

# Chapter 6

# Behavioral classification with reduced data: Netflow and flow-sampling

The second part of this thesis is dedicated to the impact of data aggregation and sampling to traffic classification. In particular this first chapter, whose results have been published in [158] and [159] tests our behavioral classifier in more challenging scenarios. First, since we claimed from the beginning that NetFlow flow-records are enough for the Abacus classifier, we actually test it with these aggregated data, facing a few difficulties with the coarser time-granularities. Second, we address one point raised in Chap. 4 about the placement of the classifier in the network: if we move inside the network core, because of routing, only a subset of traffic flows directed to the target host is observed by the classifier. We study then the performance of Abacus in presence of flow-sampling.

This chapter is organized as follows. In Sec. 6.1, after introducing NetFlow flow-records and the modification needed by the original Abacus classifier to be compliant with them, we perform some experiments to test its accuracy with this aggregated data. Sec. 6.2 instead deals with the issue of flow-sampling. Such phenomenon arises when the classification engine is moved from the edge of the network towards the core, where, because of routing, only a portion of the traffic directed to a host is observed by the classifier. By using both a simulated and real-life scenarios based on routing tables from actual core routers, we tested our classifier in settings where only an incomplete knowledge of the traffic directed to the target host is available.

## 6.1 Behavioral classification with NetFlow

For many network management tasks, the use of aggregated measurements has represented an efficient way to cope with the huge amount of traffic. For instance, in recent years, we have assisted to the confirmation of NetFlow as the main solution for flow-level measurements. Widely implemented in routers and recently standardized in a IETF draft [57], NetFlow reports aggregated information on network traffic in the form of *flow-records*. It is a common belief that the amount of information carried by flow-records is not enough to support an accurate identification of network applications. Yet, given the widespread adoption and availability of NetFlow we argue that a classification engine based on flow-records would be highly appreciated by ISPs.

Actually, the research community has also investigated the possible utilizations of NetFlow data for different network managing tasks. The most natural applications of NetFlow records are clearly represented by *accounting* [151, 168] and network traffic *reporting and monitoring* [66]. Other works [113, 150] presents effective techniques to perform *anomaly detection* based on Net-

Flow data. Only a few works [43, 97] closer to ours have recently proposed *traffic classification* techniques based on NetFlow records. Both these works belong to the family of statistical based classifier, while in we rather explore the used of such data for behavioral traffic classification. Moreover, as Abacus distinguished itself for being a fine-grained classifier, we would like to keep this property even if using less informative, aggregated data.

In the following we will first describe the NetFlow architecture, to better understand what changes are required for Abacus to deal with flow-records. Then we test our classifier against such data.

### 6.1.1 Netflow data

NetFlow [56] represents the de facto standard for flow level measurements in the networking operational community. Originally designed by Cisco as a cache to improve IP flow lookups in routers, it soon revealed a very useful tool network traffic monitoring and reporting. NetFlow main feature relies in the level of information it provides – more compact than packet level traces, but still more expressive than coarse-grained counters of SNMP. After several subsequent versions, with v5 being the most commonly used, the most recent NetFlow v9 has evolved in a IETF standards, IPFIX [57], already implemented by most network equipment vendors.

A NetFlow probe tracks *flows*, i.e. unidirectional sequences of packets exchanged by two endpoints. First, it extracts from each packet a *key* composed of specific header fields (for v5 it is the classical 5-tuple: IP source and destination addresses, transport-layer source and destination ports, IP protocol). This key identifies a record in memory, where the probe stores, besides the key itself, a number of *attributes*, like cumulative packets and bytes counters, flow starting and finishing timestamps, IP type of service, TCP flags, MPLS label, physical input and output interface indexes, only to cite the most important ones. Even if newer versions like NetFlow v9 and IPFIX introduce the possibility to specify custom flow keys and fields templates we employ the flow definition of v5.

Whenever a flow expires, the router transmits a UDP packet containing the related record to a NetFlow collector, which elaborates and eventually stores this information. Different reasons can cause a flow expiration:

- A packets explicitly terminates the flow (e.g, a TCP FIN packet).
- The flow has been *inactive* for a time greater than the `inactive_timeout`.
- The flow has been *active* for a time greater than the `active_timeout`.
- The flow cache is full and some space needs to be freed for new flows.

Default values for `inactive_timeout` and `active_timeout` are respectively 15 seconds and 30 minutes. In Sec. 6.1.4 we will see that the values of these timeouts have an important impact on classification performance,

### 6.1.2 Using flow-records for classification

To avoid repeating ourselves, we refer the reader to Chap. 4 for a complete description of the Abacus classifier internal workings. We just recall that the Abacus signatures (i.e., the features fed to the supervised classification engine) are basically the concatenation of two probability mass functions: considering the peers contacted by the target classification host $X$ in a small time-window, first we calculate the distribution of such peers according to the number of packets sent

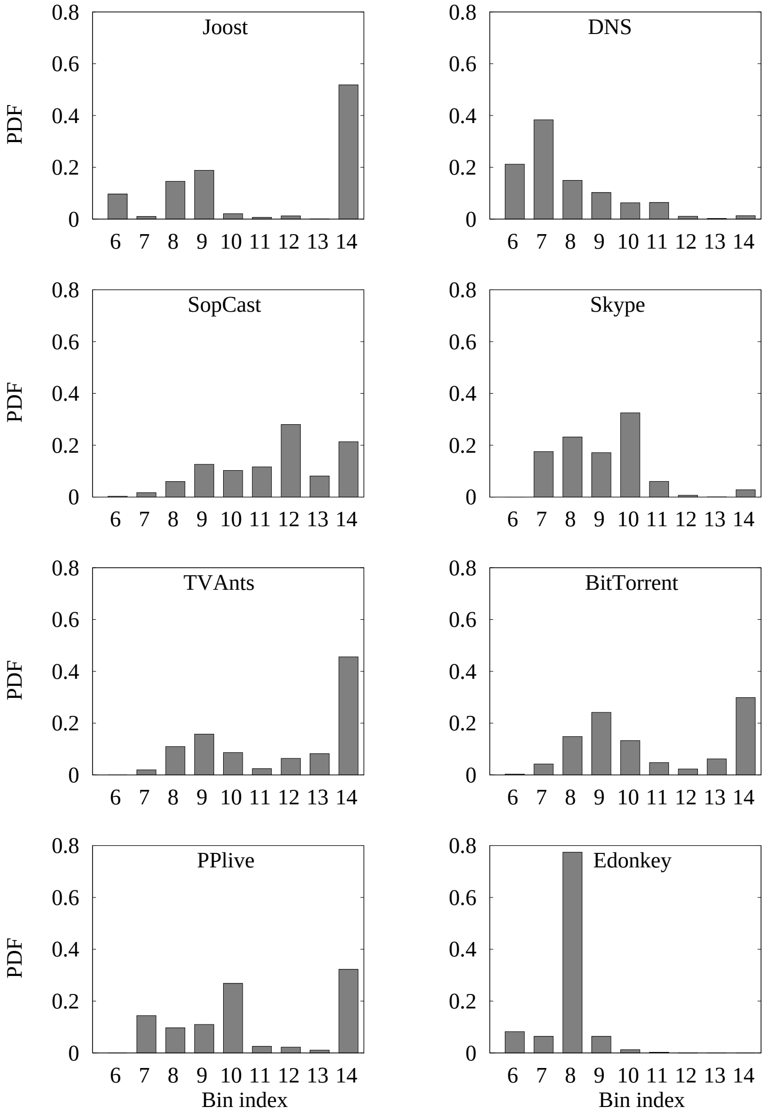


Figure 6.1: Mean byte-wise signature for the considered applications.

to $X$; then we calculate the distribution of peers according to the bytes sent to $X$. We use a logarithmic binning with base 2 as support, which captures the most important information while also reducing the size of the signatures.

It can be seen that standard NetFlow records contain all the information needed by Abacus, which is simply the number of packets and bytes received by the target host from its peers. Nevertheless, some modifications were needed to make the Abacus classifier work smoothly with NetFlow data. First, in our previous work we used a duration of 5 s for the endpoint observation window, which is incompatible with the time granularities of NetFlow: hence, we now consider values of $\Delta T$ in the time-scale of minutes. A second difference derives from the fact that flow-records exports are not synchronous. In other words we cannot ask NetFlow to terminate and emit all flow-records at a give time (e.g. at the end of $\Delta T$). As a result, we may end with some records spanning over two windows. In this case we simply divide the flow in two parts, by prorating the number of packets and bytes (i.e., we split the flow-record in correspondence to the end of the current $\Delta T$ and divide the packets and bytes between the two segments proportionally to their

Table 6.1: Summary of the dataset

| Category | Application | Packets | Bytes |
|---|---|---|---|
| P2P TV | PPlive | 7,3 M | 1,13 G |
| | Sopcat | 3,2 M | 0,45 G |
| | TVAnts | 2,4 M | 2,56 G |
| | Joost | 3,4 M | 2,14 G |
| P2P File-sharing | eDonkey | 22,4 M | 6,93 G |
| | BitTorrent | 1,4 M | 0,74 G |
| P2P VoIP | Skype | 6,1 M | 2,91 G |
| Naming | DNS | 1,5 M | 103 M |
| Remaining UDP | Other | 10 M | 10.9 G |

duration), as already done by [168].

Finally we actually extended the target applications to include also other kinds of P2P applications. We will see later when we describe the dataset, that we train the machine also with signatures for VoIP and filesharing P2P applications, and even with signatures for DNS (which, despite implementing a client-server applications, from a higher perspective behaves much like a P2P service). In this way we want to see whether Abacus signatures can also discriminate among these services. As a consequence of such extension, we also drop the rejection criterion used to recognize “unknown” traffic (i.e., traffic not pertaining to the applications included in the training set): inspired by the Kiss classifier, we introduce an additional class in the training set, containing signatures derived from all traffic different from the target applications (see also Chap. 5).

Before presenting the classification results, in Fig. 6.1 a pictorial representation of the average byte-wise signature of all the applications considered in the experiments, to give an intuitive example of Abacus signature discriminative power. We represent signatures as histograms, where each bar corresponds to the value of the component of the distribution identified by the index on the x-axis. Notice that each application generates a different and characteristic pattern. For instance, DNS adopts short exchanges of data for the majority of communications, which contribute to larger values of lower bins (e.g. most DNS queries fall into bin number 7, corresponding to single packets having size in the 128–255 Bytes range). All P2P-TV applications (top row) together with BitTorrent instead show a considerable percentage of peers falling in the last bin (corresponding to hosts which are contributing with most of the video or data stream), but still differentiate among themselves thanks to lower bins.

This example confirms that, despite using simple counters, furthermore of a single direction, Abacus signatures capture distinctive properties of the observed applications. Concerning P2P applications, a number of design choices directly reflect in the signature: for instance, different peer discovery techniques (e.g., single-packet probe versus handshakes) clearly make peers fall into different bins (e.g., the first or second respectively). In the same way, even though several applications may adopt similar content-diffusion techniques (e.g., mesh-pull diffusion for BitTorrent, SopCast, PPLive) the use of specific chunk-sizes, or even packet sizes, still makes them easily recognizable (e.g., by making a specific bin more likely than others). Concerning client-server traffic, we point out that it still makes sense to look at a single direction (namely, the incoming direction), as we expect the length of requests to be tied to the protocol (e.g., very short DNS queries, versus medium length HTTP request, versus rather long SMTP message transmissions).

Table 6.2: Confusion matrix: Classification accuracy of Signatures (S) and Bytes (B)

| | PPLive | | TVAnts | | SopCast | | Joost | | eDonkey | | BitTorrent | | Skype | | DNS | | Other | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | S | B | S | B | S | B | S | B | S | B | S | B | S | B | S | B | S | B |
| PPLive | **63.6** | **96.0** | 1.0 | 3.2 | 0.7 | 0.3 | 0.1 | - | - | - | 0.1 | 0.4 | 2.9 | - | 9.4 | - | 22.3 | - |
| TVAnts | 3.1 | 6.8 | **54.4** | **92.9** | 1.0 | 0.3 | 0.2 | - | - | - | 0.2 | - | 7.4 | - | 9.5 | - | 24.3 | - |
| SopCast | 0.7 | 0.2 | 0.4 | 0.4 | **49.7** | **99.4** | - | - | 0.1 | - | 0.3 | - | 4.8 | - | 15.9 | - | 28.1 | - |
| Joost | 0.2 | - | - | - | - | - | **53.2** | **99.9** | 0.3 | - | 0.2 | - | 4.5 | - | 19.1 | - | 22.5 | - |
| eDonkey | - | - | - | - | - | - | - | - | **94.4** | **98.9** | - | - | - | - | 0.7 | 0.2 | 4.8 | 0.9 |
| BitTorrent | 0.6 | - | 0.5 | 0.1 | 0.8 | 0.8 | 0.3 | 1.9 | - | - | **12.5** | **89.1** | 5.2 | 1.7 | 61.3 | 5.8 | 18.8 | 0.6 |
| Skype | - | - | - | - | 0.1 | 0.3 | - | - | - | - | 0.2 | 0.4 | **86.1** | **90.5** | 5.8 | 2.5 | 7.8 | 6.4 |
| DNS | 0.1 | - | - | - | 0.1 | 0.3 | - | 0.2 | 0.3 | 0.9 | - | 0.5 | 6.5 | 3.9 | **63.9** | **91.2** | 29.1 | 2.9 |
| Other | 0.1 | - | - | - | - | - | - | - | 0.4 | 0.1 | - | 0.1 | 3.5 | - | 8.3 | - | **87.6** | **99.8** |

### 6.1.3 Dataset and Methodology

As done in previous chapter we use a subset of our overall dataset, described in Sec. 2.4. For P2P-TV application we use the traces from the European Testbed set up in the context of the NAPA-WINE project, while for all other protocols we used a passively collected traces from an Italian operator and later processed with a traditional DPI classifier to extract the packets pertaining to our target protocols as well as the background traffic to assess the capacity of the classifier of dealing with “unknown” traffic. In this case, as done in Chap. 4, We concentrate on UDP traffic, which is not only the preferred transport-layer protocol of P2P live-streaming application and VoIP, but has also recently become the choice of the most popular file-sharing application, namely BitTorrent.

We implemented a custom tool similar to [66], which converts traffic traces into NetFlow-records. Flow-records related to the target hosts are then combined together and eventually split to calculate the Abacus signatures. We randomly sample 10% of the signatures of each protocol that we use to train the SVM model. The trained model is then applied to the remaining signatures, used as validation set. This process is repeated 10 times, varying the train and validation set each time.

### 6.1.4 Classification results

In Tab. 6.2, we show the classification results obtained by setting the `active_timeout` as well as $\Delta T$ to 120 s, while we leave the `idle_timeout` to its default value of 15 s (we will discuss this parameter choice later). The table adopts the classical confusion matrix representation and, as usual, classification performance is expressed both as percentage of signatures and as percentage of bytes. Values in bold, i.e. the recall, on the diagonal represent the percentage of correctly classified traffic.

For the sake of simplicity, let us consider different families of applications separately. We first observe the performance related to P2P-TV services, which were the original target of the Abacus classifier. Byte accuracy is extremely high for all four applications, always greater than 90%, with Joost and SopCast exceeding the 99%. Instead, the percentage of correctly classified signatures is lower (not even approaching the 65%), with a large fraction of classifications falling in either the DNS or the “other” class. From this, we can conclude that heavy signatures (i.e., carrying more bytes) are more robust to the classification, which is a positive finding as ISPs are interested in classifying the bulk of the traffic. This is also somewhat expected, as signatures carrying less traffic (e.g., few flows in the unit of time $\Delta T$) are also statistically less significant.

Byte accuracy remains high also for the two P2P file-sharing applications, with eDonkey showing also very high values at a signature level. BitTorrent signatures are instead harder to classify in general, except those carrying the greatest portion of traffic (bytes accuracy approaches 90%).

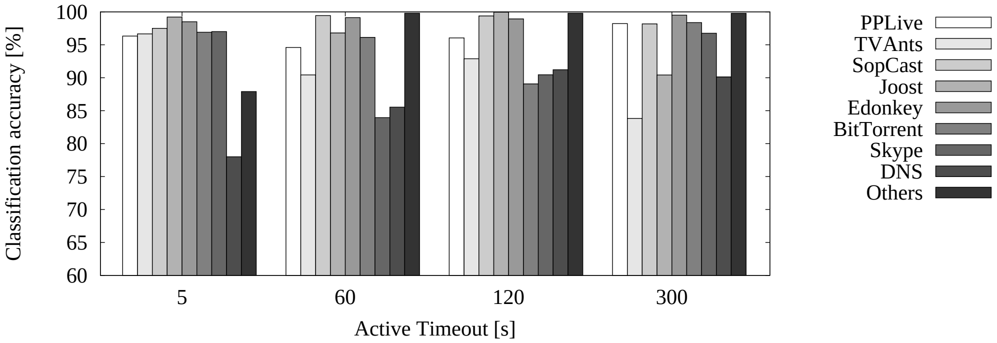


Figure 6.2: Byte accuracy for different values of active timeout.

Again, it seems that the classifiers correctly identifies BitTorrent traffic when the application is actively downloading from other peers, while it has some problems under different conditions (e.g., when just discovering new peers or performing signaling). Moreover, we point out that due to the recent evolution of its protocol, we have collected just a limited dataset, so the poor performance may also be caused by the incomplete representation of the protocol in the training set. Interestingly, notice also that BitTorrent, which adopts swarming algorithms similar to P2P-TV ones, is sometimes misclassified as P2P-TV, whereas eDonkey is rather confused with DNS.

Similar considerations apply also to Skype and DNS, both performing better when considering bytes instead of signatures. DNS, the only non-P2P applications considered in this work, is the protocol which shows the greatest portion of signatures labeled as “other”. This is likely due to the presence of other client-server applications in the background class, which, having a behavior similar to DNS, cause the misclassification.

In the last row of the table we instead evaluate how the classifier deals with traffic other than the target applications. Again, the signatures corresponding to the majority of bytes are correctly assigned to the “other” class, showing the effectiveness of this strategy for handling any kind of traffic (a limited percentage of signatures is still misclassified as DNS, which further confirms our previous considerations on the similarities between these classes).

Besides the results reported in Tab. 6.2, we conducted a series of tests to investigate the effect of different settings for the NetFlow timeouts on the classification performance, that we only briefly report here for lack of space. Fig. 6.2 depicts the byte accuracy of the different applications for increasing values of the `active_timeout`. As a reference, the leftmost set reports the performance from Chap. 4 gathered for $\Delta T = 5\,$s (not applicable for real NetFlow). Coherently with our previous work, this small interval provides the best performance for P2P applications, but deals poorly with DNS and the “other” classes. From the picture, it can be gathered that $\Delta T = 120\,$s represents a good compromise between smaller intervals, which enhance P2P-TV classification accuracy, and larger intervals, which perform better for the other applications.

## 6.2 Behavioral classification in the core: the issue of flow sampling

By flow sampling we mean that depending on the location of the vantage point where the classification decisions are taken, possibly not all the traffic generated by an endpoint can be observed. For instance, when the classification engine is moved from the access (e.g., DSLAM) deeper into the aggregation network (e.g., at the first or second IP router), only part of an endpoint traffic is likely available in this new position, for, due to routing issues, some packets may follow a different path and be handled by other routers. In such a case, even though observing only a *subset* of the whole endpoint traffic, classification might still be achievable and accurate, or, on the contrary, it might also drastically fail.

We assume that an Abacus classifier, trained with unsampled traffic (i.e., to be used at the edge of the network), is then employed to classify a subset of the total traffic received by an endpoint (i.e., it is deployed in the network core). In other words there will be a mismatch between training and validation. Notice, in fact, that an Abacus classifier in the core has no way of knowing what portion of the endpoint traffic is transiting on the path on which it is deployed, hence the mismatch cannot be avoided when the vantage points moves from the edge.

In our tests, we apply two different policies to sample flows to be included in the subset: a realistic one, that takes into account the real router forwarding tables, and an idealized one, where flows are sampled at random depending on the originating subnet. This two-fold approach ensures realism of the results on the one hand, while, on the other hand, it also allows to perform controlled experiments to precisely gauge the impact of sampling.

While we defer a detailed overview of the related work on packet sampling and its applications to the next chapter, we anticipate here that the effect of sampling on the performance of traffic classification techniques has rarely been considered in the literature [33, 43, 73, 77, 97, 138]. Moreover, while most previous works consider the impact of *packet sampling* on classification accuracy, here we take an orthogonal perspective considering the issue of *flow sampling*, which, to the best of our knowledge, has not been dealt with so far.

In the remainder of this section we present the results of our experimental campaign. We use the same dataset earlier presented in , but we only focus on the six pure P2P applications, namely PPLive, SopCast, TVAnts, eDonkey, BitTorrent and Skype. As our purpose is to quantify the classification accuracy when only a portion of traffic is observed, we first (i) study how application traffic spreads over the IP address space, and (ii) by using RIB data from routing tables of a real core Internet router, how the traffic might be split over different paths in the network. Afterwards, we assess the change of the Abacus signatures due to traffic sub-sampling, and how these changes affect the overall classification performance.

### 6.2.1 Spatial distribution of application traffic

As a preliminary step we analyze how application traffic is distributed in the IP address space, i.e. whether it spreads over the whole range of IP addresses, so that no subnet is more important than the other, or whether it is concentrated in a few subnets whose observation may be fundamental for the classification.

To gather a quick representation of the traffic distributions, we partition the IP address space in 256 subnets with a netmask/8 (i.e., we consider just the first byte of the IP address) and accumulate the traffic received by each subnet in terms of bytes. The resulting histograms for each application are reported in Fig. 6.3, where on the x-axis we have the value of the first byte of the IP address (which obviously ranges from 0 to 255) and on the y-axis we have the number of bytes received by such addresses. The gray-shaded areas refer to the logarithmic scale reported on the left axis, so that we can see all networks from which even a small amount of (signaling) traffic is received, while the solid bars are reported in linear scale, to highlight the subnets with which the bulk of (data) traffic is exchanged.

Looking at the shaded areas, we see that applications known for their aggressive peer discovery behavior, like PPLive or Skype, exchange traffic with large portions of the IP address space. Other applications such as TVAnts or SopCast (top two plots) present instead large white gaps, meaning that some address ranges are not part of the P2P overlay. On the other hand, the few dark bars

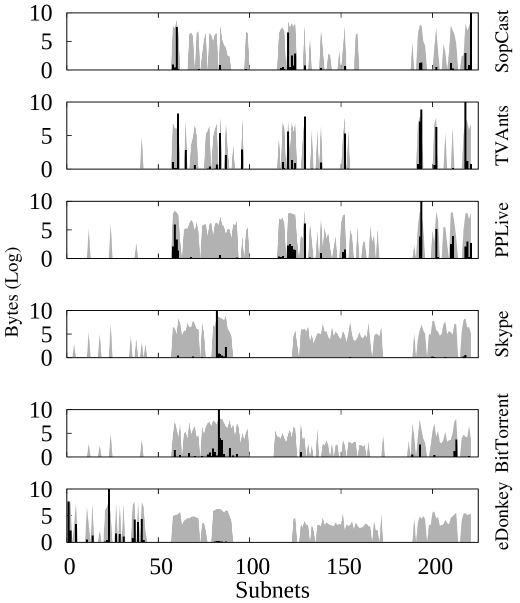


Figure 6.3: Distribution of bytes across the IP address space, linear (solid) and logarithmic (shaded area) scales.

tell us that the bulk of traffic comes mostly from a small number of subnets, around a dozen for P2P-TV applications (top three plots) or even less for the remaining graphs.

### 6.2.2 Real-life IP routing analysis

While the previous analysis yields a coarse-grain view of traffic distribution across the IP space, it oversimplifies the picture with respect to real IP routing. In fact, routes found in routing tables of IP core routers and announced in BGP updates are much more fine-grained. This means that traffic can be split in far smaller subsets when it is actually forwarded to its destination.

For this reason we used public available real-life routing information [12] to precisely emulate how our target application traffic is forwarded along different path by a router in the Internet core. The RIS project [12] collects and makes public available to researcher periodic BGP RIB dumps of several core routers, which basically contain all the prefix announced by other peer routers. We downloaded the RIB of a core routers located in Amsterdam with 84 peering routers, for April 4th 2008, i.e., the very day in which the active traces of our dataset were captured.

In our analysis we assume that the IP router has to forward the dataset traffic from its internal network to the outside peering routers. Although we do not know the internal BGP rules of the router, we can however estimate the routing table by means of the standard criteria used by BGP routers: if the same prefix is announced by two peer routers, we choose as next hop for such destination the one announcing the shortest AS path, or, in case of paths of the same length, the one with the lowest peer ID. Fig. 6.4 shows the percentage of the overall traffic which is received by each peering router in terms of bytes and peers, omitting the peers that receive negligible amount of traffic. It can be seen that most of the traffic is carried by a few outgoing links: in particular, in the plot we can see that the Amsterdam router would forward roughly 70% of traffic to peer 0.

It looks like even in real-life scenarios, when moving our vantage point for classification towards the network core, it is still possible to observe significant portions of the total endpoint traffic on specific paths. In turn, we expect this phenomenon to allow accurate classification, at

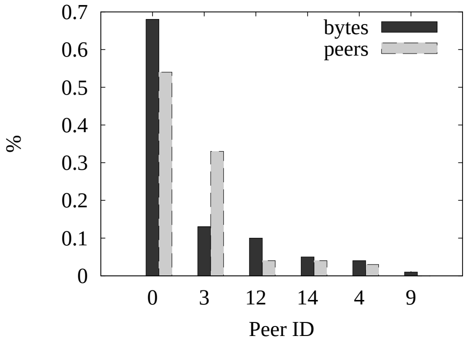


Figure 6.4: Percentage of bytes seen by the different BGP peers.

least for some paths. Besides, an important observation is that we also expect the *relevance* of the classification to be directly proportional to the amount of traffic observed: in other words, on links that carry only a negligible portion of the endpoint traffic, providers are unlikely to be interested in classifying such data, while classification accuracy is required on links carrying the bulk of traffic.

### 6.2.3 Impact of flow-sampling on Abacus signatures

In this section we assess the distortion that flow-sampling induces in the Abacus signatures. Recall that signatures basically represent the breakdown of peers according to the number of packets and bytes sent to the target endpoint: we evaluate if, and how much, the shape of such distributions changes when only a portion of the traffic is available to the classifier.

To have more control on the sampling parameters, we use an idealistic scenario rather then the real routing table. We randomly sample the /8 subnets used to build Fig. 6.3 with a decreasing probability $k \in \{1, 1/2, 1/4, 1/8\}$: only flows pertaining to the sampled subnets are considered when building the Abacus signature. Fig. 6.5 shows the average Abacus signature (over all the time-windows) for all the applications, with growing values of sampling rate which directly correspond to fading colored bars. At first glance the shape of the distributions seems to be only slightly affected by sampling, especially when $k < 1/8$. Changes start to be more evident for $k = 1/8$ (in particular BitTorrent and Skype shapes are significantly altered by sampling) and signatures appear more similar *across* applications.

A more quantitative analysis of distortion is provided in Fig. 6.6-(a). Here we use the Bhattacharyya distance $BD(p, q)$, which we already employed for the rejection criterion of Abacus in Chap. 4. It is a measure of similarity between probability distributions $p$ and $q$. The BD distance takes values in the range $[0, 1]$, with larger values corresponding to bigger differences. The graph relates to packet-wise signatures only, and report two different types of curves. First, for each application $X$ we show the intra-application distance $BD(X_1, X_k)$, i.e. the distance between the unsampled and sampled signatures. The distance increases with the sampling rate, but values keep lower than 0.3, consistently with the qualitative analysis of Fig. 6.5 showing only small changes in the signatures. The top curve shows instead the mean inter-application distance $E[BC(X_k, Y_k)], \forall X, Y$, i.e., how much the applications differ between each other, thus being an index of the separability of the classes. The fact that the value keeps quite stable, and much larger than $BD(X_1, X_k)$, suggests that differences among the applications are well preserved even in harsh sampling conditions.

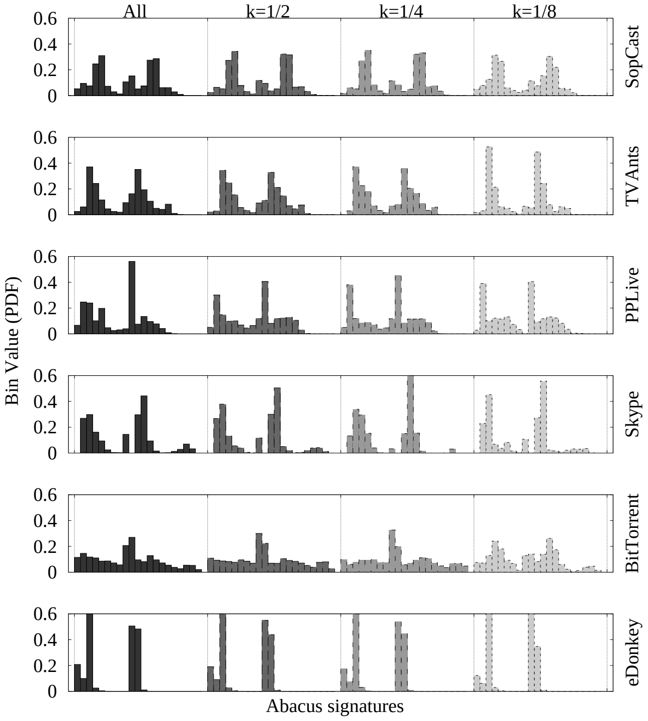


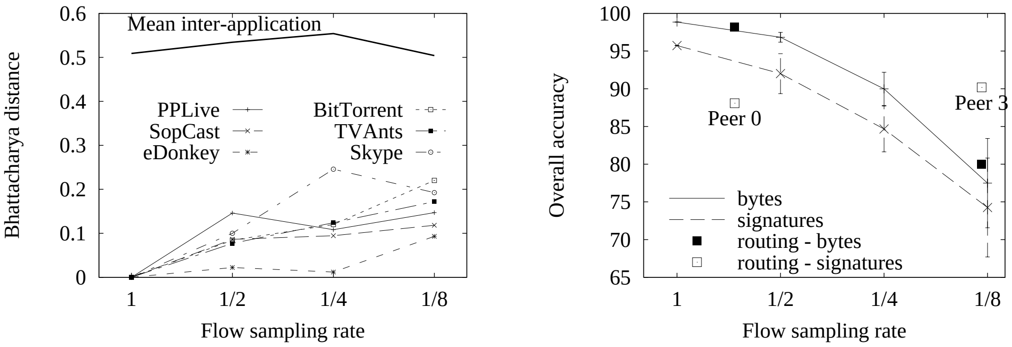
Figure 6.5: Mean signatures of each application for increasing values of flow sampling.

### 6.2.4 Impact of flow-sampling on classification accuracy

We finally apply our classifier to the sampled traffic. First we consider the idealistic case of randomly sampled subnets that we have introduced in the former section. More in details, we first train our classifier with 10% of the *unsampled* signatures. The trained classifier is then applied to the test set, which is made up of signatures derived from *sampled* traffic. To gather robust results, for each flow-sampling rate we generate 5 different test-sets, randomly selecting different subnets to be sampled, and 10 different training set from the unsampled traffic (experiments explore the full cross product of the training and validation sets).

Fig. 6.6-(b) reports the average classification accuracy computed over all experiments (standard deviation is reported as error bars) as a function of the sampling rate. Accuracy is expressed both in terms of signatures and bytes, which is usually the most significant metric from an network operator perspective. Accuracy degrades gracefully with the increasing sampling rate: indeed, notice that when as low as $k = 1/4$-th of the traffic is sampled, the accuracy is reduced of about 10% (20% at $k = 1/8$). As sampling rate increases, the standard deviation increases as well, as different subnet sets may yield radically opposed results. In fact, in the lucky cases where an important subnet (i.e., in which a large number of either peers or bytes fall) is selected, then the classifier is able to correctly classify the traffic without difficulty; otherwise, classification fails.

Fig. 6.6-(b) also reports results for the real-world scenario with squared points for the first two BGP peering routers, receiving respectively about the 70% and 10% of the traffic share (cfr. Fig. 6.4). The points have as x-coordinate the value of sampling probability that would sample approximately the same amount of traffic observed by the router. Interestingly, while the byte accuracy is in line with our reference scenario, the signature accuracy behavior is odd (i.e., Peer 3 exceeding Peer 0), which is rooted in the different mixture of traffic observed by the two routers. Namely, Peers 0 handles the bulk of the traffic and therefore the most of the data-transfer: hence,



(a) (b)

Figure 6.6: (a)Bhattacharyya distance: inter-application $E[BD(X_k, Y_k)]$ difference at sampling $k$ and intra-application distance due to sampling $BD(X_1, X_k)$. (b) Overall accuracy for different training sets in function of the amount of traffic observed.

correctly classifying a few heavy signatures has a great impact on the byte-accuracy. Conversely, Peer 3 handles 30% of the peers that however generate less than 10% of the traffic share (cfr. Fig. 6.4): hence, correct signature classification do not necessarily translate into high byte-wise accuracy.

## 6.3 Summary

In this chapter we have tested the behavioral classifier proposed in the previous chapters in more challenging settings. First, motivated by the widespread of NetFlow as a monitoring tool, we evaluated the performance achieved by Abacus when exploiting only this kind of aggregated data and characterized by a coarser time-granularities with respect to our previous experiments. We tested our methodology on a large set of traces, ranging from P2P applications to traditional client-server services. Results show that the proposed technique, albeit not infallible in term of individual signatures, classifies all the target applications with high byte accuracy, and also correctly handles “unknown” traffic.

Second, we studied the impact of flow-sampling on the accuracy of Abacus. This important issue arises when the classifier is moved from the network edge toward the core, so that only a portion of P2P endpoints traffic can be observed. Our study shows that the normalized Abacus signatures degrade slowly under sampling: as a random selection of subnets (hence, of peers) is unbiased with respect to the application behavior, this makes Abacus classification robust under this scenario – even in case of real routers as considered in this work. More precisely, our results show an accuracy drop of about 20% when only about the 10% of the subnets is sampled.

# Chapter 7

# Impact of Sampling on traffic characterization and classification

As a side effect of the increasing transmission speed of nowadays networks, network operators must deal with ever growing traffic and the consequent huge amount of measurements, extremely challenging to collect, store and process. Therefore, *packet sampling* has become mandatory for effective passive network measurements, especially in the core of the network, to reduce the amount of data to a manageable size. Naturally such reduction comes at the cost of less accurate data and several studies have focused on the impact of different sampling policies on traffic measurement [28, 49, 55, 67, 69, 109, 133, 146] or on the performance of various networking activities related to measurement, such as monitoring, SLA compliance, anomaly detection and traffic classification [42, 43, 88, 97, 120, 137, 188]. However, in spite of the importance of traffic classification we have seen in previous chapters, the impact of sampling on traffic classification is a rather little investigated subject.

In this chapter, whose results were in part published in [142], we deeply investigate this issue. First, we measure the effect of different sampling policies and rates on a large set of traffic properties, without being tied to any particular application. Second, we focus on traffic classification under sampled data (hence, traffic properties become "features", in machine learning terms, upon which classification decisions are taken): in this context, we evaluate the extent of the degradation of the information content conveyed by the features, besides investigating the achievable classification accuracy.

In order to gather general results, we employed an extremely heterogeneous dataset, composed by four different traces, from both academic and commercial networks, representative of different access technologies, network setting, user population and years. When possible we used publicly available dataset to promote cross-comparison in the scientific community. This dataset was processed by a popular flow-level analyzer, `tstat` [16], which outputs not only detailed per-flow features at network and transport layer, but also aggregated distributions of feature. For this work, we enhanced the tool with the possibility of applying several sampling policies (namely, *systematic*, *uniform*, *stratified* and *biased*) with arbitrary sampling rates.

We characterize the distortion introduced by sampling as the distance between the distribution of properties for sampled and unsampled traffic, by means of two commonly used statistical metrics, namely the *Hellinger Distance* and the *Fleiss Chi-Square*. As for the impact of the sampling on traffic classification, we rather use a biased sampling policy which tries to capture the most important pieces of information. For this task, we first asses the usefulness of features using a information theoretic metric, namely the *Information Gain*, and afterwards we evaluate the classification accuracy through C4.5, a widely know supervised classification algorithm. We study the impact of different sampling rates, feature groups, datasets and training policies.

The remainder of this chapter is organized as follows. First an overview of the related work is found in Sec. 7.1. Then, Sec. 7.2 presents the data used for the experimental part: first, the packet level traces and, second, the features produced by our monitoring tool. We then describe the methodology used to process this data in Sec. 7.3: in particular the sampling policies, the statistical metrics and the classification algorithm. Results are split in three sections: Sec. 7.4 deals with distortion of the *aggregate features*, Sec. 7.5 with the distortion of *per-flow features*, and finally Sec. 7.6 presents the *traffic classification* results. We summarize our findings in Sec. 7.7.

## 7.1 Related work

Packet sampling is not a novel technique [68]. Yet, given its increasing importance, it has recently received a lot of attention from the research community. In the following we overview the most important pieces of work on this topic, both on sampling itself and on its effect on traffic classification, in order to better highlight our contribution. Yet, this is far from being a complete survey, for which we rather refer the reader to [68].

First of all, researchers have categorized packet sampling methods in a few classes, starting from [28], until they have finally converged to a common framework standardized as an IETF RFC [189]. Summarizing, a first distinction can be made according to the selection scheme, which can be *deterministic*, *random* or *content-based*. Second, we can differentiate sampling techniques according to the selection trigger, based on the amount of *time* or number of *packets* between two different sampling events. As far as the selection scheme is concerned, researchers have demonstrated that the statistic properties of random sampling, especially in its stratified declination, make this technique particularly robust to evasion and attacks [55, 68, 139]. On the other hand, recent studies showed that statistical multiplexing of traffic may have the same effect of a random selection [49], especially when considering estimation of traffic volumes. There is much more of an agreement, instead, about the most effective selection trigger: [55] showed that time-based triggers are less robust than packets-based ones, because they suffer from the bursty nature of network traffic. A few works have proposed more sophisticated sampling techniques which help in estimating specific traffic features, for instance trajectory sampling for spatial properties [70] or sketches for flow-size [109]. Other works have proposed to make the sampling rate *adaptive* [53, 67, 88] for instance to the traffic load, to reduce the estimation error of some traffic metrics.

Besides investigating the properties of sampling itself and its impact on mostly traffic volumes measurements [49, 69, 109, 133, 146], researchers have also studied the possible applications of sampled data for various network administration tasks, such as network management [88], SLA verification [188], anomaly detection [42, 120, 137] and, lately, on traffic classification [33, 43, 73, 77, 97, 138, 186]. It must be said that in this kind of evaluation is not easy to distinguish between the actual impact of sampling from the intrinsic performance issues of the application itself. In this sense, the first part of this chapter, which studies the effect of sampling on its own, considering different policies, rates as well as a wide range of traffic features well beyond simple volume measurements, is particularly helpful in shedding light on this issue.

As already mentioned, to date there are not many papers jointly considering both traffic classification and sampling [33, 43, 73, 77, 97, 138, 186], and, moreover, the majority among them only treats sampling as a minor issue. For instance, [73] analyzes how sampling methodology influences the selection of both elephant and mice flows in the *training* data set, aggravating the traditional class imbalance problem; the same issue is mentioned as particular interesting, but only

Table 7.1: Summary of dataset used in this work.

| Trace | Auckland | ISP | Campus | UniBS |
|---|---|---|---|---|
| Year | 2001 | 2006 | 2008 | 2009 |
| Packets | 291M | 44M | 17M | 26M |
| Flows | 11M | 219K | 422K | 34K |
| Packets/flow | 26.2 | 202 | 40.8 | 764 |
| IPs | 410K | 61K | 81K | 6.59K |
| Available at | [2] | – | – | [18] |
| Ground truth | Port-based | – | DPI [121] | `gt` [82] |

as a future work in [186]. Other papers only try to predict what sampling might imply for the classifiers they propose: authors of [33], whose technique is based on the size and direction of the very first packets of a flow, sustain that their classifier would badly suffer packet sampling, whereas being robust to flow-sampling; on the opposite side, authors of [77] argue that accuracy of stochastic packet inspection should be not influenced by sampling altogether (provided that enough packets are sampled to get statistically relevant signatures).

The impact of packet sampling is experimentally addressed in [43, 97, 138]. In more details, [138] investigates the sampling effect on Reduced Error Pruning Tree (REPTree) classifiers, and limitedly reports a single case study for $p = 1/3$ (asserting that classification accuracy lowers of 10-20%, depending on the client-to-server or server-to-client traffic direction). Instead [97] investigates the sampling effect on a lightweight traffic classification approach (using Naïve Bayes on NetFlow records, and varying the sampling rate) finding that packet sampling does not worsen the results (rather, accuracy may increase under heavy sampling) and suggesting this may be due to an artifact of packet sampling (though a more detailed analysis is missing). Finally, the only work that shares its main focus with ours is [43], which studies the accuracy of statistical traffic classification based on NetFlow sampled data. From extensive experiments and a formal probabilistic analysis, authors of [43] draw a conclusion similar to ours, showing that the use of sampled data both in the training and testing phase greatly improves the otherwise degraded accuracy obtained with sampled NetFlow. Nevertheless they only consider the limited set of features available in standard NetFlow v5.

## 7.2 Dataset and Features

Given the experimental nature of this analysis, it is extremely important to give as many details as possible on the data employed. In this section we first describe the details of the packet-level traces. Second, we present the tool used to analyze such traffic [16], which applies sampling to the traces and extracts, as well, a wealth of features able to characterize several properties of the traffic at different layers of the networking stack.

### 7.2.1 Dataset

In order to gather results that are representative of a wide range of network environments and epochs, we use several traces, whose main characteristics are summarized in Tab. 7.1. We used the whole dataset described in Sec. 2.4, composed by the traces from an Italian operator (*ISP*), our university network (*Campus*), the University of brescia (*UniBS*) and the University of Auckland (*Auckland*).

Table 7.2: Subset of the dataset used for classification, and application breakdown.

| | UniBS | | Campus | | Auckland | |
|---|---|---|---|---|---|---|
| | Flow | Byte | Flow | Byte | Flow | Byte |
| Protocol | % | % | % | % | % | % |
| HTTP | 49.3 | 5.6 | 41.8 | 62.7 | 34.8 | 25.3 |
| HTTPS | 1.5 | 1.2 | 41.8 | 30.6 | 34.8 | 23.4 |
| FTP | - | - | 4.8 | 0.03 | - | - |
| IMAPS | 3.7 | 0.1 | 0.2 | 3.9 | 0.6 | 0.9 |
| POP3 | 1 | 0.01 | - | - | 5.6 | 2.8 |
| SMTP | - | - | - | - | 23.9 | 47.5 |
| Skype | 1 | 0.7 | 11.1 | 2.6 | - | - |
| eDonkey | 40.1 | 87.2 | - | - | - | - |
| BitTorrent | 3.3 | 5.0 | - | - | - | - |

Extremely heterogeneous network scenarios are taken into account: we consider both commercial and academic environments, as well as different access technologies and security settings. Moreover traces are collected in a time which spans nearly a decade, the oldest dating 2001 (Auckland), whereas the newest being from 2009 (UniBS). Such a diverse dataset is fundamental to gather statistically meaningful results. Diversity is even more important for our classification purposes. It is known, in fact, that the accuracy of a classification algorithm is heavily impacted by several factors like different traffic mixes, different network setups, different times of the day and so on. For this reason, all these aspects must be taken into account and possibly included in the dataset, so as to gather a reliable evaluation of the classifier performance. Furthermore, sampling makes this issue more critical, as it possibly discards most of the traffic volume when aggressive rates are applied.

As usual, when testing classification accuracy, we put extra care in the definition of the ground-truth. For this reason, we decided to extensively rely on the UniBS dataset, which is public and has a very reliable ground-truth associated thanks to the `gt` tool [82] developed at University of Brescia and Politecnico di Torino. For the remaining datasets, instead, we need to build our own ground-truth: for Campus we used the DPI classifier described in [121]; for Auckland we employed a simple port-based classification scheme, which was very reliable in 2001, when applications still abide by the standards IANA well-known port allocation; we neglect the ISP trace in the classification part, to avoid using traffic with uncertain application labels.

Tab. 7.2 summarizes the composition of the traces according to our pre-labeling. As expected most of the traffic is carried over HTTP, which together with IMAPS, is the only protocol common to all traces. The mix of protocols also reflects the date of the traces: in Auckland we found exclusively traditional client-server applications, whereas more recent traces include also P2P applications, both file-sharing (eDonkey and BitTorrent) and VoIP (Skype).

### 7.2.2 Features

In our experiments, packet-level traces were processed with `tstat` [16], which logs several traffic features as output of its analysis. We actually enhanced the tool, adding the possibility of preliminary sampling the input traffic (with configurable policies and rates, as it will be explained later on), before experimentally evaluating the features.

More precisely, `tstat` outputs two different kinds of metrics: some are *per-flow measure-*

Table 7.3: Summary of *aggregated features*, divided by protocol. We report the number of distinct features and, in boldface, the number when considering different traffic directions (i.e., incoming, outgoing, local).

| Type | Protocol | Example | Features | |
|---|---|---|---|---|
| | | | Adirectional | Directional |
| Single Packet | IP | Packet Length | 5 | **15** |
| | UDP | Destination Port | 6 | **14** |
| | TCP | Destination Port | 11 | **21** |
| Multiple Packets | TCP | Maximum RTT | 16 | **20** |
| | RTCP | Average bitrate | 11 | **39** |
| | RTP | Stream bitrate | 21 | **63** |
| | | Total | 70 | **172** |

Table 7.4: Summary of TCP *per-flow features*, divided by category. Number of features in boldface again includes multiple directions (i.e., client-to-server, server-to-client).

| Category | Example | Features | |
|---|---|---|---|
| | | Adirectional | Directional |
| Flow ID | IP address | 2 | **4** |
| Flag counts | Number of ACKs sent | 5 | **10** |
| Volumes | Number of bytes sent | 9 | **18** |
| Packet size | Max segment size | 3 | **6** |
| Window size | Maximum congestion window size | 9 | **18** |
| Timings | Mean RTT | 7 | **14** |
| Congestion control | RTX timeout | 8 | **16** |
| Flow duration | Completion time | 5 | **5** |
| | Total | 48 | **91** |

*ments*, i.e. the tools gives the value assumed by the feature for each observed flow; others are *aggregated measurements*, in the form of distribution of the values assumed by the metrics over all observed flows. Notice that these are just two different points of view of the same measurements (in fact most features are available in either flavor), but they are naturally suited for radically different types of analysis. In this work, we take advantage of either viewpoints, as each of them is best instrumental for one of the two objectives of this work: namely aggregated measurements better reflect monitoring applications, while per-flow measurement are more suited for the classification task.

We omit the complete list of features, which can be found in the tool website [16] as well as in Appendix B. Tab. 7.3 is a condensed view of the *aggregated features set*, listing the number of features related to different network protocols. As most features are evaluated on different traffic directions we report both the number of distinct adirectional features and the number when considering traffic directions with respect to the measurement point (i.e., incoming to the measurement point, outgoing from the measurement point, local but switched at the measurement point). For what concerns *per-flow features*, we basically concentrate on TCP properties, as they are the most interesting ones for the classification in reason of the protocol breakdown shown early in Tab. 7.2. Tab. 7.4 lists flow-level features divided by type of property that are related to, e.g., traffic volumes, congestion control, timings or TCP flags. Again, the table contains both the number of distinct adirectional features and the number considering traffic directions with respect to the flow initiator (i.e., client-to-server and server-to-client).

We underline that the features we consider are substantially in agreement with the feature set listed in [129], which contains an exhaustive set of features for traffic classification. This agreement follows from the fact that `tstat` started as evolution of Shawn Osterman's `tcptrace`[14], which is also used by authors in [129]. At the same time, the match is not perfect, as e.g., [129] misses some features of `tstat` (e.g. flag stating whether a TCP flow has been interrupted [152], detailed counters about anomalous TCP behavior [126], etc.) and `tstat` does not implement all features listed in [129] (such as the Fast Fourier Transform of the packet inter-arrival time, or the count of valid RTT samples, etc.).

## 7.3 Methodology

First, we detail the different *sampling policies* we apply to packet level traces (Sec. 7.3.1). Second, we present the *statistical metrics* which measure the distortion induced by sampling on feature distributions (Sec. 7.3.2).

### 7.3.1 Sampling Policies

We implemented in the `tstat` tool different sampling policies as defined in [189], and that we overview in the following, explaining their peculiarities with the help of Fig. 7.1. The picture shows how different sampling policies with the same sampling step $k = 4$ operate on the same sequence of packets; in the picture, packets are represented with different levels of gray associated to different flows, where an "S" denotes a SYN packet.

- **Systematic sampling**: packets are sampled in a deterministic fashion, with 1-out-of-$k$ packets selected. In the example it can be seen that for each 4-packets window, the first packet is always selected.

- **Random sampling**: packets are sampled at random, in particular each packet is sampled independently at a rate $p = 1/k$. As displayed in the example, since the process is completely random, packets might be sampled in sequence, or there may be several consecutive unsampled packets (obviously with a geometrical decreasing probability).

- **Stratified sampling**: $k$ consecutive packets are grouped in a window, in which a single packet is randomly sampled. Looking at the picture, for each 4-packets window, one and only one packet is always selected, but, unlike systematic sampling, instead of selecting always the first, the algorithm randomly chooses which packet to sample out of the four.

- **Systematic SYN sampling**: is the superposition of two independent processes: (i) a systematic sampling process, which selects every $k$-th packet; (ii) a process which selects all TCP packets with the SYN flag active. This is particular evident in the illustration, where you can see that this policy selects all the SYN packets, in addition to all the packets that a normal systematic sampling would pick.

The first three sampling strategies belong to the family of *unbiased* algorithms, which are the simplest one, being completely unaware of any traffic property. Since these algorithms are extremely lightweight, they are commonly implemented in network equipment, reason why we are particularly interested in their performance. The last one, instead, is what is usually called a *smart* sampling algorithm, because some intelligence is introduced to sample the "right" packets, i.e. the

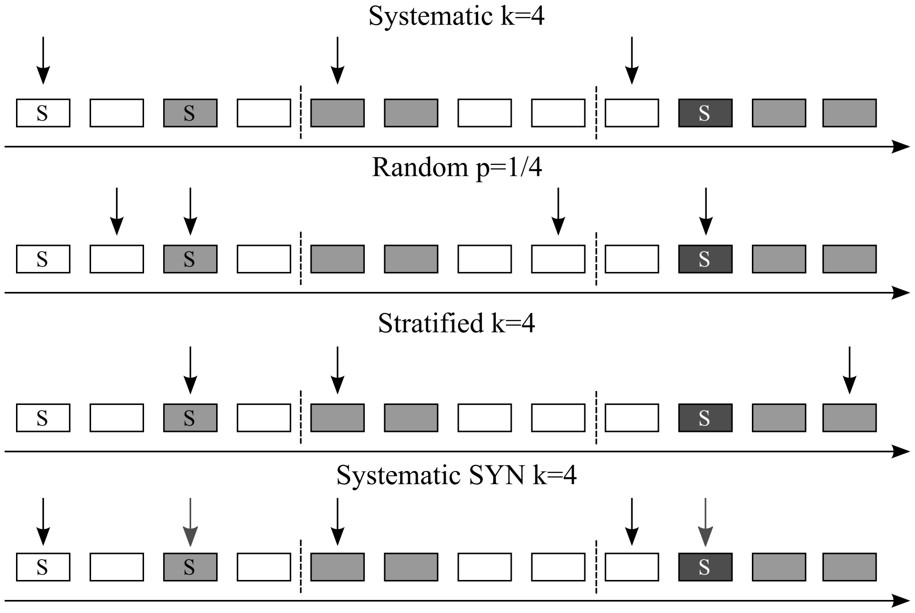


Figure 7.1: Illustration of sampling policies.

ones conveying the most precious pieces of information. There is no intrinsic limit to the amount of intelligence that one can put in such a sampling method and several smarter algorithms have been proposed by researchers; still, it should be remembered that the purpose of sampling is to reduce computational consumption, thus we want to keep the policy as simple as possible. We argue that Systematic SYN sampling represents a good compromise between these two aspects, particularly for traffic classification. On the one hand, as shown in [146], it improves the estimation of aggregated traffic counters (e.g. total flow length) which are known to be particularly important for traffic classification. Moreover, it ensures that at least one packet for each flows is sampled, or, in other words, that all flows are seen: this solves the problem of results representativeness faced in [43, 97] for traffic classification. On the other hand, computational complexity is very low, since the algorithms needs just a counter and a simple check on packet header (furthermore at a fixed offset) to choose whether to sample a packet.

### 7.3.2 Metrics

In order to quantify the distortion introduced by any sampling policy into the metrics features by `tstat`, we consider different statistical indexes, suited for either aggregated or per-flow features.

#### 7.3.2.1 Aggregated features

Denote by $P$ an unsampled feature, which is described by the probability density function $p(x)$ measured over the traffic aggregate. Denote by $Q$ the same feature as measured under a sampling process, which is then described by the probability density function $q(x)$ measured over the sampled traffic. To express the distance between $p(x)$ and $q(x)$ we consider the following standard metrics:

- **Fleiss Chi-Square ($\phi$)**

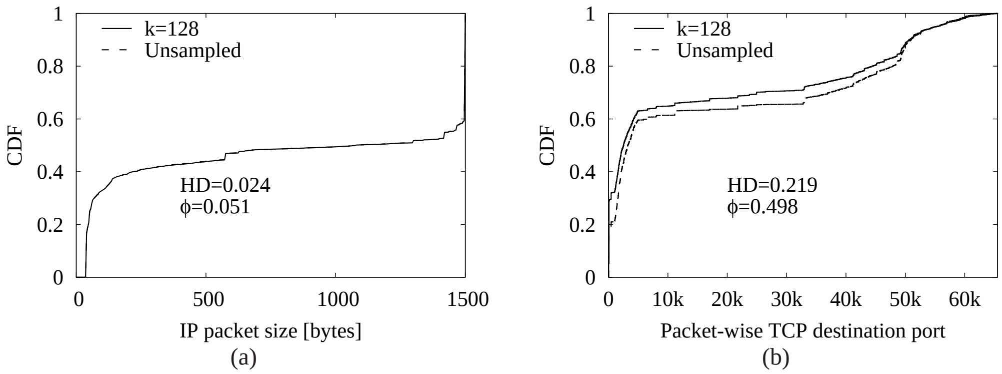


Figure 7.2: Example of distortion of aggregate features (Campus dataset): CDF of IP packet size (a) and number of packets per destination TCP port (b). Plots report the CDF gathered from the unsampled vs sampled traffic aggregate, along with the statistical indexes of distortion.

$$\phi(p,q) = \sqrt{\frac{\sum_{x\in X}[q(x)-p(x)]^2/p(x)}{\sum_{x\in X}[q(x)+p(x)]}} \tag{7.1}$$

- **Hellinger Distance (HD)**

$$HD(p,q) = \sqrt{1-\sum_{x\in X}\sqrt{p(x)q(x)}} \tag{7.2}$$

To provide backward compatibility with [55], we consider the $\phi$ metric, which is a normalized version of the standard Chi-Square metric: increasing values of $\phi$ correspond to increasing distortion. As the Chi-Square statistic is sensitive to the size of the data set, this makes it difficult to compare samples of varying sizes: thus, it cannot quantify significant trends when varying the sampling fraction. Fleiss' definition of $\phi$ directly derives from Chi-Square but overcomes this limitation, being independent from the sample size [55].

The Hellinger Distance (HD) is typically used as a score of similarity between metrics: HD values are confined in the range $[0,1]$, with lower values corresponding to higher similarity between the distribution under comparison.

To have a first idea of the scale of the distortion scores defined so far we provide a preliminary example of some relevant features. Fig. 7.2-(a) and Fig. 7.2-(b) report the CDF of two features, respectively counting the IP packet size in bytes and the number of packets directed to a given TCP port. CDFs for the Campus trace are reported for both original unsampled traffic, as well as for uniformly sampled traffic with $k = 128$. Values of distortion metrics are also reported in the picture. The CDF of the packet-wise destination port (Fig. 7.2-(b)) shows a moderate distortion, with a corresponding degradation of $HD = 0.219$ and $\phi = 0.498$: in this case, differences in the CDF, although of small dimension, can be seen with naked-eyes from the plot. Conversely, IP packet size (Fig. 7.2-(a)) shows a degradation score of about one order of magnitude smaller for both metrics $HD = 0.024$ and $\phi = 0.051$: in this case, no remarkable difference appears from the plot.

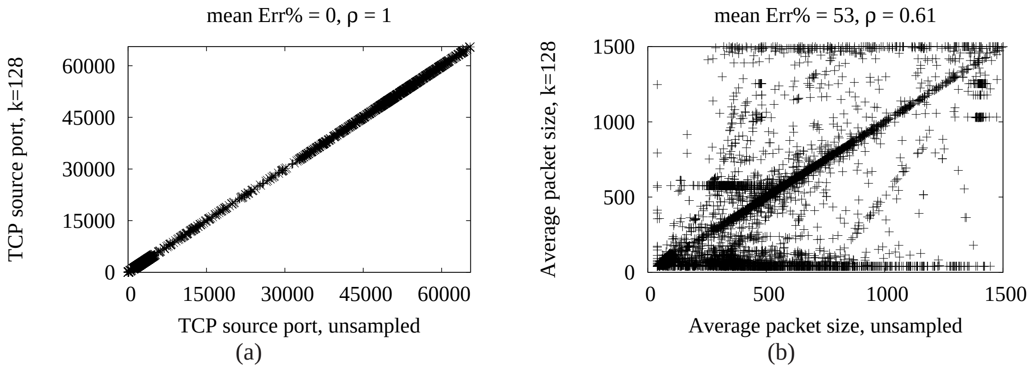


Figure 7.3: Example of distortion of per-flow features (Campus dataset): scatter plot of TCP source port (a) and average packet size (b) for unsampled vs sampled traffic, along with statistical indexes of correlation.

#### 7.3.2.2 Per-flow features

Quantifying the distortion in the case of per-flow features is not only useful for monitoring purposes, but also for the classification process. In this case, we compare the exact values measured by our monitoring tool for the same flow with and without sampling. We used two classical metrics to measure the distortion: (i) the mean percentage error *Err%* and (ii) the correlation coefficient $\rho$ between the sampled and unsampled values. The mean percentage error tells us how much the sampled values diverge from the unsampled ones: the smaller the distortion the better; the correlation, instead, tells us whether a linear dependence exists between the unsampled and sampled values. Like the previous two statistical metrics, the scatter plots of Fig. 7.3 show two examples of features and the corresponding value of the distortion scores. The right plot is again the average packet size per flows, the same feature whose distribution has been shown in Fig. 7.2-(a), whereas the left plot shows the source TCP port. The $x$ coordinate is the value of the feature for unsampled traffic, while the $y$ coordinate is the same feature when a systematic sampling with $k = 128$ is applied.

Two opposite behaviors stand out from the pictures. The feature displayed in the left plot is correctly estimated simply by inspecting a single packet header: for this reason no error is observed and the correlation is maximized. As we will see later on, this kind of features will prove the most valuable discriminators for traffic classification under sampling. On the contrary, the feature in the right plot, being an average, depends on the observation of several packets: therefore, we observe a substantial distortion introduced by sampling, testified both by the large value of relative error, and the lower correlation coefficient as well. In fact, despite many points still align on the $y = x$ bisector line, we can notice a large number of flows falling on a few distinct horizontal lines (namely $y = 40, 576, 1500$). We found that only a single packet was sampled from these flows, which is not representative of the average packet size. In fact, with a single observation, it is likely to get a typical-sized packet (e.g, a $40-$byte packet without data, or $1500-$byte full payload packet, or a $576-$byte packet) which will lead to a bad estimation of the actual average packet size of the flow.

Actually, in the second part of this work, we will be interested more in how sampling affects the relevance of features for the classification, rather than in their mere distortion. Therefore, we need a metric able to capture how much information regarding the application label is conveyed by any

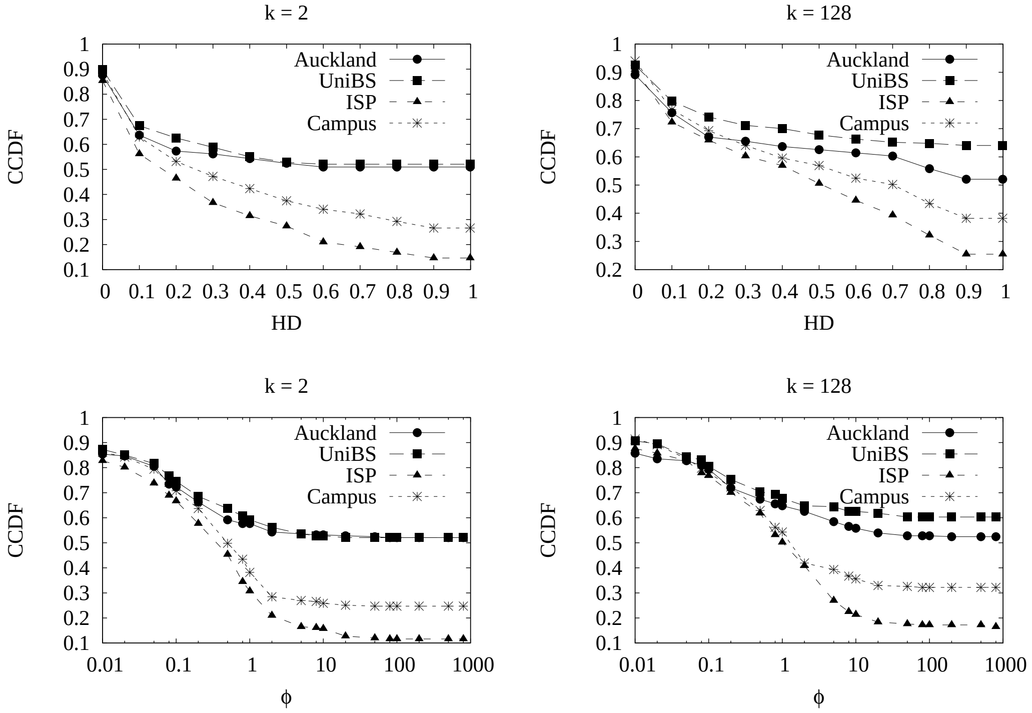


Figure 7.4: Distribution of the Hellinger distance and $\phi$ coefficient over the whole features set for systematic sampling and $k \in \{2, 128\}$.

given features. For this purpose we resort to the *information gain* [127] metric from information theory, which has been already introduced in Chap. 3.

## 7.4 Aggregate Feature Distortion

In this section we concentrate on the pure distortion introduced by sampling in traffic measurements, in particular observing the impact it has on aggregate traffic metrics. We first characterize the overall feature set and the range of value scored by our statistical indexes, in order to have a general picture (Sec. 7.4.1). We then focus on smaller sets of features defined by the protocol layer they pertain to (e.g. network or transport layer), identifying which family of attributes is more heavily affected by sampling (Sec. 7.4.2). We finally individuate a set of robust features, whose distortion keeps bounded even under heavy sampling, on which we investigate the impact of different sampling policies (Sec. 7.4.3).

### 7.4.1 Overview of Sampling Impact

In this first part we look at the complete set of features at once, in order to observe the general trend of feature degradation under sampling and to better understand the range of variation of the statistical metrics. For this purpose we use the distributions of the two distortion scores over the whole set of features reported in Fig. 7.4 and represented in the form of complementary cumulative distribution function (CCDF). The graphs report the distributions for systematic sampling with $k = 2$ and $k = 128$, respectively in the left and right column, while the top row is related to

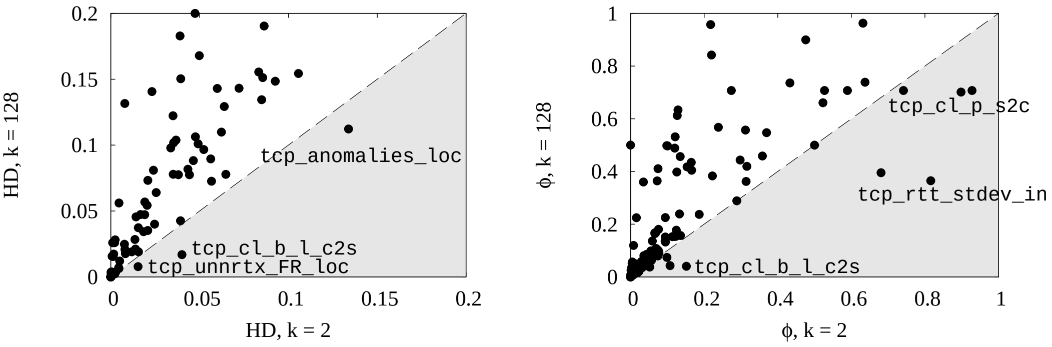


Figure 7.5: Scatter plot of Hellinger distance and $\phi$ coefficient over the whole features set for systematic sampling with $k \in \{2, 128\}$.

the HD score and the bottom one to the $\phi$ coefficient. We keep a separate curve for each dataset as they represent different network conditions and traffic mixture, thus we prefer to observe their behavior individually.

At first glance, the two statistical metrics are quite coherent with each other, showing practically the same trends, even though HD is bounded in the interval $[0, 1]$ whereas $\phi$ is not. Moreover, it looks as Campus and ISP appear more robust than the other traces, as features degrade more gently: the heterogeneous traffic mixture found in this traces is likely the key reason why they are less affected by sampling.

On the contrary, it is quite striking that for the other traces around 60% of features are completely distorted (i.e., the distance metric is maximized) already with $k = 2$: this means that they are impossible to evaluate even with very light sampling. On the other hand, 10% of features show no distortion at all, scoring a zero for both metrics, meaning that they are perfectly estimated regardless of the sampling rate. As a matter of fact, all these features are not distorted as they can be correctly measured simply by inspecting one single packet of any given flow.

To dig further, we look at the same data in a different way by means of the scatter plots of Fig. 7.5, where each feature is represented with a point whose $x$ and $y$ coordinates are respectively the distortion score for $k = 2$ and $k = 128$. We have zoomed to show the area close to the origin, where we see a cluster of points showing no degradation.

However, the pictures show an interesting artifact: notice that the estimation of a few features seems to improve under heavy sampling, as shown by the points falling in the gray-shaded part of the graphs (some of which are labeled with the feature name). This weird effect is mostly due the different way sampling impacts on long and short flows, since long ones have a larger probabilities of being sampled while the short ones are likely to disappear after sampling. For instance, this effect is particularly evident for the `tcp_cl_b_l_c2s` feature, i.e. the TCP flow length, measured with a coarse granularity. In this case, for larger sampling steps, many short flows are no longer sampled, with a corresponding decrease of the mass of flows falling into the smallest bins. Thus the improvement of the feature estimation is a joint consequence of the traffic nature (sampling tends to select packets from the same elephant flows, yielding a better estimation of the length of such flows) and the specific binning adopted (as this behavior is not shown by the corresponding feature calculated with finer granularity `tcp_cl_b_s_c2s`, since in this case it is less likely for the sampled feature to fall into the same bin of the unsampled traffic).

Notice that this effect is instead less evident in the $HD$ score plot, where only a single feature falls in the gray region, than in the $\phi$ plot where we actually find more points in this area. Moreover for the $\phi$ coefficient many features actually fall closer to the bisector as well, which means that only a slight degradation is detected in spite of an increased sampling rate. In fact, it seems as though different choices of binning have a greater impact on the $\phi$ metric, sometimes compromising its accuracy. On the other hand, the $HD$ distance appears able to better characterize the distortion, because a greater score usually corresponds to a larger sampling step. This is due to the different weighting of the errors in $\phi$ and $HD$: in the former, larger discrepancies will be amplified (i.e., squared difference) with respect to the latter score (i.e., product): this entails that several small errors, affecting several bins, may produce a larger distortion score in $\phi$. The main outcome of this behavior is that special care must be also taken in the selection of the distortion metric used, as otherwise similar artifacts may yield to misleading conclusions.

### 7.4.2 Impact of Protocol Layer

We now group the features in different subsets according to the protocol layer: in particular we consider IP features, UDP single-segment features, TCP single- and multiple- segment features as in Tab. 7.3. By comparing the effect of sampling on these groups, we want to find out whether there exists a family of features which is by definition more robust to sampling.

In the light of what observed in the previous section, without loss of generality nor of information, we express the distortion scores using the Hellinger Distance alone. For the time being, we still focus on a single sampling policy (namely, systematic sampling), delaying the consideration of different sampling policies to the following section. However, we do take into account a large range of sampling rates, from 1/2 to 1/1024. Results are reported in the four graphs of Fig. 7.6, corresponding to the different datasets. In every single plot, each curve depicts the mean and the variance of the HD metric over a given group of features as a function of the sampling step $k$.

A general observation which holds for all of the datasets, is that some features prove to be intrinsically easier to measure under sampling. For instance the curves of distortion scores for both IP and UDP single-segment features are considerably closer to the minimum value for the HD across all datasets (apart from Auckland, as we will explain later). This confirms our previous intuition that features relying only on the inspection of a single packet (e.g., IP packet size) are more robust to sampling than features depending on the observation of multiple packets (e.g., RTT time).

On the contrary, while ISP and Campus exhibit rather consistent and coherent trends, Auckland and UniBS have some anomalies. For instance, in the Auckland trace, the single-segment TCP features have an unusually low distortion score. Investigating this issue further, we found that in this dataset TCP options (e.g. MSS negotiation, window scale) are obfuscated for privacy reasons (actually set to 0) together with the rest of packet payload, which makes `tstat` unable of correctly estimating the related features (i.e., more precisely, `tstat` assumes a maximum value for MSS, and by default considers timestamp, window scale and sack options as unused). Therefore, in this case the low distortion score is an artifact, arising from the impossibility of correctly estimating most of the features of that group from the trace under investigation, even in the unsampled case. Instead, the higher degradation detected for TCP features in the UniBS trace derives from the artificial nature of this trace, built with a few hosts automatically generating specific traffic. As a result, the trace includes many elephant flows (notice the large mean flow size in Tab. 3.2) and when sampling is applied this strongly biases the distributions, yielding larger distortions.

Campus and ISP instead, show a similar behavior when considering TCP features as well. Interestingly, notice that, at lower sampling rates, TCP features depending on multi-segment suffer a smaller distortion than features depending on single-segment observation. Also, the HD value for TCP multi-segment features keeps increasing with the sampling, whereas TCP single-

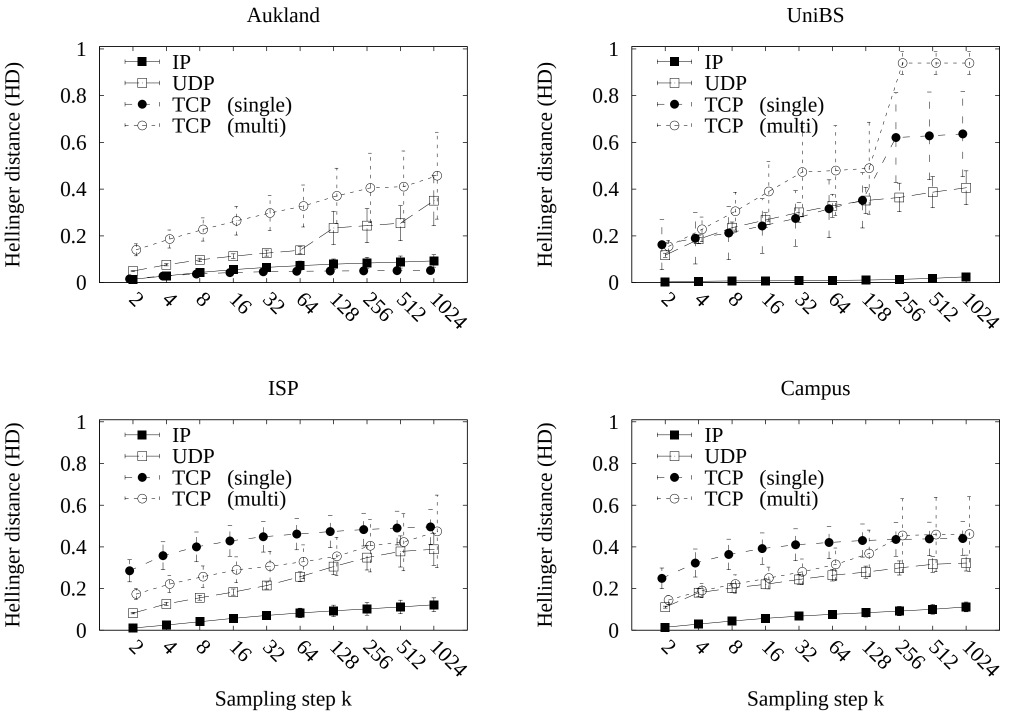


Figure 7.6: Mean and variance of the HD distortion score, for features grouped by protocol layer, as a function of the sampling step under uniform sampling policy. The extremely low distortion of "TCP single" features in the Auckland trace is an artifact due to the lack of packet payload, in particular of TCP options.

segment features, albeit already distorted for low levels of sampling, do not further degrade for high sampling factors. This unexpected behavior is due to the fact that, in the TCP case, some of the single-segment features require *specific segments* to be monitored: for instance, the segment corresponding to the negotiation of a specific option. If this segment is missed because of sampling, which is often the case already at low sampling rates, the features estimation is compromised. Conversely, some of the features requiring multiple segments (e.g., average and maximum value of the receiver window, etc.) can be safely estimated for low sampling steps, as all segments anyway carry useful information that can improve the feature estimate.

### 7.4.3 Impact of Sampling Policy

In this section, we assess the impact of different sampling policies on the accuracy of traffic feature estimation. However, first we need to define the subset of features to use for our analysis: this is not an easy choice given the large number of features already distorted for small sampling steps on the one hand, and the extremely varied behavior of different feature groups across each trace. Therefore, we adopt a simple but effective threshold-based selection criterion: we consider as robust, and focus on in this section, all features whose $HD$ distance is lower than a predefined threshold. Hence, we no longer take into account the grouping by protocol layer when applying the robustness criterion: rather, features are evaluated individually, so that the robust set actually consists of properties belonging to different groups. As we also consider each direction separately (i.e., incoming versus outgoing versus local traffic), it may happen that a feature is robust for a

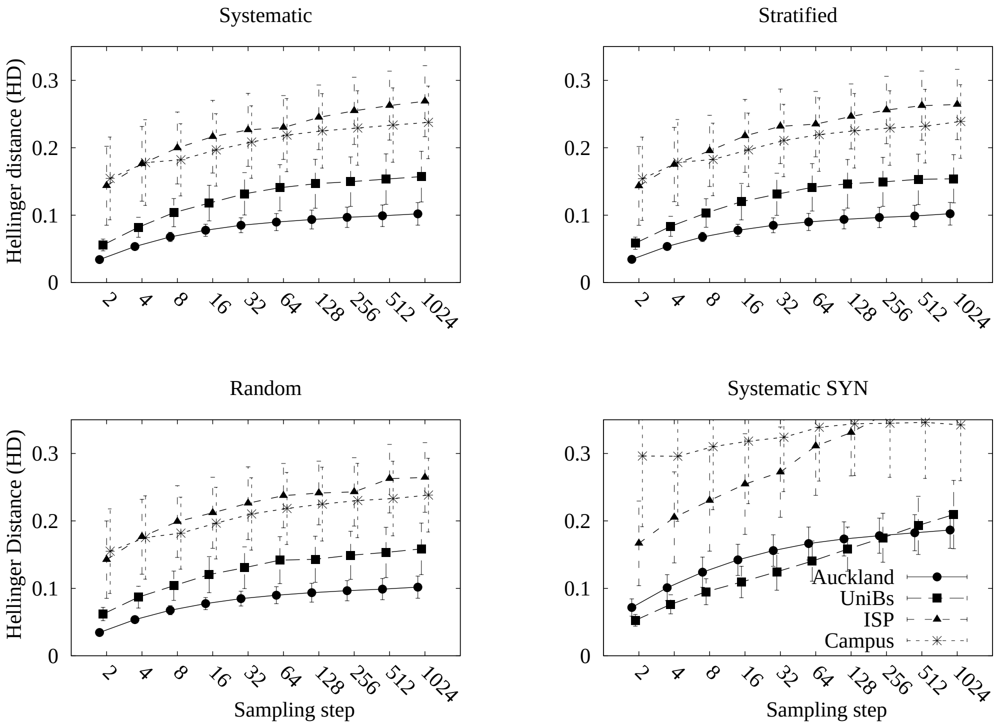


Figure 7.7: Mean and variance of the HD distortion score for the robust group of features as a function of the sampling step for different sampling policies.

given direction, but not for the opposite one. Moreover, we conservatively require features to be *jointly* robust across all datasets under consideration: in other words, the resulting set is the *intersection* of the sets of robust features on each single datasets.

Without loss of generality, results in this section refer to features which have an $HD < 0.3$ with a sampling of $k = 128$. Notice that we select this values of threshold in reason of the knee of the distribution observed in Fig. 7.4. Notice also that different threshold values, as the $HD < 0.1$ we used in [142], yield to similar considerations: yet, as in this work we consider a larger dataset, and we require features to be robust in all datasets, we prefer to apply a less stringent threshold, so to assess the impact of sampling policy on a larger number of features.

The final set contains 36 features, equally distributed over the 3 protocols IP, TCP and UDP. Thus, each protocol layer is represented in the robust set, except for the RTCP and RTP layers. In fact, the relatively low amount of RTP/RTCP traffic present in the Auckland dataset makes it difficult to evaluate the related features for this trace, especially when hard sampling conditions further limit the number of valid samples.

Results of this analysis for the robust features set are reported in Fig. 7.7, composed of one graph for each sampling policy. We employ an exponentially increasing sampling step $k = 2^i, i \in [1 \dots 10] \subset \mathbb{N}$, reported on the x-axis of every plot. Each graph contains four curves, one for each dataset, depicting the average distance score over the robust features set; variance of the distance score is also reported by means of vertical error bars (notice that we employ variance instead of standard deviation, as the latter is visually noisy, as the square root of HD values in $[0, 1] \in \mathbb{R}$ explodes).

At first glance, we can observe that there is no clear advantage in the choice of random sampling or systematic sampling: considering the corresponding three plots, one can gather a striking similar behavior. This finding holds whenever several features are considered, and contrasts with earlier results supporting stratified sampling techniques [55]. Our intuition is that, given the level of statistical multiplexing of traffic flows, the sampling policy has a minor impact, especially when complex traffic properties are considered. Also, notice that similar conclusions have been recently reported by independent research[49], which however limitedly considers only traffic volume measurements under sampling (i.e., flow length).

Conversely, our smart sampling policy has a noticeable impact on measurements accuracy, yielding completely different trends in addition to high distortion scores for high sampling steps. Intuitively, Systematic SYN sampling heavily biases the distribution: indeed, it samples at least one packet for each flow by definition, which for high sampling steps is also the only sampled packet, which hence introduces a significant distortion in the aggregate features. While, as we will see, traffic classification accuracy is not affected by this distortion, for the time being we can conclude that biased techniques are not indicated in the context of traffic monitoring and characterization.

If we focus on the difference between datasets, we see that they are ranked almost in the same order by different sampling policies: apparently Auckland and UniBS are the easiest one, which seems to be in contrast with our previous analysis. Yet, recall that we have conditioned distortion scores over the robust subset of features: in the case of Auckland, this means that we are averaging with TCP features getting extremely low scores due to the measurement artifact early outlined; in the case of UniBS we have removed exactly those features that we saw heavily distorted in Fig. 7.6.

It is also worth noticing that sampling error saturates, in the sense that distortion scores do not increase as fast as the sampling step, which is exponential. The reason of such a behavior is twofold: first, as most features are estimated from the observation of a single packet, they degrade gently when increasing the sampling step; second, some of the artifacts showed in the previous section may still arise (i.e., features whose distortion score decreases rather than increasing for higher sampling rates).

## 7.5 Single-flow Feature Distortion

In this section we study the distortion of per-flow features and, with respect to the former section, we change both the viewpoint and the metrics employed to measure the impact of sampling. We also apply the information gain metric to assess the amount of information conveyed by each feature: such analysis is instrumental for the evaluation of traffic classification accuracy, that we will carry on in the next section. To avoid cluttering the pictures, this preliminary analysis will be done considering only the UniBS dataset, whose ground-truth is the most reliable; however, we will come back to the whole dataset for our last experiments to draw our general conclusions.

In the remainder of this section we always refer to Systematic SYN sampling, which was introduced specifically for traffic classification, as it gives us two main advantages over the other policies. First it overcomes the problem of dataset representativeness, for at least one packet per flow is always sampled: in this way even protocols with short flows (corresponding to a small probability of being sampled) are, nevertheless, included in the dataset after sampling. Second, the SYN packet carries very important information about the related flow (e.g. initial sequence number, initial timestamp, eventually some TCP options) which `tstat` can leverage to improve the estimation of many flow properties.

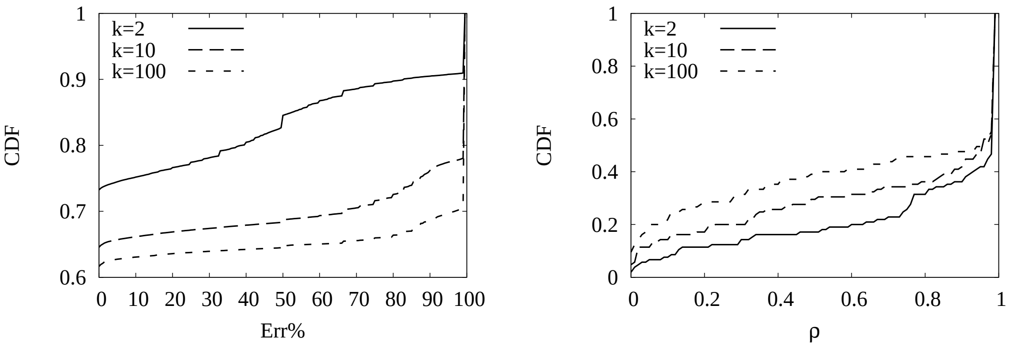


Figure 7.8: CDF of (left) Err% and (right) $\rho$ for UniBS trace and different sampling step.

### 7.5.1 Overview of Sampling Impact

Following the same approach of Sec. 7.4, we first consider the whole set of per-flow features, to gather an overall picture of the impact of sampling. Fig. 7.8 shows the distributions of the relative percentage error and the correlation coefficient between the sampled and unsampled values of all features for the UniBS trace, comparing different sampling steps represented with distinct curves.

Focusing on the percentage error plot on the left, a large number of measures appears not affected by sampling, scoring a 0% error; we must not forget, however, that the distribution may be biased towards zero by features that are estimated by a single packet inspection, or that score their default value that `tstat` assigns to features it cannot evaluate. On the other hand, an increasing portion of the features, from 10% for $k = 2$, to 30% for $k = 100$, are completely distorted with an error greater than 100%. Interestingly, the number of features falling between these extremes is decreasing with the increasing sampling step. However, high distortion does not necessarily imply that such features is useless for traffic classification: indeed, provided that distorted features are still clearly *separable* across applications, their information would still be extremely valuable for classification purposes. For completeness sake, we reported also the correlation coefficient distributions in the right plot, from which the same conclusion can be drawn the same percentage of features that scored a 0 error gets the maximum value of correlation, and again higher sampling steps cause a larger degradation highlighted by a larger portion of features with small correlation with the unsampled case.

Let us now focus on specific features. We present two examples in the scatter plots of Fig. 7.9: the left plots are related to the maximum packet size observed in a flow, while the right ones show the average RTT. On the x-axis we report the value of the feature in the absence of sampling, while on y-axis the one when a sampling with $k = 100$ is applied. We represent flows belonging to traditional client-server (CS) applications in the top plots and to peer-to-peer (P2P) applications in the bottom plots, in order to see whether different classes of applications correspond to different behaviors under sampling.

The plot allows us to gather some observations. For the packet size, while P2P applications employ either very small (signalling) or full-size (data) packets, client-server applications often use medium-sized packets as well. Yet, for CS applications the features are underestimated mostly of the time, meaning that usually only a single small packet (e.g., SYN) of a short CS flow is sampled due to biased-SYN sampling; on the other hand, this same feature is correctly evaluated for P2P applications, whose longer exchanges increase the odds that the monitor tool samples bigger packets as well. In the right plots, we can find again two clouds: the top one, concentrated specially in the top left corner for P2P, and the bottom one, concentrated in the area close to the

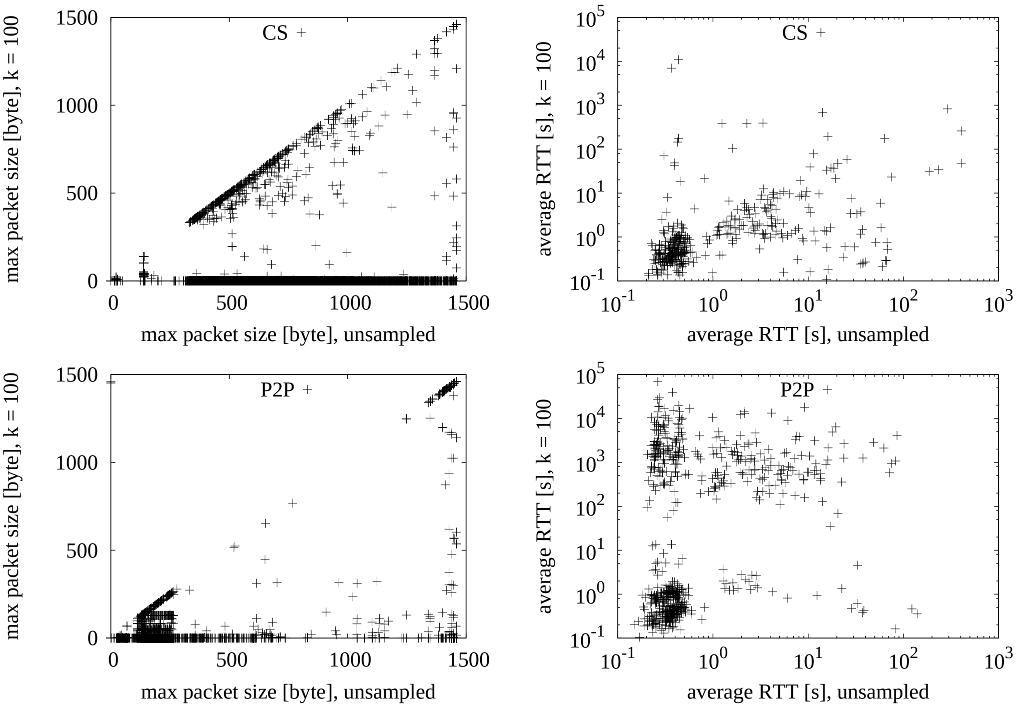


Figure 7.9: Scatter plot of features values for unsampled and SYN Sampling k = 100 for UniBS trace, contrasting peer-to-peer (P2P) and traditional client-server (CS) applications.

origin, which is equally composed by P2P and CS flows. A possible explanation of this behavior is that many P2P flows have intermittent but long-lived flows (for instance because they speak with a recurring peer after a while), which yields estimation of very large RTTs (due to pairing the ACK with the wrong data message); on the contrary, CS applications have shorter flows, that reduce the likelihood of pairing a very late ACK and allow an easier measure of the RTT. Again it looks like traffic belonging to different applications is differently impacted by sampling, which is good news for our classification purposes.

### 7.5.2 Ranking Features

After this preliminary evaluation of feature distortion, we want to assess the degradation of the *amount of information* conveyed by features due to traffic sampling. This analysis is complementary to the one presented in [43], where authors analytically study the error introduced by sampling in NetFlow features, but do not evaluate whether sampling also impacts the *information* content of such features, which is a critical component of the classification process.

To measure this quantity we finally use the Information gain score introduced in Sec. 7.3.2.2. We use this measure to partition the features set in two groups, using a simple threshold-based criterion: (i) the *most-relevant* feature group including features whose information gain is greater than 1 bit (i.e. that are able to discriminate between two labels); (ii) the least-relevant feature group, containing all the remaining features. The performance of such subsets of features will be evaluated in the next section; here we report the scores for the most-relevant features in Tab. 7.5 together with their rank, comparing the unsampled case with the sampled case with $k \in \{2, 10\}$.

The information-gain metric partitions our feature set as expected. For instance, features like the client ephemeral source port, which is chosen randomly upon connection setup, clearly ends

Table 7.5: Feature Information gain for UniBS trace at different sampling rates.

| **Features** | **Unsampled** | | **Sampled k=2** | | **Sampled k=10** | |
|---|---|---|---|---|---|---|
| | Score | Rank | Score | Rank | Score | Rank |
| Server-IP-address | **1.68** | 1 | **1.68** | 1 | **1.68** | 1 |
| cwin-min-c2s | **1.49** | 2 | **1.20** | 6 | 0.60 | 14 |
| min-seg-size-c2s | **1.48** | 3 | **1.22** | 5 | 0.47 | 23 |
| cwin-max-c2s | **1.47** | 4 | **1.11** | 8 | 0.56 | 15 |
| max-seg-size-c2s | **1.43** | 5 | **1.17** | 7 | 0.46 | 24 |
| initial-cwin-c2s | **1.41** | 6 | 0.71 | 26 | 0.29 | 32 |
| First-time | **1.37** | 7 | **1.37** | 2 | **1.37** | 2 |
| cwin-min-s2c | **1.35** | 8 | **1.06** | 11 | 0.53 | 16 |
| Server-TCP-port | **1.34** | 9 | **1.34** | 3 | **1.34** | 3 |
| initial-cwin-s2c | **1.33** | 10 | 0.77 | 22 | 0.30 | 31 |
| Client-IP-address | **1.31** | 11 | **1.31** | 4 | **1.31** | 4 |
| cwin-max-s2c | **1.28** | 12 | 0.99 | 14 | 0.49 | 21 |
| min-seg-size-s2c | **1.22** | 13 | 0.96 | 16 | 0.51 | 19 |
| max-seg-size-s2c | **1.21** | 14 | **1.03** | 12 | 0.50 | 20 |
| Last-time | **1.14** | 15 | **1.09** | 9 | **1.02** | 5 |
| win-max-s2c | **1.08** | 16 | **1.07** | 10 | 0.98 | 6 |
| Completion-time | **1.03** | 17 | 0.97 | 15 | 0.42 | 25 |
| win-min-s2c | **1.02** | 18 | **1.01** | 13 | 0.94 | 7 |
| unique-byte-s2c | **1.02** | 19 | 0.74 | 23 | 0.42 | 27 |
| data-byte-s2c | **1.01** | 20 | 0.74 | 24 | 0.42 | 26 |

up in the least relevant feature set, being useless for the classification process. On the other hand, the server IP address exhibits always a high score regardless of the sampling rate: both the fact that this feature is correctly estimated by inspecting a single packet and the reduced number of servers in the UniBS trace concur to make this value a strong discriminator.

Besides, notice that the larger the sampling period, the smaller the number of features showing scores greater than 1 bit. Additionally, since the ranking changes from one sampling rate to the other, this suggest that the most-relevant feature set gathered for unsampled traffic may no longer be the same for higher sampling.

To visually represent the effect of sampling on the most-relevant feature set, we use the parallel coordinate plot of Fig. 7.10-(a). Each line represent one feature and connects the Information gain score and mean percentage error for that metric for the two sampling steps $k \in \{2, 10\}$. Two evident patterns emerge from the picture, which are highlighted by means of two different line types. Full lines clearly represent those features evaluated from a single-packet inspection: they have high information-gain, which means they are extremely correlated with the application label, along with a low relative error distortion, which means they are correctly measured regardless of the sampling step. On the other hand, features denoted by dashed lines, though more degraded by sampling (their percentage error increases with sampling), are included in the most-relevant set because they still bring benefit to the classification, as testified by high information gain for $k = 2$, moreover only moderately decreasing for $k = 10$. Hence, while these features are degraded under sampling, they still allow to separate application labels as exemplified in Fig. 7.9

In Fig. 7.10-(b) we extend our considerations to the whole set of features: each feature is represented by the point whose coordinates are the information gain score for $k \in \{2, 10\}$ respectively on the x and y axis, using different point types for most relevant (empty squares) and least-relevant (filled circles) features. First, the picture confirms that the ranking of features is not stable: notice that the score of some least-relevant features exceeds the one of a few most-relevant ones for $k = 2$ (remember that the partition of the feature space has been performed using the ranking of unsampled features). This behavior is more evident considering $k = 10$: some features in the least-relevant group should be considered as good discriminators, not only because of the higher

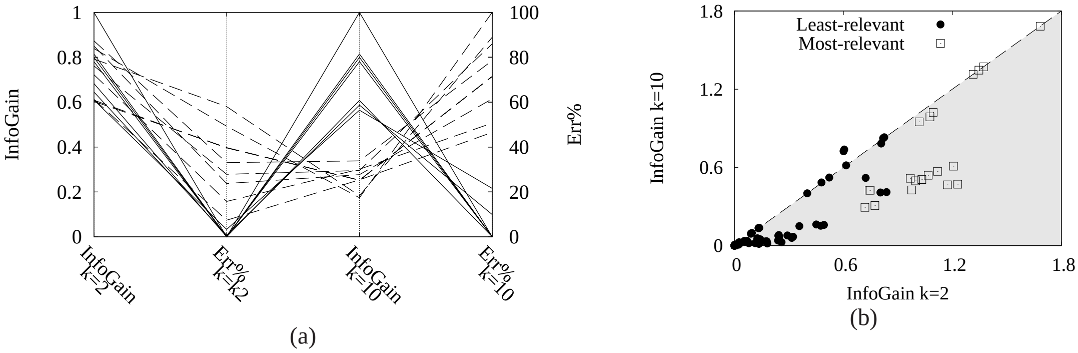


Figure 7.10: (a) Parallel coordinates plot for most-relevant features and (b) scatter plot of information gain for all features with $k = 2, 10$.

scores, but rather because they fall on the bisector, showing no degradation of their information content. Overall, it looks like it is extremely difficult to predict how the information content of a feature might be degraded by sampling, as this may furthermore vary depending on the sampling step. This suggests using the whole features set and letting the classification algorithm deal with that. In the next section, which compares the classification accuracy of different features sets, we will see whether this is a good strategy.

## 7.6 Traffic classification under sampling

After the analysis of the distortion due to sampling in both aggregate (Sec. 7.4) and single flows features (Sec. 7.5), in this section we provide a detailed evaluation of the impact of the sampling on traffic classification. More precisely, we first gather baseline performance for different features sets for unsampled traffic (Sec. 7.6.1) and then we measure the impact of training policies (Sec. 7.6.2) for sampled traffic, considering the UniBS trace. Finally, we extend our investigation to other datasets as well (Sec. 7.6.3). We report the overall accuracy of the classification and omit the detailed per-application accuracy as such an analysis is already found [43] and we rather concentrate on uncovered aspects of traffic classification under sampling.

### 7.6.1 Impact of Feature Set

We start by comparing the classification performance of different sets of features with unsampled traffic, using only the UniBS trace. We consider the following sets:

*S1* **baseline-features** is a simple set of features, that basically contains the information derived by a flow-level monitor (e.g. NetFlow) defined as in [97].

*S2* **all-features** is the whole set of features produced by `tstat`, coherent with [129].

*S3* **no-IPs**, obtained removing from the whole set both source and destination IP addresses, i.e. $S3 = S2 \setminus \{srcIP, dstIP\}$.

*S4* **no-IPs/Time/Flags**, obtained removing from the whole set IP addresses, TCP timestamps and flags, i.e. $S4 = S2 \setminus \{srcIP, dstIP, timestamp, flag\}$.

*S5* **no-Ports**, obtained removing from the whole set both source and destination transport layer port, i.e. $S5 = S2 \setminus \{srcPort, dstPort\}$

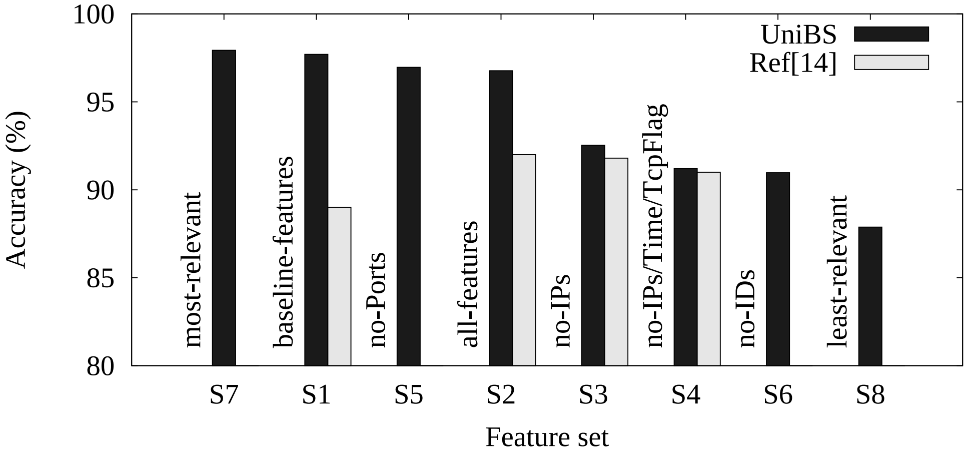


Figure 7.11: Classification accuracy achieved by different features sets with unsampled traffic for UniBS trace.

*S6* **no-IDs**, obtained removing from the whole all flow identifier $S6 = S2 \backslash \{srcIP, dstIP, srcPort, dstPort\}$

*S7* **most-relevant**, comprising the features listed in Tab. 7.5, formally defined as $S7 = \{x \in S2 : InfoGain(x) > 1\}$

*S8* **least-relevant** complementary of the most-relevant features set, i.e. $S8 = S2 \setminus S7$

The first four groups are in common with [97], in order to have a direct comparison with previous work. We notice that features included in the baseline-features set *S1*, such as destination IP and port, start and end time and total transferred bytes, were already identified as the most reliable ones, showing high information gain in our earlier analysis (Sec. 7.5).

The accuracy achieved by these features sets in our experiments is depicted in Fig. 7.11, where we also plotted the results reported by [97], where available, as a comparison. Yet we underline that this is an *indicative* comparison as results are not directly comparable for three reasons: first, the dataset differs; second, the machine learning technique differs; third, there are slight differences in some feature sets as the overall set $S2$ produced by `tstat` is a super-set of the one considered in [97] (see the discussion in Sec. 7.2.2).

Speaking about the comparison, we gather that for the features sets $S1$ and $S2$ the accuracy for the UniBS dataset is higher than that reported in [97], whereas values scored by sets $S3$ and $S4$ are coherent with previous results. To explain this behavior, we must go back to the information gain score. We have noticed before the high correlation between the application label and the network-layer identifiers for the UniBS trace (i.e., IP address of the server), which causes the performance drop from sets $S1$ and $S2$ to sets $S3$ and $S4$.

Considering all subsets, we gather that, as expected, the most-relevant feature set exhibits the best accuracy, though being the smallest one. The complete set $S2$ instead turns out in a slightly worse result: we see a little overfitting phenomenon (i.e. useless features disturbing the classification process), but the difference is negligible in our case. Therefore, we estimate the prominence of feature selection less relevant, and will consider other, less explored, issues in what follows.

We make a few final remarks before changing subject. The performance of sets $S3$, $S5$, $S6$ further shows that IPs are much more relevant than ports, causing a larger decrease in performance when removed from the feature set. Finally the last vertical bar, though referring to a quite numerous set of 85 features, shows as expected the worst performance as it comprises features carrying less information about the application label. Notice that accuracy however exceeds 85%, meaning that the set of the least-relevant features still includes valid discriminators.

### 7.6.2 Impact of Training Policy

Clearly, the selection of flows to include in the training set has a great influence on the final classification accuracy (i.e., how to find representative samples, how to face the class imbalance problem, etc.). Yet, here we are more concerned with a novel factor: i.e., whether training and validation data should be gathered at the same sampling rate.

This is an important point as it means that ISP may need to use different classification models, one per each sampling rate, or may use a single unique model for all rates. In the following, we use $k_T$ to denote the sampling step used for the training data, and $k_V$ for the one used for validation data.

- **Homogeneous classification**, in which training and validation sets contain data obtained with the same sampling rate (i.e., $k_T = k_V$). This corresponds to the case of ISPs using different training sets, one for each sampling rate. Results shown in the previous sections were gathered using this training policy.

- **Heterogeneous classification**, in which training and validation sets contain data obtained with different sampling rates (i.e., $k_T \neq k_V$). Intuitively one may think that richer data with a lower sampling period might contain more information that could be successfully exploited for the classification. At the same time, we may expect that under sampling, the feature estimation error will grow large, with a corresponding information loss. In our experiments we investigated the full space resulting from the cross product of $k_V \times k_T$, but we report here only two examples: first the extreme case where we train the machine with unsampled data ($k_T = 1$) and then the results gathered with $k_T = 2$, i.e., the minimum level of sampling.

We test these policies on the UniBS trace, performing a cross-validation and using 10% of this data as training set and the rest as validation set. The cross-validation procedure repeats the train/validation process 10 times, randomizing each time the training set (and changing the validation set as a consequence). As for the class imbalance, we took extra care in building the training set so that the proportion of the different application are the same of the original data (i.e., as 49% of the original trace is constituted by HTTP flow, the training set is composed for 49% of HTTP flow samples).

In Fig. 7.12 we report the flow (left plot) and byte (right plot) accuracy obtained by the C4.5 algorithm with the complete set $S2$ of features provided by `tstat` (whose performance was only slightly affected by overfitting with respect to the most-relevant feature set), for both the heterogeneous and homogeneous cases, for increasing sampling step. We report two cases of heterogeneous policies: first the case where the classifier is trained with unsampled data, i.e., $k_T = 1$; second the case where the classifier is trained with lightly sample data with $k_T = 2$. As a reference and lower bound, we also plot the results of two dummy classification processes: (i) *Uniform* selects the classification label uniformly at random among the possible classes; (ii) *Proportional* selects the label at random, but with a probability proportional to the number of flows belonging to that class.

Looking at the flow accuracy, the homogeneous case exhibits the best results, achieving an high accuracy which furthermore does not deteriorate under more aggressive sampling. In the

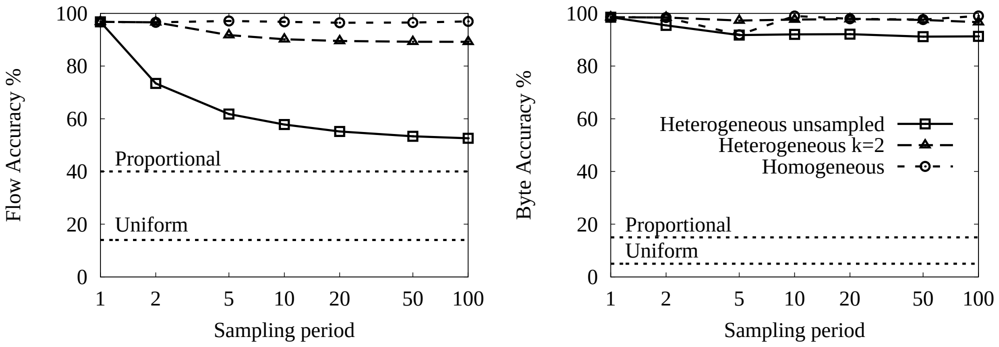


Figure 7.12: Impact of Homogeneous vs Heterogeneous training set policies at varying sampling rates in terms of flow and byte accuracy.

heterogeneous unsampled case, the accuracy drops considerably already with a sampling period of $k_V = 2$ and then decreases until at $k_V = 100$ it achieves only slightly more than 50%, which is close to the proportional random labeling process: in other words, the heterogeneous training process was only able to correctly learn the *flow proportion* (due to our balanced training set selection policy). The heterogeneous case with $k_T = 2$ has a better performance, though far from the homogeneous case: this means that, whatever the sampling step $k_V$, it is always better to train the classifier with already sampled data (i.e., $k_T > 1$) when this is the kind of data to be classified. Interestingly, there is a much smaller difference between the two training policies in term of byte accuracy even in harsh sample conditions: this means that elephant flows are always correctly classified, whereas classification errors are much more frequent for mice flows.

In agreement with [43], these results show that, even though features are distorted, the amount of information they convey on the application label is still relevant for the classification, as shown by the analysis in the previous section as well. On the contrary, while unsampled data captures the properties of real traffic pretty well, it is unsuitable to characterize, and therefore classify, the sampled traffic. Hence, ISPs shall use a specific classification model for each sampling rate. In case of variable sampling rates, this may not be feasible: however, as classification degrades smoothly between the homogeneous $k_T = k_V$ and heterogeneous $k_T = 1$ cases (i.e., $k_T$=10 performs better than $k_T$=1 for $k_V$=100), a viable compromise would be to select the closest $k_T$ from a set of models.

### 7.6.3 Impact of Dataset

Finally, we extend our analysis to the other traces. In principle we were extremely interested in testing the *portability* of the method: i.e., to assess the performance of a model trained on a trace and validated on the other traces. Unfortunately though, due to the heterogeneity of the dataset, a thorough and coherent comparison is possible only with an extremely reduced set of protocols (only HTTP, HTTPS and IMAPS are common to the three traces, and no more than 4 protocols are common to any 2 traces), resulting in a statistically not significant comparison. Therefore, we limitedly report results considering each dataset in isolation. We use a sampling period of $k = 100$ (a common value used in operational networks), the C4.5 algorithm in the homogeneous case, with data split again in 10% and 90% for training and validation sets respectively. The whole classification process is performed separately for each dataset, using the sets of features defined in Sec. 7.6.1, and results are reported in Fig. 7.13. Notice that, as the Auckland ground truth is

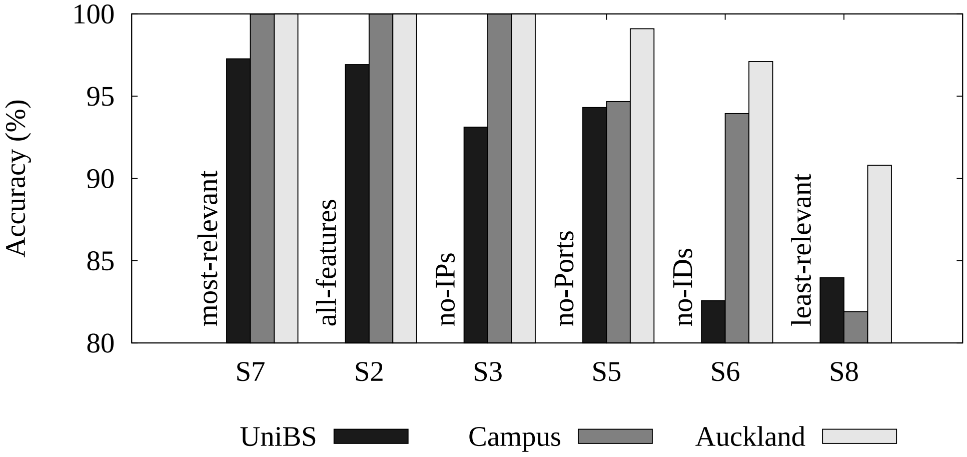


Figure 7.13: Flow classification accuracy for different traces and features sets (homogeneous training, $k = 100$ sampling).

established based on well-known ports, we expect 100% accuracy when this feature is included in the feature set (on which the ground truth for this dataset relies).

The set of most-relevant features is consistently the one yielding the best result, for all datasets. Interestingly, also the set composed by all features achieves nearly the same performance, in contrast with the common belief that classifiers are misled by an excess of information. As a matter of fact, notice that the set of least-relevant features still contains a valuable amount of information as it correctly identifies more than 80% of flows for any trace.

As previously done, we also investigate the importance of IP addresses and transport-layer ports features by evaluating the classification accuracy after their removal. We find that transport layer identifiers are particularly important for classification purposes, hence removing them has a great impact on the accuracy (especially for Auckland as expected). Conversely, IP addresses are particularly relevant only for the UniBS trace, where, as previously observed, the same server IP is very often associated with the same application, thus becoming a good discriminator; while in more diverse traces removing IP addresses from the feature set does not affect the performance at all. Removing both transport and network layer identifiers gives the worst results – for the UniBS even worse than the least-relevant set of features. Moreover, the performance loss may be more pronounced than in the unsampled case shown earlier, meaning that information contained in IP address and ports is even more important under sampling. At the same time though, the removal of IP and ports does not drastically compromise the accuracy in the Auckland case, where the features of the different traffic classes are likely more separated.

## 7.7 Summary

In this chapter we empirically studied the impact of packet sampling on traffic measurement and traffic classification. Sampling is already a very common practice in operational networks and the increasing trend of network traffic is likely to spread its adoption even more among operators. For this reason, we accurately assessed the amount of information lost when applying sampling to traffic characterization, as well as the repercussion of such loss on the performance of different applications of sampling data, in particular of traffic classification.

We processed an extremely heterogeneous dataset composed of four packet traces (representative of different access technologies and operational environments) with a traffic monitoring tool able to extract several traffic features both in aggregated and per-flow fashion. The tool was modified ad hoc to apply different sampling policies and arbitrary sampling rates to the traces. Moreover, in an attempt to foster cross-comparison in the community, we made an effort with respect to both previous literature (i.e., by considering the same features sets of [97]) and future research (i.e., by using open datasets and describing our labeling ground truth). Such data allowed us to conduct an extended experimental campaign with two main objectives: (i) assessing the degradation introduced by different sampling policies and rates in aggregated traffic features, irrespectively of the possible applications (e.g. classification, intrusion detection) of such measurements; (ii) evaluating whether flow-level features derived from sampled data are suitable for statistical traffic classification. In the following, we separately summarize the main contributions.

### 7.7.1 Impact on traffic characterization

Our experiments, which considered four sampling-policies (namely systematic, random, stratified and systematic SYN sampling), about 170 traffic features and two distortion metrics, yielded the following findings.

- Unfortunately most of the features are already distorted at low sampling, regardless of the sampling policy.
- Generally a lower degradation affects features based on the inspection of a single packet (such as those related to IP and UDP) with respect to those depending on the analysis of more packets. An exception is represented by those features relying on the inspection of very specific segments (e.g., some TCP options).
- Regardless of the protocol layer, we isolated a small set of features robust to sampling across all the datasets.
- A sensitivity analysis conducted on this reduced set shows no remarkable advantage of one sampling policy over the others, thus partly contrasting previous studies in favor of random sampling.
- We identify two reasons for the previous finding: the statistical multiplexing may partly eliminate the bias induced by simple strategies (e.g., systematic sampling); second, this evidence may have been hidden by previous work which typically focused on a few specific features only (e.g., traffic volumes).
- We spotted a number of counter-intuitive behaviors and measurement artifacts, showing that it may be challenging to correctly assess the impact of sampling even on simple measures.

### 7.7.2 Impact on traffic classification

Regarding traffic classification, we specifically focused on a biased, yet practical, sampling policy (namely systematic SYN sampling) which overcomes the problem pointed out in [43, 97] concerning the statistical significance of the results. Before applying the classification based on C4.5 classification trees, we quantified the information conveyed by features about the application label by means of the information gain metric. The main findings of our experiments follow.

- The cross-investigation of the pure feature distortion and its information-gain loss shows a complex non proportional relation. In particular, there are few features whose information gain remains unchanged under sampling – interestingly they almost coincide with those features derived from a single packet and less distorted by sampling.

- Even more unexpected, some features, though heavily distorted, show an high information-gain score, thus proving important discriminators.

- The information-gain ranking depends on the sampling rate applied, thus suggesting the use of a larger features sets for training the classifier which appears only slightly affected by overfitting.

- Coherently with [43], we show that even in our larger dataset a homogeneous training (i.e., where the same sampling rate has been applied to both training and validation traffic) yields extremely good results, even for harsh sampling (e.g. 1 out of 100 sampling).

- If on the one hand the former observation implies that different training sets should be kept for each sampling rates, on the other hand the classification accuracy degrades gracefully for intermediate heterogeneous solutions (i.e., training on sampled data, but with a different sampling rate). Therefore, good results might be achieved by employing a few training sets obtained with carefully chosen, representative sampling rates.

# Part III

# Congestion control for P2P

# Chapter 8

# A measurement study of LEDBAT, the new BitTorrent congestion control algorithm

## 8.1 Introduction

In this part, we change perspective in perceiving our goal of making life easier for ISPs by helping them manage P2P traffic better. While in the previous part we focused on providing tools to enable an effective as well as efficient identification of such traffic, first mandatory step to implement special policy for P2P traffic in the network core, in the following chapter we rather propose to change this traffic altogether. The subject of our study is LEDBAT , which stands for Low Extra Delay BAckground Transport procols, a new lower-than-best effort congestion control algorithm proposed and implemented on top of UDP by the popular file-sharing application BitTorrent .

Though regarded with skepticism in the first place [31, 134] because implemented on top of UDP which was associated right away with unresponsive source, LEDBAT has actually very sound goals: (i) it aims at saturating the available bandwidth, (ii) while adding only a small delay on the forward path (to avoid bothering interactive traffic), and (iii) quickly yield to other traffic (i.e., to more than TCP friendly). To achieve this goals, LEDBAT implements a delay-based congestion control algorithm, constantly monitoring the additional delay on the forward path, modulating the congestion window (and the throughput) by means of linear PID controller using the delay measure as input signal (see Chap. 9 for a detailed explanation of the algorithm). With such a proposal BitTorrent wanted to make an effort to collaborate with ISPs and make its traffic gentler towards both the network and other applications, to put an end to the struggle with ISPs which had brought one of the biggest American operators, Comcast, to throttle file-sharing traffic[21].

The adoption of this protocol by such a popular application and the successive standardization process undergone by LEDBAT within the IETF [166] make it quite a relevant subject of research, especially as it pursue the same goals of this thesis. Moreover, independent implementation and evaluation of protocols and technologies deployed in the Internet are actually extremely important and even required for a new protocol to become an integral part of the network architecture. Having started our research on LEDBAT early from its announcement in December 2008 [134], even before its specification were made public, we employed different methodologies to tackle this study. In this first chapter, whose results have been published in [156], we employ a black-box approach and by means of testbed experiments using the actual BitTorrent implementation, we show the evolution of the protocol across successive releases gathering a first picture of the protocol

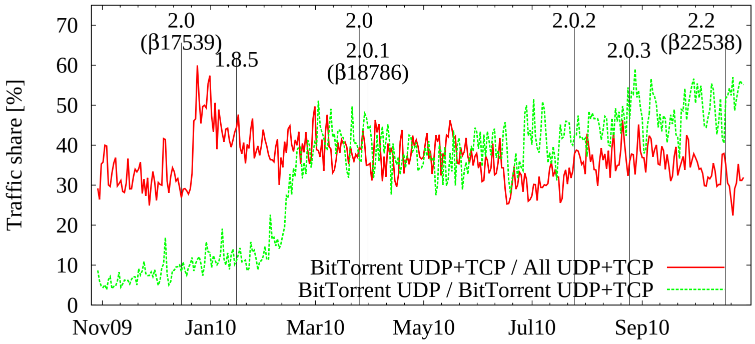


Figure 8.1: Proportion of BitTorrent traffic and of BitTorrent Ledbat in the wild.

merits and issues. In the following chapters instead, we take advantage of the LEDBAT draft, we implement it in the packet-level simulator `ns2` [15] and we carry on a simulative and analytical study of the protocol, proposing a modification of the original design to correct its flaws.

Before delving into our experiments, we would like to report on the actual diffusion of LEDBAT in real networks, to highlight the importance of this work. Although recent research shows an increasing importance of video over the share of Internet traffic [169], BitTorrent still represents a significant portion of user generated data. Moreover, as BitTorrent 3.0 is pushing a new live streaming feature, we may expect that it will still be present in the landscape of Internet popular application even considering the increasing thirst for video streaming content [169]. In Fig. 8.1, we depict the BitTorrent traffic share (UDP and TCP traffic, over all traffic) measured for about one year at a PoP at a large European ISP network that we continuously monitor [115]. The picture also shows the relative percentage of BitTorrent traffic carried over UDP (hence, over the LEDBAT transport protocol), normalized over the total amount of BitTorrent traffic. Labels report a few BitTorrent application release over the considered period[1]. The picture clearly shows, soon after the release of uTorrent 2.0-$\beta$17539, which first introduced data transport over LEDBAT *by default*, a steep increase of BitTorrent traffic volume, followed by an increase of the percentage of BitTorrent traffic carried over UDP. The percentage of BitTorrent appears slightly eroded by other traffic in the subsequent period, due to file-hosting (e.g., MegaUpload, RapidShare, etc.) and VoD traffic (e.g., YouTube, Dailymotion), yet the percentage of UDP BitTorrent traffic is instead slowly increasing (which may reflect adoption of newer release of the protocol by the user population), and finally exceeds 50% of the BitTorrent traffic volume. We also that recent work [76] has showed that the decreasing trend of P2P traffic has reverted to increase in late 2010. Such a large diffusion clearly motivates our study of the protocol, to verify it honors its goals and to check that no flaw affects its design as this might possibly cause severe harm to the network.

The aim of this chapter is twofold. On the one hand, we target at understanding the performance of LEDBAT in a number of simple single flow scenarios, considering multiple versions of the official client so to better clutch its evolution. On the other hand, by means of multiple flows scenarios, we aim at gathering a preliminary understanding of the implication that a widespread adoption of LEDBAT could have on the current Internet landscape. We find active testbed experimentation extremely useful for several reasons. First, the BitTorrent implementation of the LEDBAT protocol may differ from any draft-compliant implementation by some design choices or parameter setting, that may have a deep impact on the protocol performance. Second, the most

[1] See "Announcement" thread from the uTorrent forum `http://forum.utorrent.com/viewforum.php?id=4`

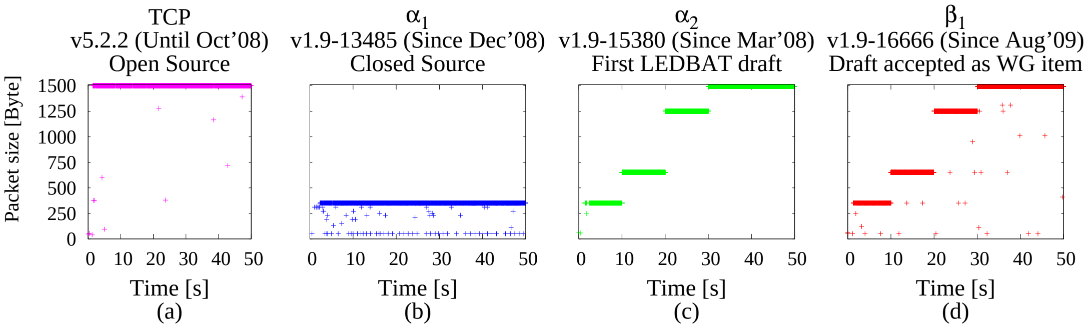


Figure 8.2: The last few months of BitTorrent client evolution: Temporal plot of packet-level traces for different BitTorrent flavors, reporting packet size during the first minute of the transfer

widespread LEDBAT implementation on the Internet will be the official BitTorrent version, rather than a legacy implementation, which motivates a direct evaluation of this client. Third the analysis of proprietary applications by independent observers has the benefit of shedding light on the protocol inner workings. Finally real-world dynamics introduced by network devices are often much more complex than the synthetic ones that a simulation environment, although accurate, can reproduce.

This chapter is organized as follows. We first describe the testbed and methodology used to obtain our results in Sec. 8.2. In we present the results of our experiments in single flow-scenarios, while multiple-flow settings are taken into account in Sec. 8.4.

## 8.2 Methodology and Preliminary Insights

For the investigation of the LEDBAT , we adopt an active-measurements black-box experimental approach, consisting in the analysis of the traffic generated by the BitTorrent client on different network scenarios. We run several versions of the new BitTorrent client on PCs equipped with dual-core processors featuring (i) unless otherwise stated, native installations of Windows XP or (ii) BitTorrent clients running on Linux using the `wine` Windows emulator. PCs are either (i) connected to the Internet through ISPs offering ADSL access, or (ii) in a local LAN testbed via Ethernet cards. In the first case we leave the default modem settings unchanged, while in the second one we disable the interrupt coalescing feature and avoid using jumbo frames. Moreover in the LAN testbed, the traffic is routed through a middlebox running a 2.6.28 Linux kernel, which acts also as network emulator by means of `netem`, in order to enforce artificial network conditions.

As formerly stated, in our experiments we consider both single flow and multiple flows scenarios. Single flow experiments are useful to understand the protocol performance under a range of different network conditions, while multiple flows experiments are needed to quantify the level of inter-protocol priority (e.g., with respect to TCP flows) and intra-protocol fairness (e.g., with respect to other LEDBAT flows) achieved by the distributed control algorithm. Under the classic BitTorrent terminology, every LEDBAT sender-receiver pair is a seeder-leecher pair, so that data transfer happens in a single direction. In case of multiple-flows experiments, every pair of actors belongs to a different torrent, so that no data exchange happens between different leechers.

We start by providing some insights on the BitTorrent evolution with the help of Fig. 8.2. Every picture refers to a different experiment, of which we report the first minute, corresponding to a different BitTorrent flavor. The seeder connects to the middlebox with a 100 Mbps Ethernet link, while between the middlebox and the leecher there is a 10 Mbps Ethernet bottleneck link. No other traffic is present on the bottleneck, and the one-way delay on the forward path is forced to 50 ms, to loosely emulate a scenario where two faraway peers with high speed Internet access (e.g., ADSL2+, FTTH or Ethernet) are connected together.

Pictures are arranged so that the macroscopic timescale of BitTorrent evolution also grows from left to right: Fig. 8.2-(a) shows, as a reference, the old open-source TCP-based client, while Fig. 8.2-(b) refers to the first closed-source version $\alpha_1$, released December 2008. Then, Fig. 8.2-(c) depicts the $\alpha_2$ version, released roughly at the same time of the first IETF draft [166] in March 2009. Finally, Fig. 8.2-(d) refers to the $\beta_1$ version, released after the draft was accepted as an official IETF WG item in August 2009.

The comparison of different versions of the protocol yields several interesting observations. First, notice that all versions analyzed correspond to important milestones in the development process of the protocol: thus, they provide a valuable perspective which highlights the flaws as well as the improvements of the subsequent steps of LEDBAT evolution. In particular, the $\alpha_1$ version (which precedes the draft specification and motivates a black-box approach) was particularly instable and soon superseded. Moreover, from this study it emerges that the LEDBAT implementation has been *constantly* evolving, reason why we decided it would be more interesting to experiment and contrast all of them rather than picking one single version.

For each flavor represented in Fig. 8.2, pictures depict the packet size on the y-axis, measured at the sender side, with time of the experiment running on the x-axis. As it can be seen, the application-layer segmentation policy is remarkably variable across different LEDBAT flavors. In contrast with TCP, which always transmits segments of maximum size, LEDBAT instead uses variable packet sizes. For instance, the $\alpha_1$ implementation of Fig. 8.2-(b) mostly used small segments of about 350 bytes, transmitted at very high rate. Although this allows a finer tuning of the congestion window size, (e.g., likely to be more reactive to network condition), it definitively results in an unnecessary overhead. This segmentation policy is a bad choice for large transfers, and was indeed soon dropped in favor of larger segment sizes. As can be gathered from Fig. 8.2-(c) and Fig. 8.2-(d), newer BitTorrent flavors start by segmenting data in small-size segments, and then gradually increase the segment size over time, rarely changing it once the full-payload segment size is reached. In case of $\alpha_2$ flavor, we observe subsequent phases, about 10-seconds long, where only a single segment size is used: it takes about 40 seconds to the application-layer segmentation policy to settle to full-payload segment size. The $\beta_1$ flavor behaves similarly, although a wider range of segment sizes is employed during the whole experiment, probably to obtain a finer byte-wise control of the congestion window.

The corresponding time evolution of the achieved throughput, measured over 1 s time-windows is depicted in Fig. 8.3-(a), using a longer timeframe of about 4 minutes. We merely superpose the curves for the sake of comparison, but experiments have been independently performed. It can be seen that, shortly after achieving a sustained throughput of about 9 Mbps during about 50 seconds, the sending rate of the $\alpha_1$version suddenly drops, and about 2 minutes are necessary to recover from this starvation (this unstable behavior was observed under a wide range of conditions). In contrast, $\alpha_2$ and $\beta_1$ achieve a lower but steady throughput, slightly above 4 and 7 Mbps respectively.

As a reference, we also report the throughput of a BitTorrent client using TCP running on the native Windows and Linux networking stacks under their default settings. The networking stack implementation and configuration dramatically impacts the protocol performance also in the TCP case. As reported in [48], in Windows XP, for transmission rates between 10-100 Mbps the default receive window is set to 17520 Bytes, whereas the default value of the Linux receive window (set

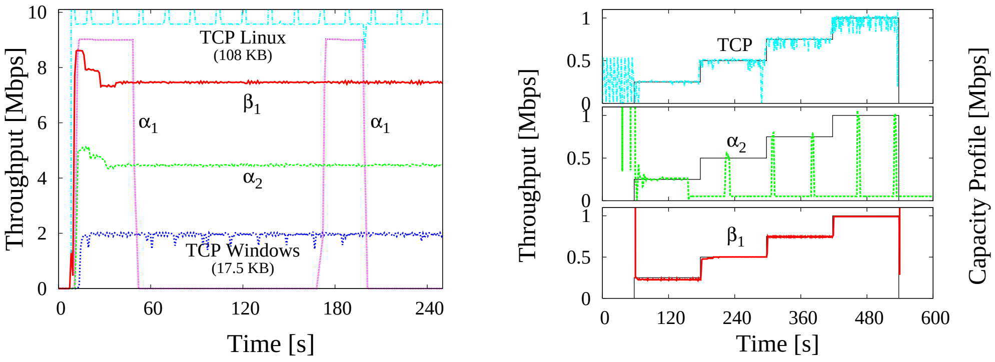


Figure 8.3: Throughput for different flavors (a) without and (b) with bottleneck capacity limitations.

in `net.ipv4.tcp_mem`) is about 6 times larger. Notice that in the Windows XP case, due to the 50 ms delay, the default value of the maximum window is not large enough to allow full saturation of the bottleneck pipe. This is an important, though not novel, observation on which we will come back later on Sec. 8.4.

## 8.3 Single-flow scenarios

Let us start by testing how BitTorrent copes with changing bottleneck capacity. We use a setup similar to the former experiment, but in this case the capacity of the link between the middlebox and the leecher is limited by means of the Hierarchical Token Bucket (HTB), available in `netem`. In more detail, we start at t=60 s to let LEDBAT throughput settle to a steady state, and then we turn on the HTB shaper. We initially tune it to 250 Kbps, increasing then the available capacity in steps of 250 Kbps every 2 minutes, as shown by the solid line capacity profile in Fig. 8.3-(b). A decreasing capacity profile yields to similar results and is thus not shown in the figure.

Time evolution of the throughput is reported for the new $\alpha_2$, $\beta_1$ flavors as well as for the old TCP client. Flavor $\alpha_2$ proves to be unable to quickly adapt to the changing link rate: it periodically enters a probing (or slow-start) phase, where it likely tries to infer network conditions by varying the segment size and sending rate. However, this phase is apparently unsuccessful and $\alpha_2$ throughput starves (we did not observe such a starvation phenomenon for bottleneck larger than 1000 Kbps). This bug has been fixed by later releases: $\beta_1$ matches the available bandwidth, and moreover LEDBAT shows a much smoother curve than TCP. In this case, we may say that one of the LEDBAT design goals, namely, to efficiently exploit the available capacity, seems to be perfectly achieved.

Then, consider that the LEDBAT congestion control is based on a linear adaptation (i.e., growth/shrink) of the sender window to variations in the queuing delay on the forward data path (i.e., as inferred by the decrease/increase of the one-way delay, with respect to the minimum measured one as reference): it is thus critical to assess its reaction to the measured one-way (OWD) delay. However, the sender response to queuing delay variations is nevertheless based on a closed-loop reaction with the receiver: therefore, we argue that the time instants at which the sender window growth/shrink decisions will be taken are also affected by the two-way delay, or Round Trip Time (RTT).

Thus, we setup and experiment in which we add an incremental OWD on either the forward (data) or backward (acknowledgement) paths. As before, after LEDBAT settles we increase the

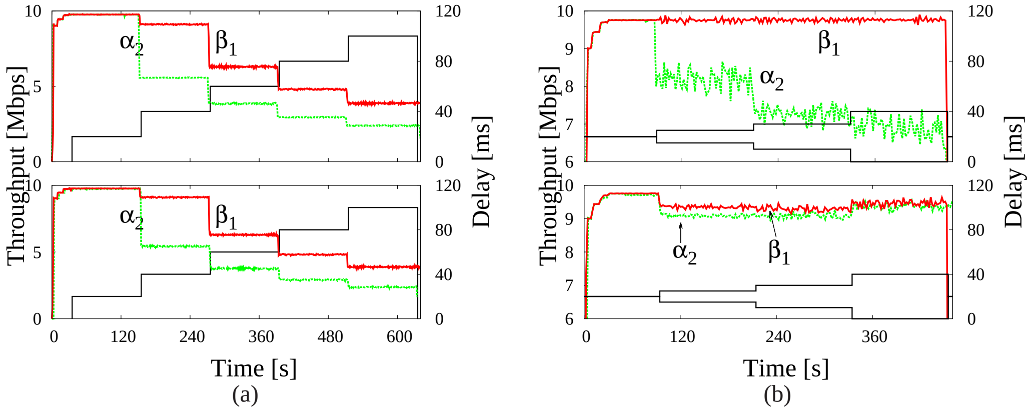


Figure 8.4: Throughput evolution for different delay settings on the forward (top) and backward (bottom) path: (a) average delay increases over time, delay is equal for all packets (b) average delay is constant over time, delay variance increases over time.

additional delay in steps of 20 ms every 2 minutes, for an RTT spanning on the 20–100 ms range as shown by the stepwise profile in Fig. 8.4-(a). The amount of OWD delay is added either to the forward path (top) or backward (bottom) path: in the former case, the delay incrementally adds to the OWD estimation performed by the sender so that it may directly affect the congestion control loop, while in the latter case it only delays the acknowledgement and may only indirectly affect the control loop.

As it can be seen from the comparison of the top and bottom plots of Fig. 8.4-(a), the overall effect on performance is the same: BitTorrent throughput decreases for increasing RTT, which is due to an upper bound of the receiver window (analogously to what seen before for TCP). With some back-of-the-envelope calculation based on the experimental results shown in Fig. 8.4-(a), one can gather that the receiver window limit has been increased from 20 full-payload segments of $\alpha_2$ to 30 full-payload segment of $\beta_1$. While the picture shows that this limit may not be enough to fully utilize the link capacity (e.g., $\beta_1$ achieves about 4 Mbps throughput on a 10 Mbps link with RTT$=$100 ms), in practice it is not a severe constraint, as the capacity will likely be shared across several flows established with multiple peers of a BitTorrent swarm (or the receiver window limit could be increased).

In Fig. 8.4-(b) we instead investigate the effects of a variable OWD delay, that changes for each packet uniformly at random, with average OWD equal to 20 ms. In this case we keep the average constant but increase the delay *variance* every 2 minutes, so that the profile reports the minimum and maximum delays of the uniform distribution. The variable delay also implies that packet order is not guaranteed, because packets encountering a larger delay will be received later and thus out-of-order. Again, delay variance is enforced on either the forward (top) or backward (bottom) path. As it can be expected, LEDBAT is rather robust to a variable jitter on the backward path, where we observe only a minimal throughput reduction. Conversely, variance in the forward path has a much more pronounced performance impact: interestingly, $\alpha_2$ throughput significantly drops, whereas $\beta_1$ performance is practically unchanged. This probably hints to the use of a more sophisticated noise filtering algorithm (e.g., that discards delay samples of out-of-order packets), although a more careful analysis is needed to support this assertion.

We finally perform an experiment using PCs connected through ADSL modems to the wild Internet. Thus, in this case we no longer have complete control over the network environment, but

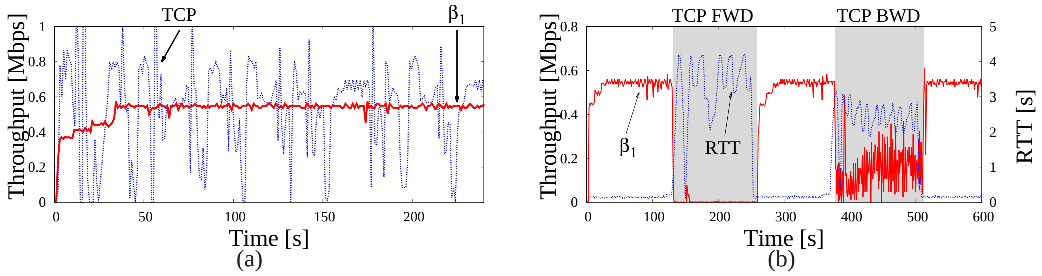


Figure 8.5: Real Internet experiments: (a) different flavors and (b) interfering traffic.

we still can assume that no congestion happens in the network and that the access link constitutes the capacity bottleneck. It can be seen from Fig. 8.5-(a) that in a realistic scenario, when the end-hosts only run LEDBAT , $\beta_1$ achieves a smooth throughput whose absolute value closely matches the nominal ADSL uplink capacity (640 Kbps). In contrast, TCP throughput is more fluctuating due to self-induced congestion, which causes fairly large queues before eventual losses occur. This confirms that the goal of avoiding self-induced congestion at the access is also met.

## 8.4 Multiple Flows

We now explore scenarios with several concurrent flows, starting with the simple one where a single LEDBAT flow interacts with a single TCP flow. Considering two PCs connected through ADSL modems to the wild Internet, Fig. 8.5-(b) reports an experiment where, during a single LEDBAT transfer, we alternate periods in which PCs generate no traffic other than LEDBAT , to periods (i.e., the gray ones) in which we superpose TCP traffic on either the forward or backward path.

The plot reports the time evolution of the LEDBAT throughput as well as the RTT delay measured by ICMP (as a rough estimation of the queue size seen by LEDBAT ). During the silence periods (0–120 s and 240–360 s), as bottleneck is placed at the edge of the network, LEDBAT is able to efficiently exploit the link rate. As soon as a backlogged TCP transfer is started on the forward path (120–240 s), LEDBAT congestion control correctly puts the traffic in low priority. Notice that in this case, ICMP reports that a fairly large queue of TCP data packets builds up in the ADSL line (roughly 4 seconds, corresponding to about 300 KB of buffer space for the nominal ADSL rate). Conversely, whenever the backlogged TCP transfer is started on the backward path (360–480 s), LEDBAT transfer on the forward direction should only be minimally affected by the amount of acknowledgement TCP traffic flowing in the forward direction. However, as it can be seen from Fig. 8.5-(b), the LEDBAT throughput drastically drops, further exhibiting very wide fluctuations (notice also that the ADSL modem buffer space of the receiver appears to be smaller, as the RTT is shorter). Notice that in this case, LEDBAT forward data path shares the link capacity only with TCP acknowledgements, which account for a very low, but likely very bursty, throughput: this may led LEDBAT into a messy queuing delay estimate, and as a result, the uplink capacity of the device is heavily underutilized (about 74% of wasted resources).

We finally perform experiments to analyze the interaction of several flows. In this case, we setup several torrents, one for every different LEDBAT seeder-leecher pair, so that no data exchange happens between leechers of different pairs. Thus, flows are independent at the application layer, though their are dependent at the transport layer, as they share the same physical 10 Mbps RTT=50 ms bottleneck.

Table 8.1: Efficiency and Fairness between multiple TCP and LEDBAT flows

| | | TCP$_W$, LEDBAT $\beta_1$ | | | | | | |
|---|---|---|---|---|---|---|---|---|
| TCP | LEDBAT | $\%_1$ | $\%_2$ | $\%_3$ | $\%_4$ | $\eta$ | Fairness | RTX% |
| 4 | 0 | 0.25 | 0.25 | 0.25 | 0.25 | 0.67 | 1.00 | 5e-4 |
| 3 | 1 | 0.14 | 0.14 | 0.14 | 0.57 | 0.94 | 0.64 | - |
| 2 | 2 | 0.10 | 0.10 | 0.40 | 0.40 | 0.93 | 0.74 | - |
| 1 | 3 | 0.08 | 0.31 | 0.31 | 0.31 | 0.92 | 0.87 | - |
| 0 | 4 | 0.25 | 0.27 | 0.24 | 0.24 | 0.96 | 1.00 | - |

| | | TCP$_L$, LEDBAT $\beta_1$ | | | | | | |
|---|---|---|---|---|---|---|---|---|
| TCP | LEDBAT | $\%_1$ | $\%_2$ | $\%_3$ | $\%_4$ | $\eta$ | Fairness | RTX% |
| 4 | 0 | 0.25 | 0.25 | 0.25 | 0.25 | 0.98 | 1.00 | 0.06 |
| 3 | 1 | 0.35 | 0.32 | 0.32 | 0.00 | 0.98 | 0.75 | 0.14 |
| 2 | 2 | 0.43 | 0.51 | 0.03 | 0.03 | 0.98 | 0.56 | 4e-3 |
| 1 | 3 | 0.87 | 0.04 | 0.04 | 0.05 | 0.98 | 0.33 | - |
| 0 | 4 | 0.25 | 0.27 | 0.24 | 0.24 | 0.96 | 1.00 | - |

We consider a fixed number of F=4 flows, and vary the number of TCP and LEDBAT -$\beta_1$ connections to explore their mutual influence. All flows start at time $t = 0$, experiments last 10 minutes and results refer to the last 9 minutes of the experiment. We generate TCP traffic using Linux (so that we can reliably gather retransmission statistics using `netstat`), setting the congestion control flavor to NewReno. We perform two set of experiments, using either the Windows or Linux defaults values for the maximum receiver windows as early stressed in Fig. 8.3-(a): in our setup, the Windows-like TCP settings (TCP$_W$) are thus less aggressive than Linux ones (TCP$_L$).

For each experiment, we evaluate user-centric performance by means of the breakdown of the resources acquired by each flow, while we express network-centric performance in terms of the link utilization $\eta$. To further quantify the protocol mutual influence, we use the Jain's fairness index of the flows throughput and evaluate the percentage of TCP retransmissions (RTX). Results are reported in Tab. 8.1, with Windows and Linux settings on the left and right respectively. Comparing the two table portions, we argue that the exact meaning of "low-priority" may be fuzzy in the real-world. Indeed, while LEDBAT -$\beta_1$ is lower priority than an "aggressive" TCP, it may be competing more fairly against a more gentle set of parameters, thus being at least as high priority as TCP. In fact while LEDBAT is practically starved by TCP$_L$, LEDBAT is able to achieve a slightly higher priority than TCP$_W$. Although we recognize that results may change using more realistic and heterogeneous network scenarios, or using the real Windows stack instead of simply emulating its settings, we believe that an important point remains open: i.e., the precise meaning of "lower than best effort", as the mutual influence of TCP and LEDBAT traffic may significantly differ depending on the TCP flavor as well.

## 8.5 Related work

Related work to this study can be divided in two cathegories. First, given its widespread diffusion as file-sharing application, BitTorrent has been studied widely studied [35, 37, 93, 145] by means of theoretical analysis, simulations and measurement. Second, there is a large literature on Internet congestion control: given the empirical nature of the results presented in this chapter, we review here a few works which employs a similar measurement based methodology to approach this subject [26, 38, 65], while we refer to the related work section of the next chapter for other works on other lower-than best effort protocols. Third, we report on the few measurement works on LEDBAT appeared so far, because of the novelty of the subject.

Due to BitTorrent very recent evolution, previous work on BitTorrent [35, 37, 93, 145] focused on complementary aspects to those analyzed in this work. In [145] authors develop a fluid model to study the properties of a BitTorrent network: in particular they calculate the average download time of a single file, proving that under some assumptions BitTorrent provides a scalable, stable and efficient service for file-sharing. They also study the dynamics of peer-selection and optimistic unchoking: on one hand, by means of game theory, they prove that the P2P swarm can reach an equilibrium point where each peer contributes with all its bandwidth to the system; on the other hand, they show that free-riding (i.e., peers that only download files without contributing anything to the system) is still possible.

Authors of [93] analyzes the log related of a BitTorrent tracker related to a popular legal torrent. This log covers an extended period of time of five months. BitTorrent proves to be a very scalable system during flash-crowds even in practice, providing good download rates and completion time to peers. However authors highlight again that the system definitely needs peers to contribute their bandwidth as seeder after completing their transfer, otherwise performance suffers.

Simulation has instead been used in [35] to study the utilization of upload capacity of peers and the fairness of transfers. In agreement with the mathematical analysis of [145], authors of [35] shows that the system performance greatly benefits from matching peers with the same bandwidth, increasing fairness and preventing free-riding as well; therefore, they propose a bandwidth aware tracker to improve the system. They also propose a few modifications to seeders to better use their precious upload bandwidth in augmenting the diffusion of rare chunks through the swarm.

Another simulation study is carried in in [37] to assess the performance of a locality-aware peer selection strategy. Authors of this work show that a significant reduction of inter-ISP traffic (and the related cost for operators) can be obtained by biasing the peer-selection scheme towards peers internal to the ISP. Such benefit comes almost at no cost for of system performance, provided that initial seeders have an high upload bandwidth to sustain the initial spreading of content.

Congestion control work closer to our adopts a black-box experimental measurements approach to unveil proprietary algorithms of, e.g., Skype [38, 65] or P2P-TV applications [26]. More precisely, [38] analyzes the reaction of Skype, which implements a VoIP service also based on a P2P overlay, to changing network conditions: authors show that the protocol way of dealing with losses differs from the way of reacting to congestion, at least when employing UDP as a transport protocol. They also employ passive traces to study Skype signaling traffic and users behavior. Conversely in [65], authors study the way Skype, and especially its video traffic, deals with changing network conditions. Especially they study multiple flows-scenarios, involving concurrent Skype flows and TCP flows. Skype confirms its capability of working even in very bad settings, thanks to several advanced mechanisms such as FEC, adaptive codecs, variable packet sizes; however they discover an aggressive behavior towards TCP, given the unresponsiveness of Skype flows to losses. In [26] an approach extremely similar to ours is employed to study of P2P-TV applications adapt to harsh network conditions. Authors enforce network impairments in term of losses, delays, available bandwidth and background traffic by means of a gateway running the `netem` network emulator. P2P-live streaming applications are able of dealing with extremely adverse network conditions, yet they can become extremely aggressive and TCP-unfriendly in some cases, eventually negatively impacting the performance of other applications and of the video-stream itself.

In [59], BitTorrent developers detail a specific aspect of their implementation: namely, an algorithm to solve the problem of the clock drift in LEDBAT, to ameliorate the queuing delay estimation at the sender side. Instead in [29] authors evaluate a Python, user-level implementation of the new protocol in a large testbed: this independent study further confirms the non-intrusiveness of LEDBAT and its lower-priority properties; on the contrary, authors detect a problem in exploiting the available bandwidth, which, however, seems due more to the overhead of their user-level implementation. Finally the work which is the closest to ours is [164], where authors test their own implementation of LEDBAT behind mainstream home gateways. From their analysis, authors found that modern gateways often already implement advanced queue management to prevent elastic traffic from interfering with interactive traffic. The evidence presented in the chapter privileges such a solution to the one provided by LEDBAT as sophisticated queuing disciplines may allow a finer tuning of traffic priorities. Moreover authors highlight that LEDBAT interacts badly with such mechanisms in home gateways, resulting in very poor performance.

## 8.6 Summary

This chapter presented an experimental evaluation of LEDBAT , the novel BitTorrent congestion control protocol. Single-flow experiments in a controlled environment show some of the fallacies of earlier LEDBAT flavors (e.g., instability, small packets overkill, starvation at low throughput, tuning of maximum receiver windows, wrong estimate of one-way delay in case of packet reordering, etc.), that have been addressed by the latest release. Experiments in a real Internet environment, instead, show that, although LEDBAT seems a promising protocol (e.g., achieving a much smoother throughput and keeping thus the delay on the link low), some issues still need to be worked out (e.g., performance in case of reverse path traffic). Finally, multiple-flows experiments show that “low-priority” meaning significantly varies depending on the TCP settings as well.

In the next chapters we abandon the black-box approach followed in this one and we go back to the formal specification of LEDBAT provided by the IETF draft [166]. We will carefully review the design and implement the protocol to extensively study its behavior by means of packet level simulation. With such an analysis we will complement the one presented in this study.

# Chapter 9

# Simulation study of LEDBAT

In this chapter, whose results were published in [46, 155], we address the study of the LEDBAT protocol by means of simulation. In Sec. 9.1, we first review the original LEDBAT design as specified in the related IETF drat, better explaining the protocol objectives as well as the algorithm used to achieve them. Then, by using our own implementation of the protocol for the packet-level simulator `ns2`, in Sec. 9.2 we perform some experiments to assess protocol performance in simple settings. First a few simple scenarios are studied, to understand the basic behavior of LEDBAT when competing with another TCP flow and another low-priority flow. We actually find that LEDBAT is an efficient protocol, correctly implementing a lower-than-best-effort service. Yet, we discover an unfairness problem when two, non synchronized LEDBAT flows share the same bottleneck: a latecomer advantage may arise, causing the first flow to starve, eventually for a long time, thus potentially affecting the performance of the above application. We verify that such an issue does exist also in real-life scenario, by conducting a testbed experiments with the recently released official implementation of the protocol by BitTorrent .

For this reason in , we present four possible solutions to the problem, some of them proposed on the IEFT working group mailing-list. Two of them try to correct the measurement error which seems the main cause of the unfairness. However, as found in earlier research on congestion control algorithms [52] and formally proved later in Chap. 10, the problem is rather due to the form of the controller, and in particular in the addictive decrease component. Therefore in the last two solution we try to reintroduce multiplicative decrease, actually obtaining the best result. Overall, while some solutions are clearly uneffective, others partially relieve the problem: in these cases we perform more complex simulations, with multiple flow-scenarios to tune their parameters. However, in the end, these four algorithms are not completely satisfactory, but were extremely helpful in putting ourselves on the right direction towards a more complete solution, which is presented in the next chapter together with a complete analytical and simulation study of its performance. At the end of this chapter we review related work in Sec. 9.4.

## 9.1 LEDBAT Overview

This section provides a basic overview of the LEDBAT draft [166]. To better understand the motivations behind LEDBAT, let us recall that the standard TCP congestion control needs losses to back off: this means that, under a drop-tail FIFO queuing discipline, TCP necessarily fills the buffer. As uplink devices of low-capacity home access networks can buffer up to hundreds of milliseconds, this may translate into poor performance of interactive applications (e.g., slow Web browsing and bad gaming/VoIP quality).

```
on data_packet @ RX:
   remote_timestamp = data_packet.timestamp
   acknowledgement.delay =
       local_timestamp() - remote_timestamp

on acknowledgement @ TX:
   current_delay = acknowledgement.delay
   base_delay = min(base_delay, current_delay)
   queuing_delay = current_delay - base_delay
   off_target = TARGET - queuing_delay
   cwnd += GAIN * off_target / cwnd
```

Figure 9.1: Pseudocode of the LEDBAT sender and receiver operations.

To avoid this drawback, LEDBAT implements a distributed congestion control mechanism, tailored for the transport of non-interactive traffic with lower than Best Effort (i.e., lower than TCP) priority, whose main design goals are:

- Saturate the bottleneck when no other traffic is present, but quickly yield to TCP and other UDP real-time traffic sharing the same bottleneck queue.
- Keep delay low when no other traffic is present, and add little to the queuing delays induced by TCP traffic.
- Operate well in drop-tail FIFO networks, but use explicit congestion notification (e.g., ECN) where available.

Intuitively, to saturate the bottleneck it is necessary that queue builds up: otherwise, when the queue is empty, at least sometimes no data is being transmitted and the link is under-exploited. At the same time, in order to operate friendly toward interactive applications, the queuing delay needs to be as low as possible: LEDBAT is therefore designed to introduce a non-zero *target* queuing delay.

In order to achieve this goal, LEDBAT follows a simple strategy. First of all, it exploits the ongoing data transfer to measure the one-way delay, from which it derives an estimate of the *queuing delay* on the forward path. Using one-way delay instead of round-trip time has the main advantage of preventing unrelated traffic on the backward path from interfering with data transmission. Second, it employs a *linear controller* to modulate the congestion window, and consequently the sending rate, according to the measured delay. LEDBAT operations can be summarized in the pseudocode in Fig. 9.1[1].

In the following, we first consider the two main components of the LEDBAT algorithm separately, and then we report some further considerations on the TCP-friendliness of the novel protocol.

### 9.1.1 Queuing Delay Estimate

Delay measurements are performed collaboratively by the sender and the receiver. The former puts a timestamp from its local clock in each packet. The latter, instead, calculates the one-way

[1] We report the code from the first draft of the protocol. Later releases have improved a few aspects (for instance to correctly manage intermittent flows) but the substance of the algorithm is practically unchanged for our research purposes.

delay as the difference between its own local clock and the received timestamp, and communicates it back to the sender in the acknowledgements. The sender, besides, maintains a minimum of all observed delays, which represent the *base delay* used in queuing delay estimate.

To explain the rationale behind such technique, let us consider the different components of one-way delay: propagation, transmission, processing and queuing. Neglecting the processing delay, propagation and transmission delays are constant components, while the only variable component is the queuing delay. Intuitively, a packet which finds the queue empty (i.e., zero queuing delay) will accurately estimate the constant portion of the one-way delay (i.e., the sum of propagation and transmission delays). This measure yields a minimum of the delay, that will be stored as a reference: then, the queuing delay can be estimated as the difference between the current and the reference delays.

One-way delay measurements are notoriously difficult, especially for non-synchronized hosts. Yet the *variation* of delay with respect to the base delay, which is actually exploited by LEDBAT, is a much more robust metric. In particular, it does not suffer from timestamp errors such as fixed offsets and skews from the true time. For instance, the sender and receiver offsets could severely affect the absolute one-way delay estimate, but they happily cancel in the arithmetic difference `queuing_delay=current_delay-base_delay` (since both delays correspond in their turn to the difference of the receiver minus the sender delay). Further considerations about clock skew, noise filtering and route changes issues can be found in [166].

### 9.1.2 Controller Dynamics

A *proportional-integral-derivative (PID)* controller governs the dynamic of the congestion window in both the ramp-up and ramp-down phases. The controller continuously adapts the window to the estimated delay, in order to match the target delay. Clearly, when the queuing delay estimate is lower than the target (i.e., `off_target<0`) the sending rate has to increase, so that queuing delay reaches the target. Conversely, when the queuing delay estimate is higher than the target (i.e., `off_target>0`) the controller slows down the sending rate.

In Fig. 9.1 we observe that the controller itself is characterized by two parameters, the `TARGET` delay and the `GAIN` coefficient. The draft states that *"`TARGET` parameter MUST be set to 25 milliseconds*[2] *and `GAIN` MUST be set so that max ramp up rate is the same as for TCP."*. The selection of a constant and moreover specific value for `TARGET` is quite controversial, as it is clear that non-compliant implementation with a larger target delay are advantaged and could introduce severe fairness issues (notice for instance that values in BEP29 [136] are larger than those specified in [166]). Concerning the second parameter, we set it to `GAIN=1/TARGET`, choice that we motivate in the next section[3].

We underline here a nice property of the PID controller: the window growth is directly proportional to the difference between the queuing delay estimate and the target `off_target`. In this way, when the queuing delay is close to the target, the controller response will be near zero, thus avoiding undesirable oscillations. Conversely, when the estimation is far from the target, the controller will increase the window faster and hopefully converge earlier.

[2] Recently, the last revisions of the draft have relaxed this condition and require TARGET to be "100 milliseconds or less".

[3] Since version 5 of the draft, the offset target is already normalized over `TARGET` and the draft imposes `GAIN` to be lesser or equal to 1.

### 9.1.3 TCP Friendliness Consideration

An important goal of LEDBAT concerns its ability to yield to TCP traffic when sharing the same bottleneck resources. LEDBAT should be able both to detect the traffic already present on links, and to yield quickly to newly incoming connections.

At the same time, LEDBAT must avoid starvation. In fact if LEDBAT always yielded to any kind of traffic, even to the one generated by non interactive application (e.g., a long-lived FTP transfer), the performance degradation perceived by users may convince them to simply revert to TCP-based transfers, regardless of LEDBAT potential advantages.

A first necessary condition for TCP friendliness, is that LEDBAT *should never ramp-up faster than TCP*. Since LEDBAT increases its congestion window of the largest amount when the delay estimate is zero (notice also that estimated delay can never be negative), by selecting `GAIN=1/TARGET` we guarantee that LEDBAT never ramps-up faster than TCP, as its maximum ramp-up speed is limited to one packet per RTT (i.e. like TCP in congestion avoidance).

A second requirement is that the delay-based LEDBAT congestion controller *should react earlier than loss-based TCP controller*: intuitively, if the former can ramp-down faster than loss-based connections ramp-up, it will yield to the latter. The draft states that LEDBAT should *"yield at precisely the same rate as TCP is ramping-up when the queuing delay is double the target"*. Again our choice of `GAIN=1/TARGET` fulfills this requirement: in fact, when the queuing delay is twice the target, LEDBAT will ramp-down at a rate equal to one packet per RTT, matching thus TCP congestion avoidance ramp-up speed.

A third final condition is that, *in case of loss, LEDBAT should behave like TCP does* (i.e., halve its congestion window). From all these considerations, one can derive that LEDBAT design follows a quite conservative approach, as in the worst case (when the queue estimation always equals zero) its most aggressive behavior simply degenerates into TCP.

## 9.2 Simulation results

In this section, we report results gathered with our implementation of the LEDBAT controller in the Network Simulator `ns2`: we start by illustrating some telling examples of the LEDBAT dynamics in simple cases, incrementally adding complexity to refine the picture later on.

### 9.2.1 Implementation details

To avoid dealing with the complexity of retransmission in case of loss, we implement our LEDBAT controller as a novel flavor of TCP, of which we change the congestion control mechanism. More precisely, we turn off all TCP feature (e.g., FastRetransmit), leaving only the congestion control algorithm early described in Sec. 9.1. For timestamping purposes, we exploit the TCP timestamping option[95].

We implement all mandatory as well as optional features of LEDBAT[166]. More precisely, we implement a cache of queuing delay minima, mandatory to cope with route changes on long timescales. As far as the optional slow-start phase is concerned, since the LEDBAT draft lacks its description [166], we adopt the standard TCP mechanism. However, unless otherwise stated, slow-start mechanism is turned off. Also, though this issue is not treated in [166], our LEDBAT implementation can work in batch-mode (i.e., all packets of a window are possibly sent out in bursts) or paced-mode (i.e., delaying the packet transmission so that packets are spaced equally during the RTT). Unless otherwise stated, packet pacing is turned on.

Then, notice that reducing the sending window to 0 constitutes a problem, since the linear controller will no longer be able to get one way delay estimates – thus, it will not be able to ever increase its sending window again. Therefore, we set a congestion window minimum of 1 packet per RTT, although this is not explicitly specified in [166].

Finally, we point out that we built LEDBAT using the `tcp-linux` module, which allows to bridge real Linux code directly into the simulator. As a non-negligible side-advantage, the implementation is then available as a kernel module offering a novel transport-layer protocol that can be used by (unmodified) real applications.

### 9.2.2 Reference scenario

As reference scenario, we consider a bottleneck link of capacity $C$ Mbps and buffer size $B$ packets. For the sake of simplicity, we assume that all transceivers adopt $P = 1500$ Bytes fixed-size packets. Traffic flows in a single direction, and acks are not delayed, dropped nor affected by cross-traffic on their return path. All flows have the same round trip time $RTT = 50\, ms$, half of which is due to the propagation and transmission delay components of the bottleneck link (i.e., a one-way base delay of 25 ms).

In this chapter we restrict our attention to a simple high-speed access scenarios, with a link of $C = 10\, Mbps$ capacity for downlink/uplink (an extended set of simulations, including an ADSL-like case is available in [157]), and different buffer sizes $B \in [10, 100] \subset \mathbb{N}$ packets. Notice that, once fixed the link capacity $C$ and the packet size $P$, we can express the queuing delay `TARGET` in terms of either a time-lapse or bytes (and packets). Denoting for short the `TARGET` as $\tau$, in the following we will refer indifferently to the queuing delay in terms of time-lapse $\tau_T = 25$ ms or packets $\tau_P = \tau_T C/8P$ (with capacity expressed in kbps and packet size in bytes). For instance in our high-speed scenario, $\tau_T = 25$ ms corresponds to $\tau_P = 20.8$ packets. Thus, a buffer size of $B = 40$ packets, almost equal to the bandwidth-delay product, can accommodate twice as much queuing delay than the LEDBAT target $\tau$.

As performance metrics, we consider the *fairness* and *efficiency* of the data transfer. For the former, we use Jain's fairness index $F$, which is defined as:

$$F = \frac{(\sum_{i=1}^{N} x_i)^2}{N \cdot \sum_{i=1}^{N} x_i^2} \tag{9.1}$$

where $\{x_i\}_{i=1}^{N}$ is the set of rates achieved by $N$ flows sharing the same bottleneck resource. This index ranges between as maximum value of 1 (when the bandwidth is perfectly shared among the $N$ flows) and a minimum of $1/N$ (in case one flow takes all the resource, leaving the others in starvation). Being LEDBAT a *lower* than best-effort protocol, we expect $F < 1$ when it competes with TCP, but $F \simeq 1$ when LEDBAT flows share the same bottleneck. Regarding efficiency, we consider the *link utilization* $\eta$ metric, defined as the ratio of the overall link throughput (including headers) over the link capacity $C$.

### 9.2.3 Homogeneous Initial Conditions

Our investigation starts by considering a LEDBAT flow competing for the same bottleneck resources with either (i) a TCP or (ii) another LEDBAT flow. For the time being, we disable slow-start in both implementation as we are interested in the interaction of the LEDBAT PID the TCP AIMD controllers. We let both flows start at $t = 0$, when the queue is empty and no other traffic is present on the link, so that LEDBAT is able to accurately measure the base delay.

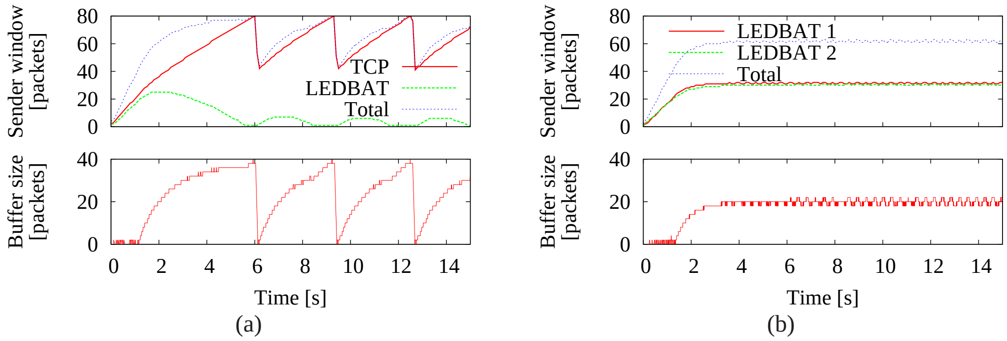


Figure 9.2: Temporal evolution of the sender window (top) and of the queue size (bottom) for TCP-LEDBAT (a) and LEDBAT-LEDBAT interaction (b)

Fig. 9.2-(a) shows the temporal evolution of the LEDBAT and TCP windows (top) as well as of the queue length (bottom), with a buffer size of $B = 40$ packets. We recognize the usual TCP sawtooth behavior, which defines a number of cycles. During the initial ramp-up ($t < 2$ s), LEDBAT and TCP windows grow *nearly* at the same speed of one packet per RTT. LEDBAT grows at its maximum speed because the available link capacity keeps the queue empty. As soon as queue builds up, the LEDBAT linear controller reacts accordingly by slowing down the increase of its sending rate, while TCP behavior remains instead unaltered. Soon after $t = 2$ s, LEDBAT hits the $\tau_P = 20.8$ packet target, and halts the window growth, so presenting a flat sender window curve. TCP, instead, continues its additive increase, so that the queue keeps building up until the queuing delay exceeds the target: the LEDBAT controller, unlike TCP, reacts by decreasing its sending rate, finally reaching the minimum rate of one packet per RTT just before $t = 6$ s.

Slightly afterwards, TCP causes a buffer overflow: consequently, TCP abruptly decreases its sending rate by halving is congestion window. The capacity drains the queue empty, giving thus start to a new cycle. In fact, LEDBAT detects the delay reduction and reacts by opening its window again. However, in this cycle TCP has an initial window size of about 40 packets, which means that it can create queuing sooner with respect to the previous cycle. Therefore, LEDBAT window growth is slower, the TARGET delay is hit earlier (at about $t = 7$ s) and also the window shrink phase appears much shorter. When TCP is again the sole sender on the link, it increases its sending rate until a new loss happens, which in turn triggers the start of a new cycle.

Fig. 9.2-(a) confirms that, as LEDBAT reacts to congestion *earlier* than TCP by estimating the queuing delay, it is able to yield to TCP, which can *work undisturbed*. In fact, losses are due to the normal AIMD dynamic of TCP rather than to the LEDBAT-TCP interaction. Fairness in this case equals $F = 0.65$, with TCP transferring 6 times as much data with respect to LEDBAT during the same timeframe. Fig. 9.2-(a) also reports the sum of both TCP and LEDBAT sender windows, which represents an estimate of the instantaneous link utilization. When TCP and LEDBAT coexist on the link, its *utilization increases* with respect to the case where TCP is alone – in the figure utilization increases by 16%, compared to the case where TCP is alone on the bottleneck.

Fig. 9.2-(b) shows a similar experiment, in which two LEDBAT sources start competing at $t = 0$ for the bottleneck resources. In this case, both senders employ a linear controller and are able to share resources fairly ($F > 0.99$) and efficiently (efficiency is only 0.7% less than in the Fig. 9.2-(a) case). As expected, once the delay target is reached, LEDBAT sources settle (since the offset from the target is zero, and so the controller response). Notice also that, since the two sources started together, they measured the same base delay at $t = 0$. Therefore, each sender independently settles when measuring a queuing delay equal to the target, thus it is actually responsible only for half of buffer occupancy.

### 9.2.4 Heterogeneous Initial Conditions

In this section we consider different start times for different sources. This implies that each sender will measure a different base delay at startup, gathering also a different estimate of the queuing delay. Indeed, assume that the first flow starts at time $t_1 = 0$, while the second starts at time $t_2 = t_1 + \Delta T$. In case the queuing delay at $t_2$ is not zero but equal to $t_Q(t_2)$, the second source will over-estimate the base delay $t_B(t_2)$ with respect to the one measured by the first source as $t_B(t_2) = t_B(t_1) + t_Q(t_2)$. So, the second source will set its target to a value higher than the first one, increasing the chances of a buffer overflow.

In case of interaction between LEDBAT and TCP, heterogeneity of initial conditions has a negligible impact. To convince of this, consider that, whenever LEDBAT starts first, it will be able to correctly estimate the base delay, and then to yield to TCP. Conversely if the LEDBAT flow starts later at $t_2$, it will over-estimate the base delay by the amount of TCP packets present in the buffer. This will in turn make LEDBAT under-estimate the queuing delay, resulting in an increased sending rate which will *anticipate* the first loss cycle. The system later evolves in a way similar to Fig. 9.2-(a), since after TCP halves its window, the capacity drains the queue empty and LEDBAT corrects its wrong base delay estimate. In subsequent cycles, LEDBAT will then dutifully yield to TCP.

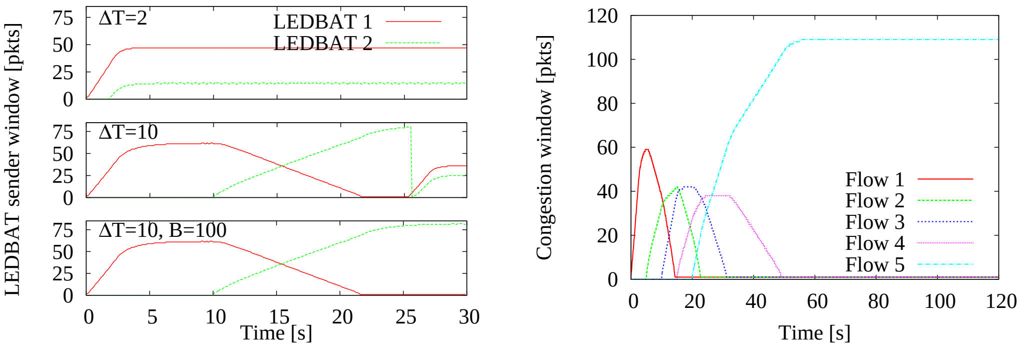
By means of Fig. 9.3-(a), we show, instead, that the interaction among LEDBAT flows is heavily influenced by the buffer size $B$ and the start time gap $\Delta T$. Each graph reports the sender window of two competing LEDBAT flows. In the top plot, obtained for $(\Delta T, B) = (2, 40)$, the second flows activates before the first one has started to create queuing. So, the two flows measure the same base delay and set the same target, which they together reach soon. But the first flow, having started before, attains a larger congestion windows, and actually owns the biggest share of the queue.

Instead, extremely different dynamics can be observed for $(\Delta T, B) = (10, 40)$ in the middle plot. In this case the second flow starts later enough to allow the first one to create some queuing delay, in particular a delay $\tau_T$ equal to its target. For this reason, the second flow wrongly senses a base delay equal to $\tau_T$, and consequently sets its target to twice this value. Therefore, the newcomer starts increasing its rate right away, while the first one senses a growing queuing delay and begins to slowdown until, slightly after $t = 20\,$s, it finally reaches the minimum rate.

Afterwards, dynamics depend on the specific buffer size. The middle plot shows a case where the buffer cannot accommodate the target queuing delay of the second flow (as $B$=$40 < 2\tau_P$=$41.6$). In fact, around $t = 25\,$s, the second flow causes a loss on the bottleneck link and consequently drops its sending rate. Afterwards, similarly to the TCP case, the capacity drains the queue empty, providing the second flows the chance to correct its wrong base delay estimation. Subsequently, flows appear to share much more fairly the bottleneck capacity.

The bottom plot, depicts instead the effect of a larger $B = 100$ buffer, able to absorb the extra delay of the second flow. Basically, since no loss occurs, the second flows reaches its target and then settles, leaving the first flow in starvation. Unfortunately this unfair state persists for a possibly long time (namely, due to route changes considerations, the draft [166] imposes a reset of the base delay every 2-10 minutes).

Finally in Fig. 9.3-(b) it can be seen that, provided that the buffer is large enough, the latecomer advantage even in a multi-flow scenario. It can be seen in the picture that the last arrived flows



(a) (b)

Figure 9.3: LEDBAT vs LEDBAT: Time evolution of congestion window for different initial condition and latecomer advantage phenomenon, even in a multiflow scenario.

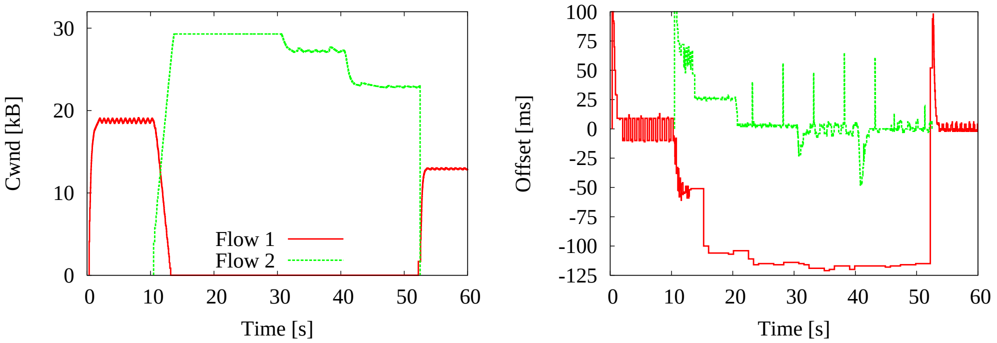


Figure 9.4: Experimental LAN testbed: Congestion window evolution (top) and offset from the target (bottom) for two competing backlogged libUTP flows.

constantly overestimates the base delay with the respect to the previous one, thus starving all the flows arrived before it. For an application like BitTorrent that involves several concurrent flows this might be a common situation.

### 9.2.5 Latecomer advantage in real networks

To further highlight the relevance of this unfairness issue, we point out that the latecomer unfairness unveiled by simulation, also holds in practice, possibly leading to severe flow starvation. We show this by performing testbed experiments of the recently released BitTorrent open-source LEDBAT library [85] (named `libUTP`). We recreate the same conditions as in the simulation scenarios: we consider two PCs connected by a $C = 10$ Mbps Ethernet bottleneck, where we emulate by means of `netem` [87] a $RTT = 50$ ms delay. The first flow starts at time $t = 0$ while we let the latecomer starts at $t = 10$ s. Backlogged transfers are started using the source code provided in [85], instrumented to produce detailed application-level logs. Packet level traces are also captured and post-processed as we did in Chap. 8 for cross-checking purposes: the results are in agreement with the application logs.

Results of the experiment are shown in Fig. 9.4, whose top portion reports the time evolution of the congestion window of the two flows. As soon as the first flow starts, it increases its congestion window until the target is reached, and then settles. However, when the latecomer kicks in at

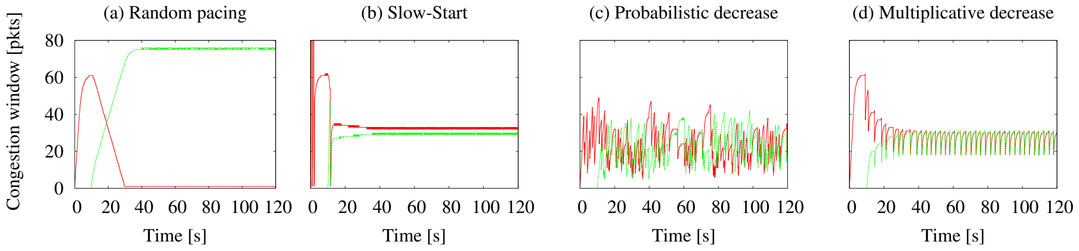


Figure 9.5: Effect of the solutions proposed in the two flows scenario.

$t = 10$ s, the congestion window of the first-comer drops until starvation. The situation persists until $t = 50$ s, time at which we stop the latecomer transfer: right after, the first-comer opens its congestion window again, saturating the link.

## 9.3 Addressing the latecomer advantage

In this section, we propose a few modifications to the LEDBAT protocol to overcome the unfairness issue, which we test by means of simulations. The first group of solutions only tries to *ameliorate the base delay measurement*. First, as suggested in the LEDBAT WG [9], we implement *random pacing* of packets belonging to the same window: this should allow flows to gather different delay samples and possibly converge to a similar view of the base delay. Second, we propose to use *TCP's slow-start* at the very beginning of LEDBAT flows: by filling the buffer, slow-start likely induces losses on already present flows, which drain the queue empty and leave a chance for newcomers to gather a correct measure of the base delay. The second group of solutions instead *addresses the window decrease decisions*, which represent a more fundamental issue. As third solution, we thus suggest introducing (infrequent) *random drops* of LEDBAT sender window, as a means to break unfair states and to de-correlate flow decisions. Fourth, we propose to replace the LEDBAT additive decrease with a *multiplicative decrease*: we indeed expect the abrupt reduction of the throughput of flows to empty the buffer and again allow latecomers to measure the real base-delay.

In the following, we consider each technique on its own, investigating deeper those who actually provide some kind of solution to the problem. However to allow an intuitive comparison we report in Fig. 9.5 four picture which show own each solution behaves in our reference scenario of two flows, where the latecomer advantage is known to show up. It can be seen that the first solution is uneffective, while the last three restore the fairness between the two flows. In the following we evaluate each proposal by means of the same metric used before: the Jain's fairness index and the network efficiency.

### 9.3.1 Correcting the measurement error

#### 9.3.1.1 Random pacing

The fairness issue was early identified during the definition of the LEDBAT protocol [9] and was later confirmed by our preliminary simulation studies [155]. Still, according to some of the participants to the draft definition, the randomness present in real networks would somehow prevent the latecomer advantage from showing up: they argue that random delays caused by OSes, routers and background traffic are enough to avoid the queue becoming so stable and the consequent flow synchronization. However, relying on external network conditions to ensure that the protocol actually works is not a good engineering practice. For this reasons, it seemed much more robust to incorporate some randomness in the protocol itself, more specifically to add a random jitter to packet transmission time. In this way the queue is expected to show a much more varying dynamic, thus allowing flows to gather different estimates of the queuing delay, eventually converging to a fair share of the resource.

Since this was officially discussed in the LEDBAT working group, we analyze it as a first solution. We add to our implementation a random pacing module, which randomly spaces the transmission time of packets belonging to a congestion window in the RTT. Each packet is delayed by a random, uniformly chosen interval of time, taking care of avoiding packets reordering.

Fig. 9.5-(a) shows the case of two flows sharing the bottleneck of our reference scenario and implementing random pacing. Unfortunately, only some minor modifications of flow behavior can be observed with respect to the plain LEDBAT situation. First, the increase phase of the second flow is slightly longer, as the perturbations of the queuing delay measurements slow down the ramp-up. Second, the latecomer flow attains a slower value of the congestion window, because of its smaller target derived by its different view of the base delay. Nevertheless, random pacing does not constitute a solution, as we assist to the same unfair situation, and we thus disregard it in the following.

#### 9.3.1.2 Slow start

We have seen that in the LEDBAT -LEDBAT interaction, the linear controller *alone* may get stuck in an unfair state during a relatively long time. Yet, comparing the middle and bottom plots of Fig. 9.3, we gather a very important observation: whenever a loss event happens, the competing flows may be able to re-establish fairness (at least to a certain degree).

In other words, a loss event resynchronizes the start of the flows, possibly draining the queue empty and thus allowing each flow to gather correct measures of the base delay. Extending this observation, it seems as though it is *necessary* for each LEDBAT flow to force a loss event at startup, so to gather a correct measure of the base delay: a simple, though intrusive, way to achieve this is to enable *slow-start*. As [166] lacks a precise description of the LEDBAT slow-start (which is only briefly mentioned as an optional feature for conservative LEDBAT implementations), we resort to standard TCP slow-start mechanism. In TCP, slow start is performed by initially setting `ssthresh` to $\infty$, performing an exponential window increase and then, in case of loss, setting `ssthresh = cwnd/2` and `cwnd=0`: this process iterates until the window exceed `ssthresh`, in which case the slow-start phase ends.

We gauge the impact of slow-start on the network and user performance in terms of efficiency $\eta$, fairness $F$ and loss rate $L$, which is a indirect evaluation of the impact of slow start on VoIP/Gaming flows.

As before, only two flows share the bottleneck and we consider i) the ideal case where neither TCP nor LEDBAT implement slow-start, ii) a more realistic case where both TCP and LEDBAT implement the same slow-start behavior. To examine the latecomer situation, we neglect the case $\Delta T = 0$, since no fairness issues were observed in this case, and instead consider the start time of the second flow to be uniformly distributed in $\Delta T = U(0, 10)$ s, reporting the average of 100 simulation runs. For reference, we also consider the two values of $\Delta T \in \{2, 10\}$ s reported early in Fig. 9.3, and perform 10 simulation runs per each value of $\Delta T$ (jittering the start time of the second flow by a time lapse uniformly distributed in $[0, 0.1]$ s at each run). We now consider both a low-capacity $C_{ADSL} = 2$ and an high-speed $C_{HS} = 10$ cases, and set the buffer size $B$ to

Table 9.1: Link utilization $\eta\%$, mean $\mu$ and standard deviation $\sigma$ Fairness $F$ and Loss rate $L$. TCP versus LEDBAT and LEDBAT versus LEDBAT scenarios, with/without Slow-Start, for different Capacities $C$, Buffer sizes $B$ and time gap $\Delta T$.

| | | | | Without Slow-Start | | | | |
|---|---|---|---|---|---|---|---|---|
| Scenario | $C$ | $B$ | $\Delta T$ | $\eta$ | $F$ | | $L$ | |
| | Mbps | Pkts | sec | [%] | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| TCP | 2 | 10 | 2 | 99 | 0.60 | $6.5\cdot10^{-4}$ | $6.2\cdot10^{-3}$ | $9.4\cdot10^{-6}$ |
| LEDBAT | | | 10 | 97 | 0.60 | $4.2\cdot10^{-3}$ | $6.2\cdot10^{-3}$ | $2.1\cdot10^{-5}$ |
| | | | U(0,10) | 98 | 0.61 | $6.8\cdot10^{-2}$ | $6.2\cdot10^{-3}$ | $4.5\cdot10^{-4}$ |
| | 10 | 50 | 2 | 99 | 0.53 | $1.1\cdot10^{-3}$ | $3.0\cdot10^{-4}$ | $1.3\cdot10^{-6}$ |
| | | | 10 | 97 | 0.55 | $8.0\cdot10^{-4}$ | $3.1\cdot10^{-4}$ | $1.0\cdot10^{-8}$ |
| | | | U(0,10) | 98 | 0.54 | $4.6\cdot10^{-3}$ | $3.0\cdot10^{-4}$ | $2.4\cdot10^{-6}$ |
| LEDBAT | 2 | 10 | 2 | 99 | 0.70 | $1.2\cdot10^{-1}$ | $5.8\cdot10^{-5}$ | $3.8\cdot10^{-5}$ |
| LEDBAT | | | 10 | 96 | 0.80 | $1.8\cdot10^{-1}$ | $4.8\cdot10^{-5}$ | $4.2\cdot10^{-5}$ |
| | | | U(0,10) | 98 | 0.83 | $1.8\cdot10^{-1}$ | $3.8\cdot10^{-5}$ | $3.7\cdot10^{-5}$ |
| | 10 | 50 | 2 | 99 | 0.73 | $4.4\cdot10^{-2}$ | - | - |
| | | | 10 | 97 | 0.53 | $4.7\cdot10^{-4}$ | - | - |
| | | | U(0,10) | 98 | 0.64 | $1.8\cdot10^{-1}$ | - | - |

| | | | | With Slow-Start | | | | |
|---|---|---|---|---|---|---|---|---|
| Scenario | $C$ | $B$ | $\Delta T$ | $\eta$ | $F$ | | $L$ | |
| | Mbps | Pkts | sec | [%] | $\mu$ | $\sigma$ | $\mu$ | $\sigma$ |
| TCP | 2 | 10 | 2 | 99 | 0.58 | $1.0\cdot10^{-3}$ | $1.5\cdot10^{-2}$ | $1.5\cdot10^{-3}$ |
| LEDBAT | | | 10 | 94 | 0.58 | $2.6\cdot10^{-3}$ | $1.3\cdot10^{-2}$ | $9.7\cdot10^{-4}$ |
| | | | U(0,10) | 98 | 0.60 | $4.5\cdot10^{-3}$ | $6.6\cdot10^{-3}$ | $4.1\cdot10^{-5}$ |
| | 10 | 50 | 2 | 99 | 0.57 | $6.4\cdot10^{-3}$ | $1.2\cdot10^{-3}$ | $1.1\cdot10^{-4}$ |
| | | | 10 | 97 | 0.58 | $6.8\cdot10^{-3}$ | $1.3\cdot10^{-3}$ | $1.1\cdot10^{-4}$ |
| | | | U(0,10) | 98 | 0.55 | $1.8\cdot10^{-3}$ | $6.8\cdot10^{-4}$ | $3.8\cdot10^{-6}$ |
| LEDBAT | 2 | 10 | 2 | 99 | 0.85 | $6.5\cdot10^{-2}$ | $7.1\cdot10^{-4}$ | $8.2\cdot10^{-6}$ |
| LEDBAT | | | 10 | 96 | 0.83 | $5.8\cdot10^{-2}$ | $6.4\cdot10^{-4}$ | $5.7\cdot10^{-5}$ |
| | | | U(0,10) | 98 | 0.83 | $1.0\cdot10^{-1}$ | $1.1\cdot10^{-3}$ | $2.3\cdot10^{-3}$ |
| | 10 | 50 | 2 | 99 | 0.93 | $9.6\cdot10^{-2}$ | $4.3\cdot10^{-4}$ | $1.3\cdot10^{-8}$ |
| | | | 10 | 96 | 0.99 | $2.6\cdot10^{-3}$ | $4.1\cdot10^{-4}$ | $2.0\cdot10^{-6}$ |
| | | | U(0,10) | 98 | 0.96 | $8.3\cdot10^{-2}$ | $4.4\cdot10^{-4}$ | $5.9\cdot10^{-5}$ |

values slightly above the bandwidth delay product and able to accommodate about twice as much as the delay target of LEDBAT flows. Simulation lasts for 300 seconds, and results refer to the time interval $[\Delta T, 300]$ s where both flows are active at the same time.

Results are reported in Tab. 9.1. Top part of the table reports the TCP vs LEDBAT case, while LEDBAT vs LEDBAT is reported at the bottom. Left portion of the table refers to the case when no slow-start is used, while results obtained when slow-start is activated are reported on the right portion.

Simulation results confirm our intuition: the slow-start phase reintroduces fairness on the LEDBAT vs LEDBAT case, while leaving the TCP vs LEDBAT case almost unchanged. For instance, notice that in the worst-case for the fairness metric (represented by $(C, B, \Delta T)$=$(10, 50, 10)$ where the behavior is similar to the one early reported in the middle plot of Fig. 9.3), the use of slow-start raises the LEDBAT vs LEDBAT fairness from $F = 0.53$ to $F = 0.99$. Even in the extreme case (not shown in the table) of a capacity $C = 2$ Mbps and a buffer $B = 100$ packets, i.e., and ADSL link with a very large buffer (about 500 ms), the fairness between two LEDBAT

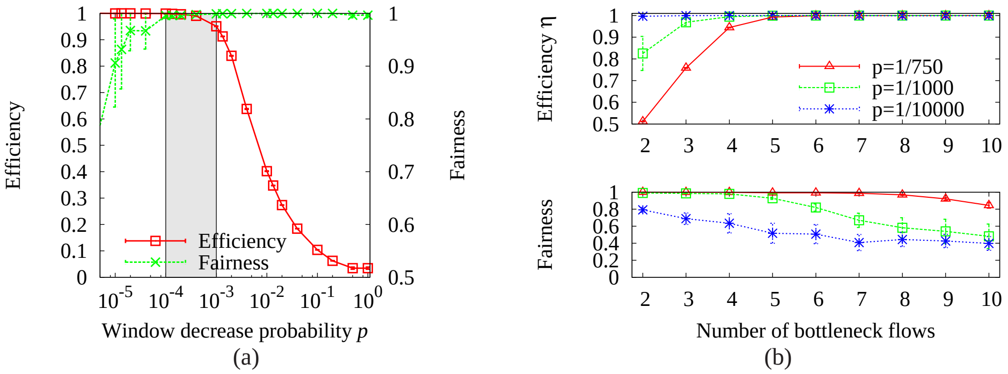


Figure 9.6: (a) Efficiency and fairness as a function of $p$ and (b) Performance of random drop for different number of flows $N$.

flows increases from $F = 0.57$ to $F = 0.77$ when slow-start is used (with a limited loss rate $L = 4 \cdot 10^{-3}$).

Concerning the loss rate, we expect slow-start to generate loss events only at the start of each connection: therefore, we expect the loss rate $L$ to be limited. From the table, we gather indeed that, despite the loss rate grows by about one order of magnitude when slow-start is enabled, nevertheless the absolute amount of losses is always very limited. In case only LEDBAT flows, with slow-start enabled, share the bottleneck, loss rate tops to about $L = 1 \cdot 10^{-3}$ in the worst case.

Despite its beneficial effects, the introduction of such an aggressive mechanism in a low priority protocol seems contrary to LEDBAT original design goals. In fact slow-start also disturbs the operation of other protocols sharing the bottleneck, as they will experience losses as well. Though we have seen the real number of packet losses can be very limited, causing only minor troubles to other services, in the following we try to devise some less intrusive solutions to the fairness issue, which will be anyway inspired by the lesson taught by slow-start.

### 9.3.2 Introducing multiplicative decrease

From the study of the slow-start solution we can derive a simple intuition: the introduction of a multiplicative decrease in the window dynamics, which causes a sudden drop of sending rate, can relieve the fairness issue. In fact, multiplicative window drops clearly accelerate the buffer drain, thus allowing flows to better estimate the base delay and potentially converge to a stable and fair regime. In other words, we conjecture LEDBAT additive decrease component to be the principal cause of unfairness.

We actually analytically demonstrate the intrinsic instability and unfairness due to the additive decrease component (which was early observed by Jain in [52]) in the next chapter, where we employ mathematical tools to support our claims. We explore two ways of explicitly introducing a multiplicative decrease in the LEDBAT protocol: first, we superpose a probabilistic window drop to the LEDBAT linear controller in Sec. 9.3.2.1; then, we directly replace the additive decrease with a multiplicative one in Sec. 9.3.2.2.

#### 9.3.2.1 Random window dropping

We have previously observed that when the buffer empties, for instance due to losses caused by slow-start, then the two flows can correct their estimation of the base delay and converge to a fair state. Should LEDBAT flows autonomously slowed down their rate at regular intervals, we could avoid forcing losses in the buffer (i.e., slow-start) altogether. A simple way to induce this behavior is to randomly drop the congestion window: upon reception of an acknowledgment packet, in addition the adjustments specified by the LEDBAT protocol, we also halve the congestion window with a constant probability $p$. At flow level, this results in a dropping rate proportional to the current transmission rate. The evolution of the congestion window in the simple case of two flows with a drop probability $p = 10^{-4}$ is reported in Fig. 9.5-(c), showing a fair share.

Now we want to identify an optimal range of values for the drop probability $p$. We preliminary consider the case of two flows arriving at the bottleneck with a gap of $\Delta T = 10\,s$ plus a random jitter uniformly distributed in $[-1, 1]$ ms. In Fig. 9.6-(a), one can observe the resulting resource allocation in terms of efficiency (left axis) and fairness index (right axis) as a function of the chosen drop probability $p$. For each value of $p$ we represent mean and variance (with vertical bars) of the considered metric over $25$ simulations, each one lasting $300$ s. As expected, for small values of $p$ we obtain a low fairness index (because the drop event is not frequent enough), but an efficient utilization of the bottleneck. On the opposite side, when $p$ becomes high the efficiency is extremely compromised, while the fairness is restored. Despite the natural tradeoff between fairness and efficiency, values of $p$ in the grey-shaded range $[10^{-4}, 10^{-3}]$ seem to allow a fair and efficient share of resources.

Still, the selection of the random probability $p$ strongly depends also on the number of flows sharing the bottleneck: the larger the number of flows, the larger $p$ should be in order to have all flows simultaneously slow down to allow newcomer flows to measure the right base delay. To confirm this intuition we report in Fig. 9.6-(b) the behavior of $\eta$ and $F$ for three values of $p$ when $N \in [2, 10]$. Mean and variance over 100 simulations of the considered metrics are plotted for the case where each flow starts randomly in $[0, 60]$ s. If the efficiency remains very high, with a good utilization of link starting from $N = 4$ for all $p$ settings, the fairness index shows an improvement over the plain LEDBAT case, but is however far from the optimum. In fact, when multiple flows are involved, one should use a much higher probability to achieve a perfect share of resource, which would in turn impose a more significant cost in term of link efficiency, especially for small values of $N$.

#### 9.3.2.2 Multiplicative Decrease

The encouraging results of the previous section suggest taking a step further and replacing the LEDBAT additive decrease with a multiplicative one altogether. Therefore, we modify the algorithm so that, whenever an ack packet carries a delay sample exceeding the target $\tau$, the window drops by a factor $\beta < 1$, i.e., in the code this translate to adding the following check

```
if off_target < 0 then cwnd *= beta
```

Notice that the multiplicative decrease is rate-dependent, and thus penalizes flows proportionally to their sending rate (window). Fig. 9.5-(d) shows the evolution of the congestion window for two competing flows with $\beta = 0.6$. We can observe rate convergence to a stable regime where each flow gets a fair share of the capacity, once both flows have correctly estimated the base delay. Moreover, at steady state flows decrease their windows simultaneously: this is a desirable property, since newly arriving flows will have the occasion to correctly measure the propagation

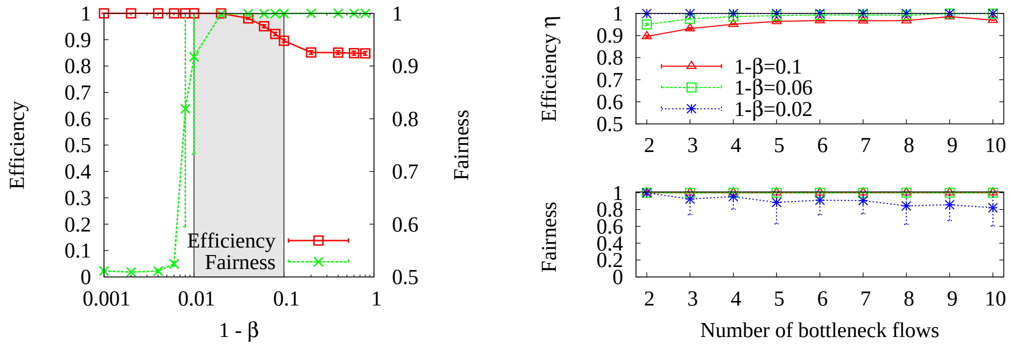


Figure 9.7: (a) Efficiency and fairness for different values of $\beta$ and (b) Performance of multiplicative decrease for different number of flows

delay.

Like in the random drop solution of Sec. 9.3.2.1, a careful choice of the multiplicative factor $\beta$ has to be made. Following the same approach used before, we first study the case of two flows and then consider the general case with a greater number of flows. In Fig. 9.7-(a) we plot mean and variance over 25 simulations of efficiency and fairness for increasing values of $\beta$ (we actually report the value $1 - \beta$ on the x-axis). As before, flows starts with a gap of $\Delta T = 10\,s$ plus a random jitter. As expected, values of $\beta$ close to $0$ (i.e., $1 - \beta$ close to 1) solve the fairness issue but introduce an efficiency loss. On the contrary values of $\beta$ close to 1 are not able to solve the latecomer advantage, yielding low fairness values. Values belonging to the gray-shaded part of the graph ($\beta \in [0.90, 0.99]$) instead guarantee both fairness and efficiency.

Fig. 9.7-(b) shows the case of $N > 2$ flows: on the one hand the efficiency of smaller values of $\beta$ improves when multiple flows are involved; on the other, the fairness decreases for larger values of $\beta$. In fact, to prevent the delay estimation error, the multiplicative decrease factor has to be smaller in the multiple flows scenarios, where a more significant drop in the sending rate is needed. However with respect to the previous solution, the multiplicative decrease achieves better performance both in terms of efficiency and fairness. In particular the value $\beta = 0.94$ has optimal results for both metrics when $N > 4$, which is a typical situation for P2P applications.

## 9.4 Related work

Congestion control studies on the Internet date back to [94] and it is out-of-scope to provide a full review of the existing literature here, as it will likely take a complete thesis. Still, it is mandatory to cite a couple of references sharing the same LEDBAT low-priority spirit [103, 111, 118, 175]. Authors of [103] implement a lower-priority protocol at the application layer, employing a receiver window based algorithm. The 4CP [118] protocol changes instead the usual TCP congestion control algorithm, introducing two phases, a good and bad phase (i.e., congestion), by means of which their controller is able to guarantee a long-term stable throughput. This is possible if the application can tolerate transient periods of time with lower sending rates, which actually allow foreground protocols to operate undisturbed.

Although not a lower-than-best-effort protocol, we cannot exempt ourselves from citing TCP Vegas [41], as it has been the first version of congestion control using delay measurements, in particular the RTT estimation, as congestion signal. Furthermore, like LEDBAT Vegas aims at introducing a small fixed amount of additional delay in the bottleneck, yet without yielding to normal TCP. While in an all TCP Vegas setting the protocol has good properties, being able to achieve better throughput with respect to loss-based congestion algorithms (i.e., TCP Reno and later versions) and not being affected by RTT unfairness, nevertheless it suffers from a few drawbacks which have hindered its diffusion.

Going back to lower-priority protocols, TCP-NICE [175] extends the delay-based behavior typical of TCP Vegas with a multiplicative decrease reaction to early congestion, which is actually detected when the number of packets experiencing a large delay in an RTT exceeds a given threshold. On the other hand, TCP-LP enhances the loss-based behavior of TCP Reno with an early congestion detection based on the distance of the instantaneous one-way delay from a weighted moving average calculated on all observations. In case of congestion, the protocol halves the rate and enters an inference phase, during which, if further congestion is detected, the congestion window is set to zero and normal TCP Reno behavior is restarted. These two solutions, TCP-LP [111] and TCP-NICE [175], differ from LEDBAT in that the latter aims at introducing a *bounded* extra delay: i.e., when queuing delay reaches a given target, the LEDBAT protocol slows down its transmission rate to ensure the queuing delay target is not exceeded. Notice that this is especially important for VoIP, gaming, and all other interactive applications that are sensitive to delay.

Related work has already tackled the intra-protocol fairness issue affecting delay-based congestion control algorithms. In particular, regarding TCP Vegas [41] which is the first example of such family of techniques, such a problem was early pointed out [128] in relation to (i) to route changes and (ii) to persistent congestion when multiple concurrent Vegas flows insist on the same bottleneck. The latter is the same malfunctioning we spotted in the original LEDBAT design: latecomer flows overestimate the base delay because of the queuing delay old flows have already put in the buffer. LEDBAT proposers clearly took advantage of this literature when designing the protocol, for instance when they adopted the same solution of [128] to the rerouting problem (i.e., by using only the recent history of delay observations for the base delay estimation), but they neglected the latecomer advantage. Nevertheless, authors of [23] have remarked some problems of LEDBAT flows subjected to route changes, in particular again an unfairness issue, which authors claim requires further adjustments to the protocol.

To solve the fairness issue, researchers have followed various approaches: on the one hand, some works try to improve the estimation of the base RTT [59, 81, 182]; on the other hand, others propose techniques to achieve fairness in spite of the base delay estimation error [40, 84, 114, 117]. The first type of work usually relies on additional support from the network to correct the measurement: for instance, [182] uses an out-of-band priority packet which skips the queue and provides a good estimation of the base delay; [81], instead, adapts its parameters according to the number of congested routers on the path, thus relying on their feedback. Authors of [40, 84] follow the opposite approach and prove that a particular choice of parameters allows flows to converge to a fair share of available bandwidth. The delay based algorithm proposed in [117] was also shown capable of dealing with noisy delay measurement, thanks to the careful choice of the function of the controller. Finally, in [114], authors propose a new delay based AIMD algorithm and choose a backoff factor which avoids measurement errors. Our work is the first to study this issue for a lower-than-best-effort protocol and to achieve together efficiency, fairness and lower-priority by reintroducing the multiplicative decrease component, for we correctly identify the root cause of unfairness in the addictive decrease component [52] rather than in the measurement error.

Other related work concerning the BitTorrent application has been already reviewed in Sec. 8.5. Most of the work on LEDBAT , instead, adopts a simulative approach [22, 44], with a few exception employing a measurement approach [164]. In [44], authors focus on a comparison of low-priority protocols, contrasting LEDBAT with TCP-LP [111] and TCP-NICE [175], showing that LEDBAT has the lowest level of priority; however, they also shows that the protocol as is lacks any easy way of tuning the level of low-priority, which might be a desirable property. Along similar lines, authors in [22] investigate the policies for dynamic parameter tuning, in particular as far as the `GAIN` parameter is concerned. They propose to adopt an adaptive value of the multiplicative coefficient of the congestion window, when the steady state is reached: in this way they prevent fluctuation around the equilibrium and reduce the delay introduced in the bottleneck. Authors in [164] instead study LEDBAT in a local testbed, employing different real ADSL modems, focusing on the interaction of LEDBAT and active queue management techniques that are becoming commonplace in modern home gateways (cfr. Sec. 8.5 for more details).

## 9.5 Summary

In this chapter, we evaluated the LEDBAT protocol by means of simulation. We carefully reviewed the specification of the algorithm as found in the IETF draft [166] and studied simple scenarios from which we gathered this main takeaways:

- LEDBAT is able to achieve inter-protocol *friendliness* (i.e., yield to TCP) and at the same time to efficiently exploit the extra available resources.
- Inter-protocol *fairness* is maintained even in case of wrong parameter settings, in which case LEDBAT simply degenerates into TCP.
- The PID controller *alone* is not sufficient to guarantee intra-protocol fairness: in presence of large buffers, a latecomer advantage arises among LEDBAT flows.

Later we concentrated on possible solution to this later problem, i.e., the intra-protocol fairness issues arising in LEDBAT. The first two solutions, based on random pacing and on an additional slow start phase, were inspired by the discussions within the LEDBAT IETF working group [9]. Their objective is to de-correlate flow dynamics, so to allow flows to get a correct estimate of the queuing delay. In both cases, unfairness appears to be only partially or ineffectively relieved: the random jitter addition shows no real improvement in terms of fairness, whereas introducing a slow-start phase goes against LEDBAT low-priority goals.

While investigating the reasons behind the unfairness, the main cause appears to be actually the addictive decrease component preventing the system from converging to a stable regime, as already observed by Jain in [52]. In the LEDBAT case, the error in the estimation of the queuing delay further hinders the convergence to a fair state. Therefore, we devised two possible alternatives to incorporate a multiplicative decrease term in the LEDBAT controller: first, by adding a probabilistic drop to the additive increase/decrease dynamics, then by directly replacing the additive decrease with a multiplicative one altogether. The results are promising as they display a region of the parameters (drop probability $p$ or decrease factor $\beta$) where fairness can be achieved at no or little expense of efficiency. Although both solutions have their merits, the multiplicative-decrease one may be more appropriate, given: (i) better results in terms of efficiency and fairness when multiple flows are competing on the same link, and (ii) the solid theoretical foundation of a purely multiplicative window decrease, already proved in [52], for increase/increase factors equal for all competing flows. In next chapter we actually propose and study in more complex scenarios an additive increase/multiplicative decrease controller tailored to LEDBAT goals, which better solves the problem achieving fairness and efficiency.

# Chapter 10

# Designing an efficient and fair LEDBAT

After unveiling the latecomer advantage which affect the LEDBAT protocol, in this chapter we propose an effective modification to its design, in oder to achieve efficiency, fairness and low-priority altogether. Results presented in this chapter are currently submitted for publication in [45]. To meet our goals, we adopt a systematic approach. First we analytically investigate the properties of the original linear controller of LEDBAT in Sec. 10.1, proving that unfairness naturally derives from the shape of the controller. Therefore, we modify the algorithm, especially, but not only, by reintroducing a multiplicative decrease component. The fairness and stability properties of the new protocol, named fLEDBAT from fair-LEDBAT are demonstrated mathematically by means of a fluid model in Sec. 10.2. We implement the fLEDBAT in `ns2` to study its properties in more complex scenarios in Sec. 10.3 which can not be treated analytically: we study the impact of the traffic model (e.g., backlogged vs chunk-based transfers) in Sec. 10.4, the sensitivity of the protocol to parameter changes in Sec. 10.5. We finally evaluate the performance of the protocol in P2P-like settings from a single-peer as well as a whole swarm perspective in Sec. 10.6.

## 10.1 Current LEDBAT fairness issues

According to the original draft proposal[166], LEDBAT maintains a minimum one-way delay estimation $D_{min}$, which is used as base delay to infer the amount of delay due to queuing. LEDBAT flows have a target queuing delay $\tau$, i.e., they aim at introducing a small, fixed, amount of delay in the queue of the bottleneck buffer. Flows monitor the variations of the queuing delay $q(t) - D_{min}$ to evaluate the distance $\Delta(t)$ from the target:

$$\Delta(t) = (q(t) - D_{min}) - \tau, \tag{10.1}$$

where $q(t)$ is the queueing delay measured at time $t$. The value of the offset $\Delta(t)$ is then used to drive the congestion window evolution, which is updated packet-by-packet at each acknowledgement reception as it follows:

$$cwnd(t+1) = \begin{cases} cwnd(t) + \alpha\frac{\tau - \Delta(t)}{\tau}\frac{1}{cwnd(t)} & \text{if no loss ,} \\ \frac{1}{2}cwnd(t) & \text{if loss.} \end{cases} \tag{10.2}$$

where $t$ is a discrete time variable that increments by 1 at each ack arrival and $cwnd(t)$ is the congestion window at time $t$. The drawbacks of such a congestion window update mechanism mainly consist in the intra-protocol unfairness coupled with a poor calibration of the LEDBAT level of (low) priority with respect to TCP.

### 10.1.1 Impact of additive decrease

We demonstrate that the additive decrease, rather than the measurement errors, is the main cause of unfairness in the LEDBAT protocol: in other words, the late-comer advantage is actually a fundamental drawback of the additive decrease term, meaning that the original design is currently misguided. Such conclusions have been already drawn by Jain for a simpler scenario in [52].

Without loss of generality, let us consider the case of $N$ LEDBAT flows with the same round trip time $R(t)$, sharing the same link of capacity $C$ and finite buffer size $B$. Each flow $i \in \mathcal{N}$, with $\mathcal{N} = \{1, 2, \dots, N\}$, starts at $t_i \geq 0$, with $t_1 \leq t_2 \leq \cdots \leq t_N$ and with an initial congestion window $W_i$. Given the packet-level congestion window dynamics in (10.2), we demonstrate the following statement.

**Proposition 1.** *If $N < \frac{B}{\tau C}$, and $d_{max}(t_N) \triangleq \max_{i,j \in \mathcal{N}}[W^i(t_N) - W^j(t_N)] > 0$, then the system is unfair, i.e. $\exists t^* \geq t_N$, such that $\forall t > t^*$ $d_{max}(t) > 0$.*

*Proof.* Given (10.2), a simple fluid representation of the window dynamics of flow $i$, $W_i(t)$, in continuous time, is:

$$\frac{dW_i(t)}{dt} = \frac{1}{R}\frac{\tau - Q(t)}{\tau}, \tag{10.3}$$

where we supposed for simplicity $R(t) \approx R$, which is true for large propagation delay (the proof can be easily extended to the case of variable round trip delays). Since the estimated queuing delay can be different for each flow, depending on its stored base delay, we replace $Q(t)$ by $Q_i(t)$, i.e., the queue occupancy measured by each sender, and simply observe that $Q_i(t)$ varies in the interval $(Q(t) - (N-1)\tau, Q(t))$. Indeed, the last flow makes the largest error in the estimation of the queuing delay, because it measures as base delay the actual propagation delay increased by $(N-1)\tau$, the sum of the target delay of all preceding flows. It follows that, $\forall i, j \in \mathcal{N}$:

$$W^i(t) - W^j(t) = W^i(t_N) - W^j(t_N) + \int_{t_N}^{t} \frac{Q_j(u) - Q_i(u)}{R\tau} du$$

where $|Q_j(t) - Q_i(t)|$ is bounded by $(N-1)\tau$. Hence, if we choose $t^*$ equal to $t_N + \frac{W^{i*}(t_N) - W^{j*}(t_N)}{(N-1)/R}$, with $(i^*, j^*) = \arg\max_{i,j \in \mathcal{N}} W^i(t_N) - W^j(t_N)$, it results:

$$\begin{aligned}
d_{max}(t) &\triangleq \max_{i,j \in \mathcal{N}} W^i(t) - W^j(t) \\
&\geq \max_{i,j \in \mathcal{N}} W^i(t_N) - W^j(t_N) + \frac{(N-1)}{R}(t - t_N) \\
&= \frac{(N-1)}{R}\left((t - t_N) + \frac{W^i(t_N) - W^j(t_N)}{N-1}R\right) \\
&= \frac{(N-1)}{R}(t - t^*) > 0, \qquad \forall t > t^*.
\end{aligned}$$

□

**Observation 2.** *The fact that the system evolves towards an unfair state is strictly related to the fact that the dynamic equations, describing the state of the system are unstable. Besides equations*

*(10.3) for the sources, we have*

$$\frac{dQ(t)}{dt} = \sum_{i=1}^{N} \frac{W_i}{R} - C\mathbb{1}_{Q_t>0}. \tag{10.4}$$

*Equations (10.3),(10.4) define a linear system of ODEs with characteristic polynomial* $\lambda^{N-1}\left(\lambda^2 + NC\tau R\right)$. *The corresponding eigenvalues are* $\lambda_1 = 0$, $\lambda_{1,2} = \pm i\sqrt{\frac{N}{R\tau}}$, *and the matrix associated with the system of ODEs is easily shown to be diagonalizable by standard algebra. As the eigenvalues have zero real part, the system cannot consequently be asymptotically stable. Being the matrix diagonalizable, the solution is limited for every* $t$, *however the dependence to the initial condition never vanishes because of the zero real part of the eigenvalues. In addition, the associated matrix cannot be inverted because of the zero eigenvalue which implies that the solution of the system has an orbit around any* $(W_1, \dots, W_N, Q)$ *such that* $\sum_i W_i = RC$, $Q = \tau$. *In other words, the linear response of LEDBAT is never able to make the stable point reachable from any initial condition: this is the root cause of the observed latecomer advantage phenomenon that we aim at solving in the following.*

## 10.2 Proposed LEDBAT modification

To address this issue, we propose to modify the delay-based decrease term and *to introduce a multiplicative decrease* continuously driven by the estimated distance from the target, $\Delta(t)$. Intuitively, the multiplicative window reduction response to congestion allows to slow down enough the source sending rate to make a stable (and fair) point always reachable. Clearly, to guarantee at the same time fairness and protocol efficiency, a proper choice of the decrease factor has to be made, so as to prevent significant (and unnecessary) drops in the congestion window. In addition, we observe that the additive increase term as in (10.2) leads LEDBAT flows to slow down the increase factor until the target $\tau$ is reached, in which case the window increase completely stops. This clearly implies a smaller convergence to the target and hence a minor efficiency if compared to the case of a constant additive increase factor independent of $\Delta(t)$. Based on the above observation, we propose to modify the increase term as well, and *to introduce an additive increase* according to a constant factor $\alpha$ as in TCP Reno. Notice that in this way, we expect to achieve better efficiency performance without violating the low priority requirements as expressed in the LEDBAT draft. Indeed, by selecting $\alpha \leq 1$ the additive increase component can be made at most as aggressive as TCP.

Summarizing the observation from the previous section, we propose to modify the congestion window evolution as follows:

$$cwnd(t+1) =$$

$$\begin{cases} cwnd(t) + \alpha\frac{1}{cwnd(t)} & \text{if no loss and } \Delta \leq 0, \\ cwnd(t) + \alpha\frac{1}{cwnd(t)} - \frac{\zeta}{\tau}\Delta & \text{if no loss and } \Delta > 0, \\ \frac{1}{2}cwnd(t) & \text{if loss.} \end{cases} \tag{10.5}$$

In the following sections we quantify the overall improvement deriving by such a congestion window update by means of both a *fluid model*, which provides a closed-form characterization of the stationary throughput and *simulations*, which allow the study of more complex scenarios. In the remainder of this chapter, we refer to the modified version of LEDBAT as fair-LEDBAT (fLEDBAT).

Table 10.1: Notation

| | |
|---|---|
| $N$ | Number of fLEDBAT flows |
| $C$ | Link capacity |
| $\{W^i(t)\}_{i=1,\dots,N}$ | Congestion windows at time t |
| $\{X^i(t)\}_{i=1,\dots,N}$ | Instantaneous rates at time t |
| $Q_t$ | Queue occupancy at time t |
| $\alpha$ | Additive Increase factor |
| $\zeta$ | Multiplicative Decrease factor |
| $R_t$ | Round trip time at time t |
| $\tau$ | Queuing delay target |

### 10.2.1 Fluid model description

In this section we develop a fluid model of the congestion window and hence of the transmission rate of one or more fLEDBAT flows aimed at capturing first order system dynamics. The congestion window is now a continuous variable both in time and in space, $W(t)$ (the notation is summarized in Tab.10.1). We consider the case of $N$ fLEDBAT flows sharing the same link of capacity $C$ and experiencing the same Round Trip Time $R_t$. The model generalizes to the case of heterogeneous RTT, yet for the sake of simplicity we focus on the homogeneous case. In addition, we make the following assumptions:

- The round trip time $R_t$ is defined by the sum of twice the propagation delay, $R$, transmission delay $1/C$ and queueing delay $q(t)$. We further assume that the propagation delay is predominant, i.e. $R_t \approx R$.

- The queueing delay $q(t)$ is defined as ratio of the queue occupancy $Q_t$ at time $t$ divided by the link capacity $C$, i.e., $q(t) = Q(t)/C$. Thus, we assume that the queuing delay information *instantaneously* propagates to the sender, neglecting thus the delay in the feedback loop.

- We further assume that flows can correctly estimate the queuing delay, which is equivalent to take $D_{min} = 0$.

- By Little's law, we assume that congestion windows and link rates are linked by $X_t^i = W_t^i/R_t, \quad \forall i = 1, ..., N$.

Remark that the assumption that flows can correctly estimate the queuing delay may not hold in practice. As such, we expect that simulation results may show an offset with respect to the model predictions, which is due to such simplifying assumption. There are however two main reasons for which we believe these assumptions, which make the problem tractable, are reasonable as well. On the one hand, additional mechanisms to enhance the delay estimation accuracy could be then adopted in order to ameliorate the overall protocol performance: this has been done in previous work [111], and is also part of the current BitTorrent effort [59] to reduce the measurement error and hence reinforcing our assumptions. On the other hand, a more fundamental reason is that the characterization of protocol dynamics in absence of such estimation error is a necessary step in the fLEDBAT protocol design – as, even though on simplistic settings, important properties of the protocol such as efficiency and fairness can be *proved* to hold with the help of a rigorous framework.

### 10.2.2 Fluid system dynamics

Let us consider the case of a N fLEDBAT connections, whose congestion window evolves according to (10.5). The corresponding flow-level congestion window evolution is:

$$\frac{dW_i(t)}{dt} = \frac{\alpha}{R} - \frac{\zeta}{\tau}\left(\frac{Q(t)}{C} - \tau\right)\frac{W_i(t)}{R}\mathbb{1}_{W_i(t)\geq 0}\mathbb{1}_{Q(t)\geq C\tau}, \tag{10.6}$$

where we denote by $W_i(t)$ the instantaneous congestion window at time $t$ for connection $i$ in the fluid system. Assuming an approximately constant round trip delay, we replace $R_t$ by $R$ in (10.6). The instantaneous queue occupancy instead satisfies:

$$\frac{dQ(t)}{dt} = \sum_{i=1}^{N}\frac{W_i(t)}{R} - C\mathbb{1}_{Q(t)\geq 0}. \tag{10.7}$$

where, in other words, only the flow that exceeds the capacity creates queuing in the buffer.

Thus, the instantaneous rate of connection $i$, $X_i(t)$, satisfies:

$$\frac{dX_i(t)}{dt} = \frac{\alpha}{R^2} - \frac{\zeta}{R\tau}\left(\frac{Q(t)}{C} - \tau\right)X_i(t)\mathbb{1}_{\{X(t)\geq 0\}}\mathbb{1}_{Q(t)\geq C\tau} \tag{10.8}$$

and (10.7) can be re-written as

$$\frac{dQ(t)}{dt} = \sum_{i=1}^{N}X_i(t) - C\mathbb{1}_{Q(t)\geq 0}. \tag{10.9}$$

### 10.2.3 System convergence

The main result we derive from the model is the existence of a unique and globally stable solution. We also express, with closed form formulæ, the performance of the protocol at the equilibrium, proving its *efficiency* and *fairness* – which was our initial goal. Let us start by proving that the system admits a unique solution.

**Proposition 3.** *The system of ODEs (10.8)-(10.9) admits the unique equilibrium* $P^* = (X_1^*, \ldots, X_N^*, Q^*)$

$$X_i^* = C/N,\ i = 1, \ldots, N \qquad\qquad Q^* = C\tau + \frac{N\alpha\tau}{\zeta R} \tag{10.10}$$

*where* $X_i^*$ *and* $Q^*$ *denotes the stationary values of* $X_i$ *and* $Q$ *respectively.*

*Proof.* We consider the stationary regime by the condition $(\dot{X}_i, \ldots, \dot{X}_N, \dot{Q}) = (0, \ldots, 0)$

$$\begin{aligned}
\dot{Q} = 0 &\Leftrightarrow \sum_{i}^{N} X_i^* = C, \\
\dot{X}_i = 0 &\Leftrightarrow 0 = \frac{\alpha}{R^2} - \frac{\zeta}{RC\tau}(Q^* - C\tau)X_i^*, \\
&\Leftrightarrow 0 = \frac{\alpha}{R^2} - \frac{N\alpha}{CR^2}X_i^* \Leftrightarrow X_i^* = C/N,\ i = 1, \ldots, N.
\end{aligned} \tag{10.11}$$

□

Then, the following proposition states that this unique equilibrium is also globally stable (see [176]).

**Proposition 4.** *The system of ODEs (10.8)-(10.9) is globally stable in $P^*$.*

*Proof.* Let us write $\mathbf{X} = (X_1, \ldots, X_N)$, we consider the trajectories of the point $(\mathbf{X}, Q) \in \mathbb{R}_+^{N+1}$ driven by the ODEs (10.8)-(10.9). In the region $A = \{\mathbf{x}, q : 0 < q < C\tau\}$, the state equations simplify to

$$\begin{cases} \dot{X}_i & = \frac{\alpha}{R^2} \Rightarrow X_i = X_i(0) + \frac{\alpha}{R^2}t, \;\; \forall i \\ \dot{Q} & = \sum_{i=1}^N X_i - C \Rightarrow \\ Q(t) & = Q(0) + (\sum_{i=1}^N X_i(0) - C)t + \frac{N\alpha}{2R^2}t^2 \end{cases} \tag{10.12}$$

Clearly, for any $(\mathbf{X}(0), Q(0)) \in A$, there exists a finite $t \geq 0$ such that $(\mathbf{X}_t, Q_t) \notin A$. This means that all points $(\mathbf{X}_t, Q_t) \in A$ are unstable. The unique equilibrium point $P^*$, calculated in Prop.3, is outside $A$. For $(\mathbf{X}, Q) \notin A$, the state equations become

$$\begin{cases} \dot{X}_i & = \frac{\alpha}{R^2} - \frac{\zeta}{CR\tau}(Q - C\tau)X_i \\ \dot{Q} & = \sum_{i=1}^N X_i - C \end{cases}$$

We now use the technique of the Lyapunov function to show that $P^*$ is a stable point, i.e. we have to show that there exist a function $V_t$ defined in a neighborhood of $P^*$, positively defined for $t \geq 0$, with orbital derivative negatively semidefinite (in which case, the solution $P^*$ is stable in the sense of Lyapunov, see [176] Theorems 8.1-8.3). Outside $A$, we define the Lyapunov function by

$$V(\mathbf{X}, Q) = \sum_{i=1}^N (X_i - X_i^*) - \log\left(\frac{X_i}{X_i^*}\right) + \frac{\zeta(Q - Q^*)^2}{2RC\tau} \tag{10.13}$$

Clearly, $V(P^*) = 0$, $V(\mathbf{X}, Q) \geq 0 \; \forall (\mathbf{X}, Q) \notin A$, and

$$\dot{V}(\mathbf{X}, Q) = \sum_{i=1}^N (\dot{X}_i - \dot{X}_i \frac{X_i^*}{X_i}) + \dot{Q}\frac{\zeta(Q - Q^*)}{RC\tau} \tag{10.14}$$

$$= \sum_{i=1}^N \frac{\dot{X}_i}{X_i}(X_i - X_i^*) + (X_i - X_i^*)\left[\frac{\zeta(Q - Q^*)}{RC\tau}\right]$$

$$= \sum_{i=1}^N (X_i - X_i^*)\left[\frac{\alpha}{X_i R^2} - \frac{\zeta(Q - C\tau)}{RC\tau} + \frac{\zeta(Q - Q^*)}{RC\tau}\right]$$

$$= \sum_{i=1}^N (X_i - X_i^*)\left[\frac{\alpha}{X_i R^2} - \frac{N\alpha}{CR^2}\right] \tag{10.15}$$

$$= \frac{\alpha}{R^2}\sum_{i=1}^N (X_i - X_i^*)\left[\frac{1}{X_i} - \frac{1}{X_i^*}\right] = -\frac{\alpha}{R^2}\sum_{i=1}^N \frac{(X_i - X_i^*)^2}{X_i X_i^*}.$$

Therefore $\dot{V}(\mathbf{X}, Q)$ is negatively semidefinite for any ball including the equilibrium point $P^*$. This proves that $P^*$ is an equilibrium globally stable as per [176].

□

Once we know that the system has a unique globally stable equilibrium, we want to show the *convergence rate* of the system in a neighborhood of the equilibrium. This can easily be evaluated considering *local stability* properties of the system.

**Proposition 5.** *The system of ODEs (10.8)-(10.9) is locally stable in the equilibrium* $P^* = (X_1^*, \ldots, X_N^*, Q^*)$

*Proof.* We write $(\dot{X}_1, \ldots, \dot{X}_N, \dot{Q}) = (f_1, \ldots, f_N, g)$, for $X_i > 0$, $Q > 0$ where $f_i$ and $g$ are defined as follows:

$$\begin{cases} f_i(X, Q) = & \frac{\alpha}{R^2} - \frac{\zeta}{C\tau R}(Q - C\tau)X_i \mathbb{1}_{Q(t) \geq C\tau} i = 1, ..., N \\ g(X, Q) = & \sum_{i=1}^{N} X_i - C \end{cases} \tag{10.16}$$

Linearizing the system of ODEs in $P^*$, and defining $\Delta X_i = X_i - X_i^*$, $\Delta Q = Q - Q^*$, and $\mathbf{Y} = (\Delta X_1, \ldots, \Delta X_N, \Delta Q)$ we obtain $(\tilde{f}_1, \ldots, \tilde{f}_N, \tilde{g}) = \dot{\mathbf{Y}} = A\mathbf{Y}$ where $A$ is a $(N+1) \times (N+1)$ square real matrix defined as follows:

$$A = \begin{pmatrix} -\frac{\alpha}{CR^2} & 0 & \cdots & 0 & -\frac{\zeta}{C\tau R} \\ 0 & -\frac{\alpha}{CR^2} & \cdots & 0 & -\frac{\zeta}{C\tau R} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ 1 & 1 & \cdots & 1 & 0 \end{pmatrix}$$

The characteristic polynomial is then

$$\left(\lambda + \frac{\alpha}{CR^2}\right)^{N-1} \left(\lambda^2 + \frac{\alpha}{CR^2}\lambda + N\frac{\zeta}{C\tau R}\right)$$

whose roots have all real part negative. □

**Proposition 6.** *The solution of the system of ODEs (10.8)-(10.9) converges to the global stable equilibrium* $P^*$ *at a rate* $e^{-\Theta t}$ *with,*

$$\Theta = \frac{\alpha}{CR^2}\left(\frac{1 + \mathbb{1}_{\zeta \leq \zeta^*}\sqrt{1 - \zeta/\zeta^*}}{2}\right)$$

*and* $\zeta^* = \frac{\alpha^2 \tau}{4NCR^3}$.

*Proof.* We calculate the dominant eigenvalue of the matrix $A$, i.e. the eigenvalue with the real part with the smallest absolute value. □

To conclude, we summarize our main findings in the following observation, expressing the results in terms of the expected performance of fLEDBAT.

**Observation 7.** *Prop. 3,4,5,6 prove that the designed protocol is efficient* ($X^* = C$)*, and long term fair* ($X_i^* = C/N$)*. In addition the queuing delay attains the target* $\tau$ ($Q^*/C = \tau + \frac{N\alpha\tau}{C\zeta R}$) *by an error of* $\frac{N\alpha\tau}{C\zeta R}$.

Thus, our initial goals of an *efficient* and *fair* protocol is met. Clearly, a number of issues need further investigation (i.e., how the protocol performs in practice where not all modeling assumption holds, what is the impact of parameters and of packet-level dynamics, how does it performs against TCP, etc.) that we dig in the next section by means of a thorough simulation campaign in a number of different scenarios.

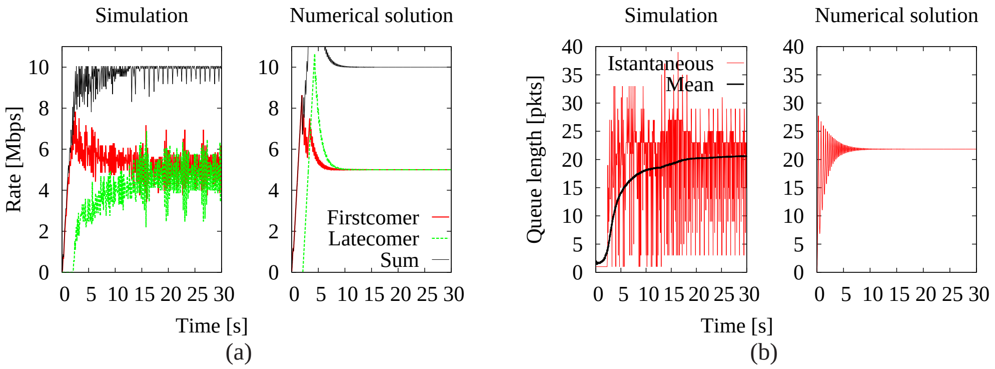


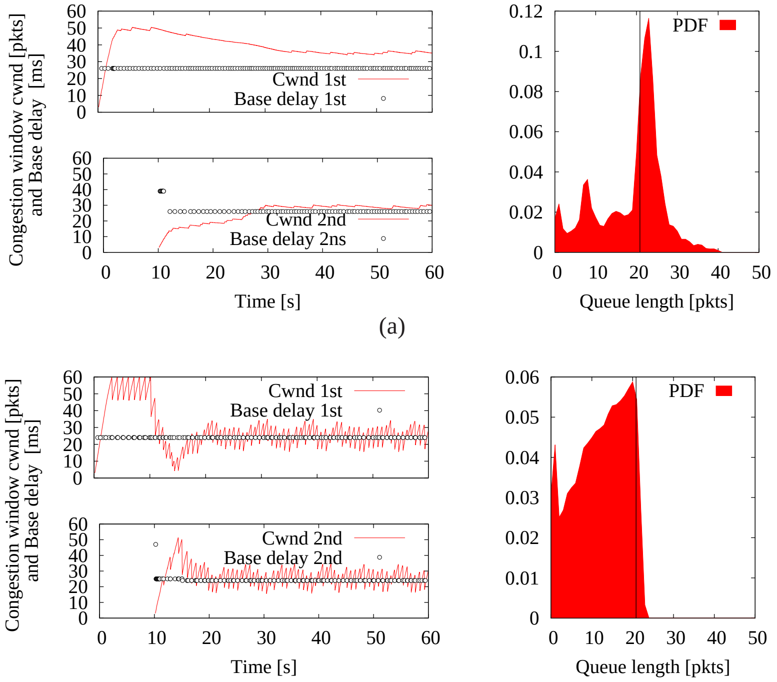
Figure 10.1: Comparison of (left) simulation and (right) numerical solution for (a) Rates and (b) Queue length. The similar average values in left and right plots confirm a good fit between packet level simulation results and flow level numerical results.

## 10.3 Simulation overview

So far, we have developed a mathematical model of our new proposed protocol in order to formally prove its properties. However, the model is based on a number of simplifying assumptions and it furthermore neglects some aspects due to packet-level quantization (i.e., queue length and congestion window in multiple of fixed-size packets as opposite to continuous rate in the fluid model). To fill this gap, in the remainder of this chapter we carry out a thorough packet-level `ns2` [15] simulation campaign, to cope with scenarios where such assumptions do not hold. We made our implementation available as open source at [10].

Unless otherwise stated, we consider a reference scenario consisting of a bottleneck link of capacity $C = 10$ Mbps and buffer size $B = 100$ packets. For the sake of simplicity, we consider fixed size packets equal to $P = 1500$ Bytes. Data flows in a single direction, and acks are not delayed, dropped nor affected by cross-traffic on their return path (except in the P2P scenarios reported in Sec. 10.6.2). All flows have the same round trip time $RTT = 50$ ms, half of which is due to the propagation and transmission delay components of the bottleneck link (i.e., a one-way base delay of $25$ ms), to which we add a jittering component uniformly distributed in [0,1] ms to avoid synchronization issues. We defer the study of more realistic scenarios, including heterogeneous delays, different access technologies, background traffic and P2P-like traffic models to Sec. 10.6. As far as TCP flows are concerned, we select the NewReno flavor, enabling the selective acknowledgement SACK option due to its growing widespread [126]. Notice that by selecting the NewReno flavor, we gather conservative results since we expect more recent TCP variants implemented by default in Linux and Windows (respectively Cubic [162] and Compound [171]) operating systems to be more aggressive than traditional NewReno flows. Each simulation point reported in the following is the results of $10$ simulation runs, over which we gather the average and standard deviation of the metrics of interest.

However, we still need to provide evidence of the fluid model accuracy: we do so by comparing the numerical solution of the fluid model with `ns2` simulation results. We consider the simple network scenario described above, and two fLEDBAT flows with the same target $\tau = 25$ ms. Notice that delay target $\tau$ was initially set to $25$ ms in the IETF draft and to $100$ ms in the BEP specification. However, as shown in [44], provided that all flows have the *same target value*, which is furthermore set to a value *not exceeding the buffer size*, the actual value of $\tau$ is not



(a)

(b)

Figure 10.2: Time evolution of fLEDBAT dynamics with (a) chunk-based $\zeta = 0.01$ and (b) backlogged $\zeta = 5$ traffic models.

critical. Hence, results shown in the following are valid for both target settings. For the time being, we fix the decrease component by setting $\zeta = 0.1$ (and explore the impact of $\zeta$ later on). To recreate the conditions for the latecomer unfairness phenomenon, the two flows do not start at the same time, but their start time are separated by 2 seconds. The time evolution of the system state is depicted in Fig. 10.1, which reports both the evolution of the flow rates $X_1, X_2$ (top) and the buffer occupancy $Q_t$ (bottom) gathered either by numerical solution (right) or ns2 simulation (left). As a general comment, the numerical solution shows a good agreement with the simulation results (although as expected packet-level dynamic exhibits much wider fluctuations, while the fluid model gives an average behavior). Indeed, notice that the *average* of queue occupancy and flow rates yielded by the fluid system closely matches the simulation dynamics (for the sake of readability, we plot the *moving average* of the queue length gathered via simulation alongside the instantaneous occupancy). As expected, both numerical and simulation results show that the capacity is, after an initial transient phase, fairly shared among flows (i.e., $\overline{X}_i \approx C/2, \forall i$) and that furthermore the queuing delay target is reached (i.e., $\overline{Q}_t \approx C\tau$).

In the following we study several aspects of the LEDBAT protocol family. Sec. 10.4 addresses the impact of different *traffic models* on protocol performance, considering backlogged vs chunk-by-chunk transfers. Then, Sec. 10.5 addresses a *sensitivity analysis* of the protocol to $\zeta$ parameter variation. Finally, Sec. 10.6 compares LEDBAT and fLEDBAT under more realistic, P2P-like, networking scenarios.

## 10.4 Impact of traffic model

In this section we assess how the fLEDBAT protocol deals with different kind of traffic. Besides the classical backlogged transfer, we simulate a chunk-based transfer, which mimics the behavior of a BitTorrent data exchange between two peers.

### 10.4.1 Chunk-by-chunk transfer

In this scenario, we consider sources that continuously transmit chunks of data, where each chunk has the typical BitTorrent size of $250\,$kB (nearly $170$ full payload packets). As soon as a chunk transmission ends (i.e., when the last acknowledgment for that chunk has been received at the sender side), a new chunk transmission is scheduled with the same peer. Notice that this traffic model, which emulates the dynamics of P2P traffic exchange, differs from backlogged transfers in that, after the last data packet of a chunk has been sent, the source peer stops transmitting for about $RTT$ seconds until the matching acknowledgement is received, and a new chunk transmission can start. Notice also that we keep the congestion window parameter across chunks (i.e., congestion window is *not* reset between subsequent chunks exchanged with the same peer).

Fig. 10.2-(a) reports the time evolution of the system dynamics when $\zeta = 0.01$: in the left portion, congestion window and base delay estimation of the firstcomer (top) and latecomer (bottom) flows are reported, while the right portion shows the distribution of the queue length. In this case, it can be seen that, although the latecomer initially has an incorrect view of the base delay (as in LEDBAT), the multiplicative decrease phase of the firstcomer allows the latter to correct its estimate, after which the performance share converges to an equitable state. Due to (i) the continuous adjustment of AI and MD dynamics and (ii) the fact that chunk transmission seldom pauses the transmission, the queue is no longer stable as for the standard LEDBAT case (see previous chapter), but fluctuates around the occupancy value predicted by the model (represented by a solid vertical line).

### 10.4.2 Backlogged transfer

In the case of backlogged transmission, a latecomer phenomenon may still arise depending on the value of $\zeta$: indeed, when $\zeta$ is too small, the multiplicative decrease component of the first flow is slower than the additive increase of the latecomer, which is thus unable to correct its wrong estimation. However, provided that $\zeta$ is large enough to let the queue flush, fLEDBAT can still reintroduce fairness.

Results for the backlogged scenario are reported in Fig. 10.2-(b) for $\zeta = 5$. Especially, the queue now seldom flushes (as it can be seen by the increased probability to have a null queue length shown by the PDF) which helps latecomers gain a correct view of the base delay. In this case, though, the model slightly overestimate the queue size: indeed, due to larger $\zeta$ values, the congestion window fluctuations are now wider. Also, as the queue flushes, the protocol is less efficient with respect to the previous cases too, because the capacity is not fully utilized all the time.

We point out that, since fLEDBAT is designed to be a low-priority protocol, slight inefficiency should be tolerable. Conversely, in case efficiency, rather than low-priority, would have been a more important goal, then an alternative approach is possible, which we already explored in the previous chapter: indeed, a simple way of draining the queue empty (which allows each sender to gather correct measures of the base delay) is to use TCP-like slow-start at flow startup. The downside, in this case, is that as slow-start causes losses, which may have undesirable side effects

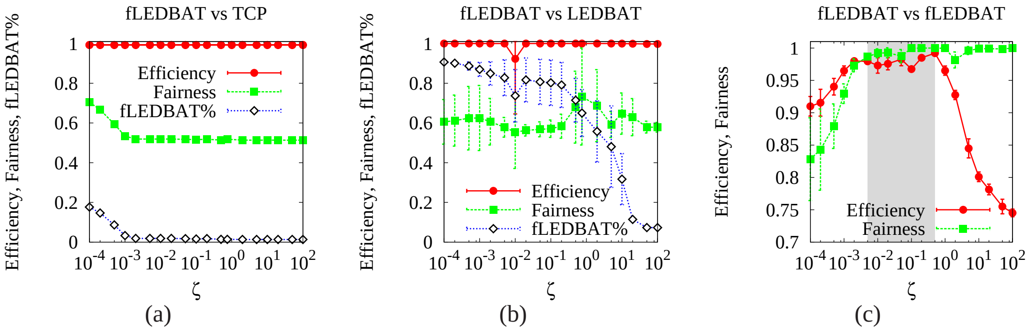


Figure 10.3: Sensitivity analysis to $\zeta$: Efficiency, long-term fairness and protocol breakdown of a fLEDBAT flow sharing the bottleneck with (a) a TCP flow, (b) a LEDBAT flow (c) another fLEDBAT flow.

on interactive traffic (e.g., VoIP, gaming) and is thus less indicated, in our opinion, in this context. Moreover, as shown in the previous chapter by simulation and in [164] by experiments, this would not be enough to fully reinstate fairness, which in the LEDBAT case is rooted in the AIAD dynamics more than the in errors of the base delay estimation.

## 10.5 Sensitivity analysis

In this section we carry out further simulations, to assess the impact of the choice of the parameter $\zeta$ on the protocol performance. In order to gather a complete sensitivity analysis of fLEDBAT parameters, we consider several scenarios: (i) a TCP Reno flow competing with a fLEDBAT flow, (ii) a LEDBAT flow competing with a fLEDBAT flow, (iii) two fLEDBAT flows competing for the same bottleneck. All flows operate in chunk-by-chunk transmission mode.

As performance metrics, we consider, as usual, Jain's *fairness* $F$, *efficiency* $\eta$ and, additionally, *protocol breakdown* of the data transfer, defined as the percentage of traffic sent by fLEDBAT sources over the total traffic, which immediately conveys the level of low priority of fLEDBAT with respect to other protocols insisting on the same bottleneck.

### 10.5.1 Observations on $\alpha$, $\tau$ and low-priority level

Since a careful sensitivity analysis focused on gain $\alpha$ and target $\tau$ has already been carried out in [44], in the following we briefly summarize the main lessons as far as these two parameters are concerned, while we provide a thorough set of simulation results for the newly introduced parameter, i.e., the decrease factor $\zeta$.

Let us consider the target parameter $\tau$ first. Already in the homogeneous case of several flow with equal settings, [44] shows that the performance of LEDBAT cannot be easily controlled by tuning the target $\tau$. Indeed, the low priority level can be changed only when the $C\tau$ product approaches the buffer size – however changes in the priority level are too steep for very small variations of $\tau$. Moreover, there is no single value of $\tau$ that can adapt to both low-capacity and high-capacity links at the same time. Finally, in the heterogeneous case of several flows with different settings, even a small difference between values of $\tau$ yield to extremely unfair situations, with flows having larger $\tau$ being more aggressive. For this reason, we adhere to the mandatory value specified by the draft $\tau = 25\,\text{ms}$ and do not consider $\tau$ as a free parameter.

Let us now consider the gain parameter $\alpha$: in this case, is worth noting that the increase component of fLEDBAT differs from that of LEDBAT. Indeed, LEDBAT increase is proportional (with $\alpha$ proportionality constant) to the offset from the target, meaning that as the estimated queueing delay approaches the target, the congestion window growth slows down. In the case of fLEDBAT instead, the congestion window growth is still proportional to $\alpha$, but constant (i.e., no longer dependent on the offset from the target). Therefore, the value of $\alpha = 1$ is constrained in reason of the low-priority goal (so to match the 1-packet-per-RTT TCP growth in congestion avoidance). Finally, due to the bounded target, fLEDBAT inherits from LEDBAT the lowest possible level of priority [44] compared to NICE and to TCP-LP.

### 10.5.2 fLEDBAT vs TCP

Fig. 10.3(a) shows the efficiency and fairness performance when a single TCP and a single fLEDBAT flow share the bottleneck: first of all, we can see that low-priority goal is met, as TCP is enjoying the largest portion of the capacity (fLEDBAT breakdown goes to 0% and fairness drops to $1/N$). As expected, efficiency is high: as we already observed in [155] for LEDBAT, fLEDBAT is still able to push some bytes on the link, thereby increasing the overall link utilization with respect to the case where a single TCP Reno flow insists on the bottleneck.

With the exception of extremely low values of $\zeta < 10^{-3}$ (which soften the effect of the multiplicative decrease, and sharpen the impact of the Reno-like additive increase), the low-priority goal is therefore satisfied. Thus, selecting $\zeta$ is not a concern as far as heterogeneous fLEDBAT vs TCP scenarios are considered.

### 10.5.3 fLEDBAT vs LEDBAT

Fig. 10.3(b) shows the efficiency and fairness performance when a single fLEDBAT flow and a LEDBAT one share the bottleneck. We randomize the start time of both flows in the [0,10] sec interval, so that the latecomer can be either of the two protocols. In this case, we gather fLEDBAT is generally more aggressive (due to the AI dynamic, which is more aggressive than LEDBAT Proportional Integer Derivative (PID) dynamic) until $\zeta$ grows too large, in which case the reverse happens (due to the MD dynamic being more drastic than the PID dynamic). Specifically, less than 20% of the bottleneck is occupied by LEDBAT when $\zeta < 10^{-2}$. For larger values of $\zeta$ though, LEDBAT becomes increasingly competitive with fLEDBAT: the crossover happens at about $\zeta = 5$, after which fLEDBAT becomes even lower priority than LEDBAT.

In no case, however, the share is fair and a latecomer phenomenon may still arise. Consider that, when LEDBAT start first and saturates the bottleneck, it induces a very steady queue. Therefore, when an fLEDBAT latecomer flow arrives on the bottleneck, it measures an incorrect base delay. However, as LEDBAT reacts with a *linear* decrease to the increasing delay, the fLEDBAT latecomer will not have the opportunity to correct is estimate – as it otherwise does whenever the firstcomer flow reacts with a *multiplicative* decrease to the increasing delay. Hence, when $\zeta$ is small, the fLEDBAT latecomer can starve the LEDBAT flow.

### 10.5.4 fLEDBAT vs fLEDBAT

We now consider the intra-protocol scenario in which two fLEDBAT flows share the bottleneck. We set the start time of latecomer flow to $t = 10\,$s, which was shown in [155] to represent a worst case scenario for the fairness index. Fig. 10.3(c) reports results for varying $\zeta$, where we omit this time the fLEDBAT breakdown. From the picture, it is clear that fLEDBAT is able to operate fairly and efficiently under a wide range of parameters. Overall, taking into account also the previous

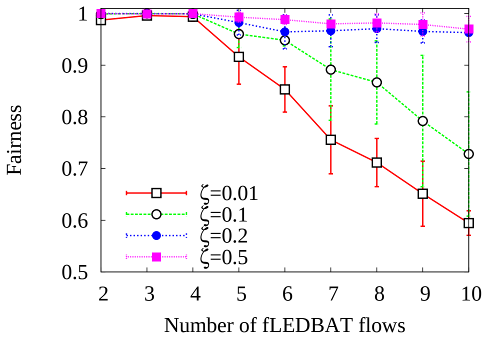


Figure 10.4: Variation of the intra-fairness index of fLEDBAT, for different values of $\zeta$ and a growing number of flows.

remark in the intra-protocol fLEDBAT vs TCP scenario, we have that any value of $\zeta$ in the gray shaded zone yield to an efficient, fair and low-priority system.

Finally, we present results for a varying number of fLEDBAT flows insisting on the bottleneck, with in $N \in [2, 10]$. Every $k$-th flows arrive at $t_k = k10$ s, and we evaluate the performance only after the $N$-th last flow has arrived in the bottleneck. Results are reported in Fig. 10.4, where we select a few values of $\zeta \in \{0.01, 0.1, 0.2, 0.5\}$ from the shaded gray zone of Fig. 10.3(c). As it can be seen, it is always possible to find a value of $\zeta$ that guarantees fairness for the whole set of flows, with any values in the range providing good results for the number of flows that are typically concurrently active in BitTorrent. Moreover, the very same values of $\zeta$ that provide fair resource share, were already shown to provide efficient use of the resources for $N = 2$ flows in Fig. 10.3(c), which still holds when $N > 2$ (omitted to avoid cluttering the pictures).

## 10.6 P2P Scenarios

In order to compare fLEDBAT vs LEDBAT performance under more realistic conditions, we finally consider a chunk-based scenario that (i) loosely mimics the behavior of BitTorrent peers, (ii) employs Internet-like heterogeneous delays and access rates, (iii) considers background traffic and coupled queues, (iv) addresses the impact of chunk sizes.

We consider two different scenarios: first a single BitTorrent peer and a single queue in isolation, using a more sophisticated traffic model and observing the effect of delay heterogeneity alone. Second, we study a BitTorrent swarm-like scenario, as close as possible to the actual target application scenario of LEDBAT, with 100 peers continuously exchanging data among them. In this case, the state of queues is no longer independent, as data traffic mixes with acknowledgement traffic in the reverse direction, causing a coupling of the congestion controllers as well. To gather even more realistic results, we finally add HTTP-like background traffic to the swarm scenario, studying its impact on congestion controls dynamics.

Before presenting the results of our simulations, it is worth spending a couple of words on the traffic model we use to simulate a BitTorrent peer. Like a real BitTorrent source [58], our simulated peers open a number of connections to other $N$ peers, but they actively exchange chunks of data with only a restricted number $M < N$ of the available peers at the same time (set to $M = 5$ and $N = 10$). We employed different chunk sizes, ranging from 250 KB - 4096 KB [122], using 250 KB chunks unless otherwise stated. At the end of each chunk transmission, the sender chooses the next destination peer as follows: with a *persistence probability* $P_P$, the sender will send another

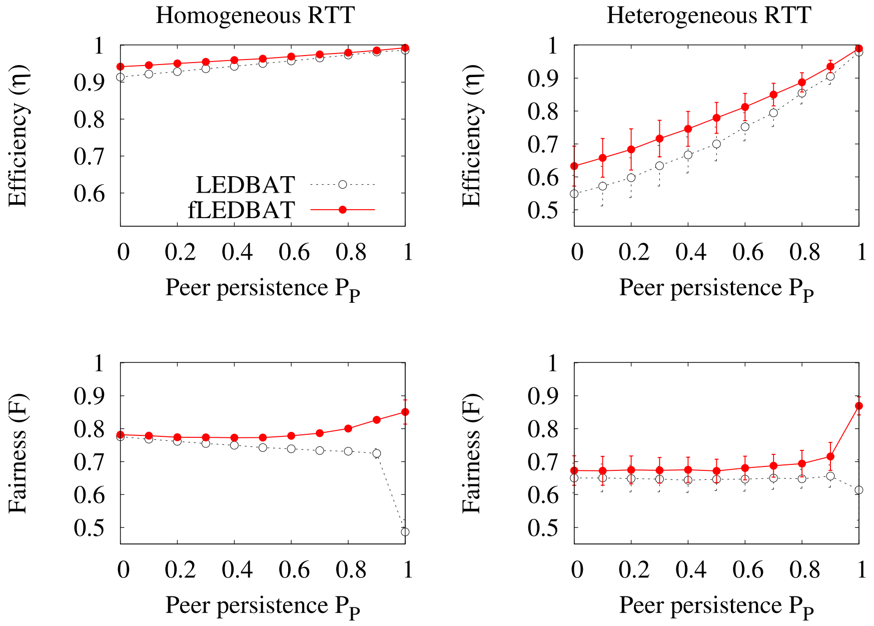


Figure 10.5: Efficiency (top) and Fairness (bottom) of fLEDBAT in the homogeneous (left) and heterogeneous (right) RTT scenarios.

chunk to the same peer, keeping the congestion window settings; with probability $(1 - P_P)$, the sender will choose an inactive neighbor at random, resetting in this case the congestion window to 1. A detailed discussion of the meaning of different values of $P_P$ is found later on. We chose not to implement all the application-level details of a BitTorrent source (e.g., tit-for-tat, optimistic unchoking, signaling, etc.), yet we argue that this simpler traffic model is a good approximation of the actual peers interaction.

As far as network conditions are concerned, we consider both *FTTH symmetric* access capacities ($C = 10$ Mbps, $B = 100$ packets, default unless otherwise stated) and *ADSL asymmetric* capacities ($C = 1$ Mbps uplink, $C = 8$ Mbps downlink $B = 100$ packets). To add further realism, we simulate both a *homogeneous* network setup (i.e., in which all peers have the same propagation delay $RTT = 50$ ms) as well as an *heterogeneous* scenario (default unless otherwise stated) where the propagation delay of the access link of each peer is distributed according to realistic delay measurement [178], with mean equal to 37.9 ms.

### 10.6.1 Single peer perspective

Let us start by considering a single peer behind a FTTH connection, exchanging 250 KB chunks with peers chosen according to the algorithm described above: when a chunk transfer is completed, the source keeps the same destination peer with a probability $P_P$, and changes it with probability $(1 - P_P)$. fLEDBAT and LEDBAT are simulated separately, i.e. all peers either use the former or the latter protocol, and we consider both an homogeneous and heterogeneous delay scenario. For fLEDBAT we set the parameter $\zeta = 0.1$, which yielded good performance in the sensitivity analysis.

In our experiments we explore the full range of $P_P$ in $[0.1]$. As $P_P \to 0$, we expect the performance of the two protocols to be close: indeed, when connections are reset every 170-th packet (which corresponds to 250 KB chunks), the protocols are basically always in transient state and the target is likely not even reached during a chunk transfer. Conversely, differences are expected to arise in the more stable scenarios $P_P \to 1$, where congestion parameters are kept across chunks. Actually, we expect real BitTorrent source to have a behavior similar to a sender with $P_P \geq 0.8$: in fact, one source normally tries to keep 4 out of the 5 “best” (i.e., higher capacity) peers while at the same time continuously discovering new, potentially “better”, peers (i.e., by means of optimistic unchoking).

Results of the comparison are reported, in term of efficiency and fairness, in Fig. 10.5. Since flows are no longer backlogged, from now on we consider *short-term fairness*, measured at 1 Hz rate (i.e., corresponding to a good tradeoff between a minimum number of RTTs to have statistically meaningful results, and a maximum time lag, as flows may die out after a single chunk transfer). Notice that we expect short-term fairness to be harder to achieve than the long-term fairness considered in the previous sections (hence we expect lower $F$ values).

At first glance, we remark that under all scenarios and $P_P$ values, fLEDBAT is more efficient (due to AI) and fair (due to MD) with respect to LEDBAT, and the gain is more evident exactly in the operational range of BitTorrent (i.e., $P_P \geq 0.8$). In the homogeneous case depicted in the two plots on the left, as expected, the fairness gap exacerbates as $P_P \to 1$: in this case, fLEDBAT ability to correctly measure the base delay leads to an increase of the fairness metric. On the contrary, LEDBAT fairness decreases as $P_P$ grows, due to the latecomer issue: the effect is stronger when $P_P = 1$, as in this case the unfair situation persists through the whole duration of the experiment and leads to a consistent drop of $F$. Similar considerations hold for the heterogeneous case (see plots on the right), although in this case the heterogeneous delays introduce $RTT$ unfairness in both fLEDBAT and LEDBAT, reducing the absolute value of $F$ (i.e., we do not attempt to account for the $RTT$ bias in the fairness definition). However, $RTT$ unfairness does not translate into serious issues (such as long time starvation), and however fLEDBAT always guarantees a fairness higher than LEDBAT (as latecomer advantage disappears).

As far as efficiency $\eta$ is concerned, when the congestion window parameters are reset every chunk ($P_P = 0$), the link capacity is not fully utilized even in the homogeneous case. The heterogeneous case further adds inefficiency, as flows with higher $RTT$ increase more slowly their congestion window, hence further wasting link capacity. However, it is worth pointing out that the additive increase component of fLEDBAT makes it more efficient than LEDBAT under any circumstance, while the multiplicative decrease component guarantees at the same time its lower-priority with respect to TCP.

### 10.6.2 Entire swarm perspective

We now consider an entire swarm, where 100 peers continuously exchange data among them using 5 parallel upload slots, as in BitTorrent: hence, the uplink queue of each peer contains a mix of (i) data packets being uploaded to the swarm and (ii) acknowledgement packets related to data being downloaded from the swarm. As this happens for each queue, P2P traffic interacts both in the forward and backward directions. Notice also that the end-to-end congestion controls of data transfers are tightly coupled in a non-trivial way, as in our model each peer maintains multiple connections to a set of other peers, which moreover dynamically evolve over time. However, this model does not try to model all BitTorrent dynamics (e.g., chunk trading logic), but rather assumes that peers are always able to find content where they seek it (i.e., a large file where there is enough chunk diversity in the system). Therefore, we do not attempt at measuring application-level statistics, such as the torrent download time, but rather focus on the transport-layer fairness

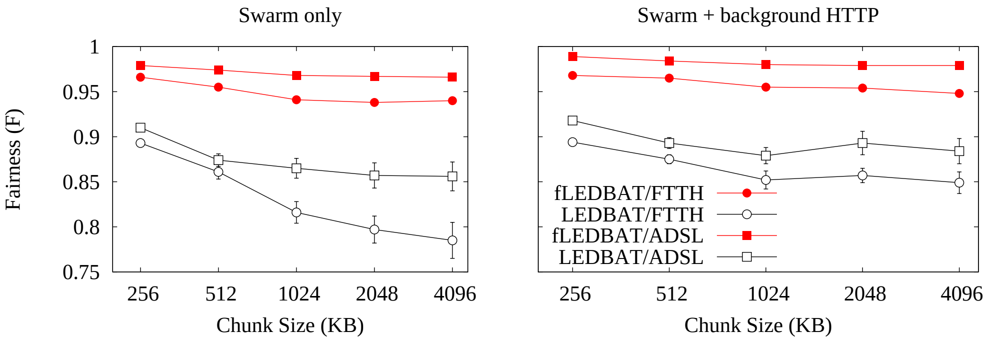


Figure 10.6: Fairness in FTTH and ADSL swarm-like scenarios, with and without background HTTP traffic

statistics.

Unlike what done for the previous scenario, here we fix the persistence probability to $P_P = 0.8$. As shown in the former experiment, each interruption in data transmission favors LEDBAT for the consequent draining of the queue lets the protocol correct the base delay estimation. Under this consideration, since BitTorrent optimistic unchoking happens on 1 slot out of 5 at low rate (each 30 seconds), $P = 0.8$ corresponds to a lower bound (since the chunk transfer can be much shorter than 30 seconds) and gives an optimistic (thus, conservative) estimate for LEDBAT fairness. Moreover, our simulation model also introduces an *idle RTT* (i.e., time elapsed between the transmission of the last data packet of a chunk and the reception of the related acknowledgment) before a new chunk is sent to a persistent peer: another pause that further simplifies LEDBAT task and that actual BitTorrent systems avoid by means of request pipelining. However, we can partly compensate for these effects by employing larger chunks.

Since homogeneous and heterogeneous RTT scenarios yielded qualitatively similar results, in the following we only consider heterogeneous delay settings for more realism, with either FTTH or ADSL access. Finally, we also simulate a scenario where peers browse the Web while participating to the swarm: in particular we add an HTTP source from which peers downloads Web-pages (files with a size uniformly distributed in the range $[0-512]$ KB) using traditional TCP. There are always 25 peers, selected at random, with active HTTP downloads, so that a quarter of the total swarm is always involved in the short interactive background Web transfers.

Short-term fairness results are shown in Fig. 10.6: first, notice that the coupling of queues is more beneficial for the fairness index (for both LEDBAT and fLEDBAT) with respect to the single queue scenario shown in Fig. 10.5. At the same time, under all cases fLEDBAT consistently achieves higher fairness than LEDBAT. Larger chunk sizes, as expected, badly affect LEDBAT because they reduce the number of transmission interruptions in the (which helped in emptying the queues). Conversely, fLEDBAT appears almost insensitive to chunk size.

Comparing ADSL vs FTTH scenarios, we see that in the latter case a further cause of unfairness may arise: indeed, as capacities are symmetric and one peer may have several downloads ongoing (only upload slots are limited in number), downlink can become a bottleneck as well (e.g., in real systems this may happen in case of popular torrents with many seeds). The impact of background HTTP traffic is instead beneficial to the fairness of both fLEDBAT and (especially) LEDBAT: this is due to the fact that peers downloading HTTP data will send out burst of acknowledgement packets, that possibly cause buffer overflows in the uplink queues (which assist in raising fairness in LEDBAT, as shown in [155]). Background traffic has instead a smaller impact on fLEDBAT, as the protocol achieves higher fairness without external help.

## 10.7 Summary

In this chapter, we presented modifications to LEDBAT congestion control algorithm, that not only are able to achieve *low-priority inter-protocol* (i.e., against TCP) and *efficiency intra-protocol* (e.g., with other fLEDBAT flows), but also reintroduce *intra-protocol fairness*, solving thus the late-comer issues of the original LEDBAT proposal.

We modeled the fLEDBAT dynamics via a fluid-model approach, which allows, on the one hand, to detect the main issue at the base of the unfairness (i.e. the additive decrease component) and to prove the correctness of its design. From such model, we derived closed-form expressions for the average rate and queue length even in the general case with $N$ sources. Furthermore, by means of packet-level simulations, we further assess that fLEDBAT can safely operate under a number of scenarios (such as chunk-by-chunk and backlogged transmission) as it is not sensitive to the parameter selection, but operate reasonably well under a wide range of parameters.

Finally, we tested the protocol in multiple flows scenarios, comparing its performance with other lower-than best effort protocols, in realistic scenarios with heterogeneous RTTs and transfer emulating the operations of a BitTorrent peer, the original target application for LEDBAT. fLEDBAT not only solves the fairness issue, but also keeps the same good properties of LEDBAT: that is, it yields to TCP while being able to exploit the spare bandwidth. Overall, we see that our proposed modifications lead to provable performance and constitute an improvement with respect to LEDBAT in terms of both fairness and efficiency. Our simulations confirm fLEDBAT robustness even under realistic heterogeneous network conditions, on which BitTorrent can be expected to operate.

# Chapter 11

# Conclusion

## 11.1 Summary

In this thesis we investigated several issues regarding the management of P2P traffic in modern networks. In fact, P2P applications still account for an important portion of nowadays Internet traffic and their traffic, given its characteristics, can be particularly difficult for operators to deal with. Besides, even if recent measurement studies indicate that web, and especially video streaming, has become the biggest bandwidth consumer, yet P2P traffic keeps growing in absolute amount and the P2P paradigm still powers new killer applications (e.g. live-streaming and music on demand). Therefore, as P2P traffic seems to be here to stay, in this thesis we developed tools and methodologies to help operators better manage it and have it coexist peacefully with other applications.

The solutions proposed in this thesis addressed specifically the *identification*, *measurement and control* of P2P traffic in modern networks. In the following we will briefly summarize our main results.

As for P2P *traffic classification*, our studies focused on behavioral classifiers, which base the classification on the characterization of the pattern of traffic generated by a host running the application. These algorithms can outperform traditional techniques (e.g. DPI) because they are more lightweight and capable of dealing with encrypted traffic, avoiding access to packet payload altogether. In this work we applied such techniques to the classification of P2P traffic, whose protocols are usually proprietary or unknown and whose large volumes require a lightweight classification process.

First of all, we explored the space of features that could be used for behavioral classification of P2P traffic. We designed a general framework that allows to combine several criteria (e.g., timeliness, granularity, directionality of traffic, etc.) and to generate a wealth of features based only on flow-level data (e.g. NetFlow records). We intentionally kept the definition as general as possible, so that features of other classifiers can be included in the framework, facilitating the task of comparing different classification engines. We then restricted our attention to a subset of all possible features and quantified their information content useful for the classification by means of two metrics, Information Gain and ReliefF. Using the best attributes selected by these metrics and a Decision Tree classification algorithm, we showed that a very accurate (i.e, > 98% on average for the seven considered applications) and, furthermore, fine-grained (i.e. identification of the specific application) classification can be achieved by means of these simple features.

Leveraging the knowledge of the most useful attributes for P2P classification gathered in the above experiments, we restricted our attention to a specific class of applications, P2P live-streaming, of which we also know the internal workings. Thereby, we designed a specific behavioral classifier for these applications, named Abacus, which is based on a simple count of the number of packets and bytes exchanged by the target host with its peers during small time-windows. Such simple counters, which however capture distinctive properties of the applications, are used to build the Abacus signatures that, fed to a Support Vector Machine classification algorithm, provide a very accurate and fine-grained classification. Besides, an equally simple rejection criterion based on the Bhattacharyya distance is also effective in detecting “unknown” traffic, for which the classifier was not originally trained. Overall, using the complete abacus signature (including both the packets and bytes information) and the best parameter values, obtained from a thorough sensitivity analysis, the classifier accuracy exceeded 99% of correct classification for all our target applications. Additionally, our experimental evaluation over an extended dataset of passive and active traces also proved the portability of Abacus signatures over different times and network settings, accesses and conditions.

To strengthen our proposal, we also performed a comparison of Abacus with a sophisticated stochastic payload based classifier, Kiss, which was proved very accurate for the same class of applications as well. We tested the classifiers on a common set of traces, observing practically the same performance in terms of byte accuracy for both of them. Our analysis of their computational requirements, instead, clearly identified Abacus as a more lightweight process of at least one order of magnitude (in the worst case) for CPU consumption. However, this is obtained at the cost of a less general classifier: in fact, Abacus is effective only with P2P applications, while Kiss can be applied to other classes of applications as well.

In the second part of this thesis, we instead focused on *data reduction* techniques, used by operators to shrink the quantity of *measurement data* gathered from their networks, which has grown together with the amount of traffic. Our aim was to investigate whether traffic classification of P2P traffic was still possible with such data, notwithstanding the smaller information content. First of all, we tested our Abacus classifier with NetFlow flow-records, which contain all data needed to calculate the signature (i.e. packet and byte counters), but have a coarser time granularities – default NetFlow timeouts are in order of minutes, while Abacus was designed with time-windows of seconds. We demonstrated that Abacus can safely cope with such data, with only a slightly reduced classification accuracy.

We then investigated the issue of moving a behavioral classifier in the core of the network, where, due to routing, only a portion of flows directed to a host are observed and, hence, only a partial characterization of the traffic pattern is gathered. In our experiments, we tested the Abacus classifier, with simulated flow-sampling at increasing rates, and with realistic routing tables from core routers. Given the way Abacus signatures are calculated (normalized over the total number of peers seen), they can withstand flow-sampling, provided that a statistically significant sample of flows is observed; in fact, if both signaling and data flows are present in the sample, without any particular bias, then the Abacus signatures keep their discriminative power intact.

Another technique commonly used in the core to cope with the increasing transmission speeds is *packet sampling*, where only a fraction of packets are processed. We performed a minute analysis of the impact of packet sampling on network measurement, by modifying a flow-level traffic monitor to apply sampling on prerecorded traces, and by running it on a large, heterogeneous set. We used two statistical metrics to evaluate the distortion introduced by four distinct sampling policies in aggregated traffic measurement: our results showed an important degradation, already for low sampling and irrespectively of the sampling policy used. However, when we evaluated the information content of single-flow features calculated on sampling data, we observed that they can still be used for traffic classification, conserving a good correlation with the application label. In fact, we performed traffic classification using such features as attributes for a Decision Tree algorithm, and showed that they are good discriminators, achieving an accuracy above 98%, provided that the classifier is trained with data gathered at the same sampling rate of the test data.

Finally, we changed perspective and focused on P2P application themselves, and in particular on *congestion control* for such traffic. We would like elastic P2P traffic (e.g. file-sharing) to automatically detect when other applications are using the network and to automatically yield to higher priority traffic. Along this line of thoughts, BitTorrent developers have recently proposed (and already implemented in the official client) a new congestion control protocol which is undergoing a standardization process at the IETF with the name of LEDBAT. Their goal is to develop a *lower-than-best-effort* protocol able to detect congestion early and then to reduce its sending rate to give priority to other more sensitive traffic (e.g., interactive traffic). We studied the performance of this protocol by means of measurement, simulation and formal analysis.

First, we used a black-box approach to study the performance of the closed-source official implementation of the protocol found in the BitTorrent client. We setup a small testbed where we artificially enforced specific network conditions (i.e., capacity and delay) and analyzed the way the protocol reacted to such changing settings. We also tested the protocol in the wild on a home ADSL connections, competing with real-world traffic. Overall we observed a good behavior of LEDBAT, which abides by its goals: it uses the spare bandwidth efficiently, it introduces only a small delay in the bottleneck, it yields to other traffic, in particular to TCP. However, we showed also a few bugs affecting the first releases of the protocol, as well as a few cases where LEDBAT does not behave as a lower-than-best-effort protocol, due to different TCP implementation and parameter settings.

Second, following the official IETF draft, we implemented the LEDBAT protocol in the packet level simulator `ns2`. Essentially, it is a delay-based congestion control algorithm, which constantly monitors the queuing delay on the forward path and adjusts its sending rate by means of a linear controller, in order to introduce a small fixed target delay in the bottleneck. Our simulations showed that this design is effective in achieving low priority, correctly yielding to concurrent TCP connections, but they highlighted also an *unfairness issue* when two, or more, LEDBAT flows share the same link. Due to a wrong estimation of the queuing delay, the latecomer flow is able to steal all the bandwidth from the first flow: unfortunately such a situation can persist for a long time, thus possibly impacting application performance as well.

To remedy this issue, we proposed, implemented and evaluated a few possible modifications to the LEDBAT protocol to restore the fairness. While random pacing of packets of a congestion window (suggested on the mailing list) did not improve the situation, we showed that using slow-start at the beginning of a LEDBAT flow, along with the consequent buffer overflows, has the benefit of draining the queue and, afterwards, of allowing all flows to correctly measure the base delay. Still, this aggressive behavior seems in contradiction with the lower-priority spirit of the protocol.

Investigating further, we were able to find the root cause in the linear controller, and in particular in the additive decrease component, which early literature on this topic [52] has already shown to prevent the system from converging to a fair state. For this reason we decided to reintroduce a multiplicative decrease component, in a random and deterministic fashion. Both solution reintroduce fairness for some parameter settings, though with a slight loss in terms of efficiency. For this reason we proposed a more radical modification of the protocol with fair-LEDBAT (fLEDBAT): we modeled this protocol analytically and showed its good properties of efficiency, fairness and low priority, which were confirmed by our simulation as well. Moreover we also simulated a complex setting with multiple sources operating in a P2P-like scenarios, showing that our proposal is convenient also for the target application of the protocol. The relevance of our work to improve the LEDBAT protocol has been acknowledge as well by the fact that it is cited and discussed in the protocol draft [166] itself.

## 11.2 Future work

As usual in scientific research, one work, however complete it might be, not only brings answers and solutions, but it always comes with new questions and issues as well. In this section, we sketch a few possible directions which could be followed to extend the contribution presented in this thesis and to respond to the interrogatives it arises.

Regarding our research on behavioral classification of P2P traffic, first of all the framework presented is Chap. 3 should be extended to consider more complex operations and categories. This would not require any modification to its main structure, as it naturally allows such an extension, but it would greatly simplify the comparison of different classifiers. In fact, algorithms such as [32, 80, 184] could be mapped onto our framework without much effort. In such a way, not only would it be easier to run all classifiers on the same set of traces, but their features would also be directly comparable by evaluating their information content with the metrics we already used for this purpose. In this context, it would also be interesting to evaluate more sophisticate feature selection algorithms, taking into account also the correlation of features among themselves (e.g. correlation based filter), so to extract the best set of features. Finally, a more rigorous study of features portability across different networks and times, similar to the one presented in Chap. 4, would be required.

Concerning the Abacus classifier, presented in Chap. 4, we have actually conducted a very thorough evaluation of this algorithm, as we have considered its portability, we have compared it to a state of the art payload based classifier and we have also tested it with NetFlow records and under flow-sampling. Yet, there are still a few questions that come to mind. For instance, an open issue is what is the effect of *packet sampling* on Abacus signatures. In theory, we expect such a technique to heavily bias the distribution of packets and bytes, as most of small flows will not be sampled and most of the information about signaling will be lost. However, smarting sampling techniques, such as the SYN sampling we introduced in Chap. 7, might somehow alleviate this phenomenon, thus likely permitting the classification.

Another aspect that deserves further investigation is the applicability of Abacus: in this thesis we concentrated on UDP traffic, (initially) on P2P-TV application and (later) on generic P2P services. Nevertheless, with appropriate changes, the same approach might be applied to other kinds of traffic. For instance, to deal with TCP on the client side, where each connection opens a new ephemeral port allocated by the operating system, one could aggregate all ports together and obtain an Abacus signature, which, though slightly altered by other application running on the same host, might contain enough information for the classifier to work. On the server side, instead, the server socket already multiplexes all incoming connections and the Abacus classifier can probably derive a meaningful signature from the analysis of the received traffic; however, if such signatures are too similar to P2P ones, then they may confuse the classifiers and decrease the overall accuracy.

Our work on the impact of packet sampling on traffic classification could also be extend in a few ways. First, it would be interesting to investigate the impact of sampling on other classification techniques, like DPI and behavioral classifiers (e.g. on Abacus, as we already mentioned before). In such a case, it would be worth extending the evaluation to the whole set of sampling policies, also the simple ones (i.e. systematic, random, stratified sampling), since in this thesis for the classification part we considered only the smart SYN sampling, for it provided considerable advantages to the statistical classifier used. On the contrary, we could consider also more sophisticated policies, for instance systematically sampling FIN packets as well (thus improving estimation of flow size and duration), to provide the statistical classifier with additional information. Finally, while we demonstrated the necessity of using the same sampling rate both on the training and validation data, it is important to evaluate space and time portability as well, despite the issues we have discussed in Chap. 7.

As for our research on congestion control for P2P traffic and on the LEDBAT protocol, we foresee a number of possible directions of work as well. For instance, apart from the intra-protocol unfairness, which was solved in Chap. 10, in our opinion there is still a critical point in the LEDBAT algorithm definition that remains an open issue. This issue, described in [44, 164] and debated in the LEDBAT IETF working group mailing list, concerns the use of a *fixed* queuing delay target. Such fixed settings (which are referred to as "magic numbers" in the mailing list) are indeed not a good practice, as they may lead to undesired behavior: as a matter of fact, not compliant implementation may set an higher target with respect to the mandatory standard values, hence easily obtaining an unfair advantage over compliant clients. However, while fixed settings are not robust against malicious or misconfigured implementations, at the same time there is no obvious way of defining an *adaptive* target without loosing guarantees on the additional delay – that interactive applications would like to be as small as possible. Even worse, fixed settings do not take into account the evolution of network capacity and speed: for instance, a target queuing delay of 100 ms at 1 Gbps already equals to a large buffer ($\sim$12MB), while in this case a smaller value would be advisable.

Finally, it must be said that we heavily relied on simulation in our evaluation of LEDBAT and especially of fLEDBAT. While simulation represents a first imperative step to study congestion control issues, we recognize that it may not represent a good fit to study other aspects of the protocol – as, for instance, the impact of LEDBAT on the P2P application layer performance. Indeed, real systems often include delicate dynamics that are not necessarily captured or described by simulation models. Therefore, we believe that another necessary research direction involves experiments on the real system – i.e., running a LEDBAT enabled BitTorrent swarm, either on controlled testbed or on the wild (e.g., PlanetLab), to refine our understanding of LEDBAT from another, complementary, perspective. In fact, it should be remarked that, despite the overall good properties demonstrated by LEDBAT its success will be determined by user adoption: if the protocol reveals itself inefficient in real scenarios (e.g., by enlarging download completion time in BitTorrent), in particular at the early stages of its diffusion, then users might go back to good old TCP and leave LEDBAT simply as an interesting experiment.

Finally, as BitTorrent has recently gone into video, one might wonder what would be the role of the LEDBAT protocol in such a scenario. In fact, although it was not designed with such an application in mind (but rather for elastic transfers), developers will surely take advantage of the knowledge acquired while implementing LEDBAT for developing the new one for streaming. As demonstrated by recent studies [25], which showed some issues of YouTube streaming over HTTP+TCP, the interaction between application layer and transport layer can be very tricky: therefore, an independent evaluation of the new protocol, from a research perspective rather than from an engineering one, might be useful not only for the developers, but for the users as well.

# Appendix A

# List of publications

We report here the list of publications and papers under submission related to this thesis.

## A.1 Publications

- S. Valenti, D. Rossi, Fine-grained behavioral classification in the core: the issue of flow sampling, In *TRaffic Analysis and Classification (TRAC) Workshop at IWCMC 2011*, Istambul, Turkey, July 2011

- P. Bermolen, M. Mellia, M. Meo, D. Rossi, S. Valenti, Abacus: Accurate behavioral classification of P2P-TV traffic, *Elsevier Computer Networks*, 55(6):1394-1411, April 2011

- S.Valenti, D. Rossi, Identifying key features for P2P traffic classification, in *IEEE ICC'11*, Kyoto, Japon, June 2011

- G. Carofiglio, L. Muscariello, D. Rossi and S. Valenti, The quest for LEDBAT fairness, In *IEEE Globecom'10*, Miami, FL, USA, December 6-10 2010.

- A. Finamore, M. Mellia, M. Meo, D. Rossi and S. Valenti, Peer-to-peer traffic classification: exploiting human communication dynamics, In *IEEE Globecom'10, Demo Session*, Miami, FL, USA, December 6-10 2010.

- A. Pescape, D. Rossi, D. Tammaro and S. Valenti, On the impact of sampling on traffic monitoring and analysis, In *Proceedings of the 22nd International Teletraffic Congress (ITC22)*, Amsterdam, The Netherlands, September 7 - 9 2010.

- D. Rossi, C. Testa, S. Valenti and L. Muscariello, LEDBAT: the new BitTorrent congestion control protocol, In *International Conference on Computer Communication Networks (ICCCN'10)*, Zurich, Switzerland, August 2-5 2010.

- D.Rossi, S. Valenti, Fine-grained traffic classification with Netflow data, In *TRaffic Analysis and Classification (TRAC) Workshop at IWCMC 2010*, Caen, France, June 2010

- A.Finamore, M. Meo, D. Rossi, S. Valenti, Kiss to Abacus: a comparison of P2P-TV traffic classifiers , In *Traffic Measurement and Analysis (TMA) Workshop at PAM'10*, Zurich, Switzerland, April 2010

- D. Rossi, C. Testa, S. Valenti, Yes, we LEDBAT: Playing with the new BitTorrent congestion control algorithm, In *Passive and Active Measurement (PAM) 2010*, Zurich, Switzerland, April 2010

- D. Rossi, E. Sottile, S. Valenti and P. Veglia, Gauging the network friendliness of P2P applications, In *SIGCOMM Demo Session*, Barcelona, Spain, August 2009.

- S. Valenti, D. Rossi, M. Meo, M.Mellia and P. Bermolen, Accurate and Fine-Grained Classification of P2P-TV Applications by Simply Counting Packets, In *Traffic Measurement and Analysis (TMA) Workshop at IFIP Networking'09*, Aachen, Germany, May 2009

- S. Valenti, D. Rossi, M. Meo, M. Mellia and P. Bermolen, An Abacus for P2P-TV traffic classification, In *IEEE INFOCOM 2009*, *Demo Session*, Rio de Janeiro, Brazil, April 2009

## A.2 Under review

- D. Rossi, D. Tammaro, S. Valenti, A. Pescapé. Exploiting packet sampling measurements for traffic characterization and classification, *submitted to Internation Journal of Network Monitoring*.

- G. Carofiglio, L. Muscariello, D. Rossi, C. Testa, S. Valenti. Rethinking the Low Extra Delay Background Transport (LEDBAT) protocol. *2nd revision submitted to IEEE Transaction on Networking*.

# Appendix B

# List of traffic features output by `tstat`

Here we report the complete list of traffic features calculated by `tstat` . These features have been used in Chap. 7 to analyze the impact of sampling on traffic measurement and classification. We report single-flow features in Tab. B.1 as well as aggregated features in Tab. B.2.

Table B.1: TCP flow-level features.

| Type | Feature | Directions |
|---|---|---|
| Flow ID | IP address | √ |
| | TCP port | √ |
| Volumes | packets | √ |
| | unique-bytes | √ |
| | data-packets | √ |
| | data-bytes | √ |
| | rexmit-packets | √ |
| | rexmit-bytes | √ |
| | out-seq-packets | √ |
| | SACK-req | √ |
| | SACK-sent | √ |
| Packet Size | MSS | √ |
| | max-seg-size | √ |
| | min-seg-size | √ |
| Window Size | RFC1323-ws | √ |
| | RFC1323-ts | √ |
| | window-scale | √ |
| | win-max | √ |
| | win-min | √ |
| | win-zero | √ |
| | cwin-max | √ |
| | cwin-min | √ |
| | initial-cwin | √ |
| Flag counts | ACK-sent | √ |
| | PURE-ACK-sent | √ |
| | SYN-sent | √ |
| | FIN-sent | √ |
| | RST-sent | √ |
| Time | Average-rtt | √ |
| | rtt-min | √ |
| | rtt-max | √ |
| | Stdev-rtt | √ |
| | rtt-count | √ |
| | ttl-min | √ |
| | ttl-max | √ |
| Congestion Control | rtx-RTO | √ |
| | rtx-FR | √ |
| | reordering | √ |
| | duplicates | √ |
| | other anomalies | √ |
| | flow-control | √ |
| | unnece-rtx-RTO | √ |
| | unnece-rtx-FR | √ |
| Flow duration | Completion-time | |
| | First-time | |
| | Last-time | |
| | first-payload | |
| | last-payload | |

Table B.2: Aggregated features. ⋆ features are available for incoming, outgoing and local traffic.

| | | |
|---|---|---|
| IP (single datagram) | ip_tos⋆ | TOS field |
| | ip_ttl⋆ | TTL field |
| | ip_len⋆ | Packet length [byte] |
| | ip_bitrate⋆ | Bitrate [kbit/s] |
| | ip_protocol⋆ | Protocol type |
| UDP (single segment) | udp_port_flow_dst | Destination port per flow |
| | udp_port_dst⋆ | Destination port per segment |
| | udp_tot_time | Flow lifetime [ms] |
| | udp_cl_b_l⋆ | Flow length [byte], coarse granularity |
| | udp_cl_b_s⋆ | Flow length [byte], fine granularity |
| | udp_cl_p⋆ | Flow length [packet] |
| TCP (single segment) | tcp_mss_used | Negotiated MSS |
| | tcp_mss_b | MSS declared by Server |
| | tcp_mss_a | MSS declared by Client |
| | tcp_opts_TS | Timestamp option |
| | tcp_opts_WS | WindowScale option |
| | tcp_opts_SACK | SACK option |
| | tcp_bitrate⋆ | Application bitrate |
| | tcp_port_syndst⋆ | Destination port (SYN segments only) |
| | tcp_port_synsrc⋆ | Source port (SYN segments only) |
| | tcp_port_dst⋆ | Destination port (all segments) |
| | tcp_port_src⋆ | Source port (all segments) |
| TCP (mutliple segments) | tcp_interrupted | Early interrupted flows[152] |
| | tcp_thru ⋆ | Application throughput [Kbps] |
| | tcp_tot_time | Flow lifetime |
| | tcp_rtt_cnt | RTT: number of samples |
| | tcp_rtt_stdev | RTT: standard deviation [ms] |
| | tcp_rtt_max | RTT: maximum RTT [ms] |
| | tcp_rtt_avg | RTT: average RTT [ms] |
| | tcp_rtt_min | RTT: minimum RTT [ms] |
| | tcp_cl_b_l | Flow length, coarse granularity [byte] |
| | tcp_cl_b_s | Flow length, fine granularity [byte] |
| | tcp_cl_p | Flow length [packet] |
| | tcp_cwnd | TCP in-flight-size [byte] |
| | tcp_win_max | TCP max RWND [byte] |
| | tcp_win_avg | TCP average RWND [byte] |
| | tcp_win_ini | TCP initial RWND [byte] |
| | tcp_anomalies ⋆ | TCP anomalies as defined in [126] |
| RTCP (mutliple segments) | rtcp_bt⋆ | Average bitrate [bit/s] |
| | rtcp_mm_bt⋆ | Associated MM flow bitrate[kbit/s] |
| | rtcp_mm_cl_b⋆ | Associated MM flow length [bytes] |
| | rtcp_mm_cl_p⋆ | Associated MM flow length [packets] |
| | rtcp_t_lost⋆ | Lost packets per flow |
| | rtcp_f_lost⋆ | Prob. of lost packets |
| | rtcp_dup⋆ | Duplicated packets |
| | rtcp_lost⋆ | Lost packets |
| | rtcp_avg_inter⋆ | Average inter-packet gap (IPG) |
| | rtcp_jitter⋆ | Average jitter |
| | rtcp_rtt⋆ | RTCP Round trip time [ms] |
| | rtcp_cl_b⋆ | RTCP flow length [bytes] |
| | rtcp_cl_p⋆ | RTCP flow length [packets] |
| RTP multimedia (mutliple segments) | mm_burst_loss⋆ | Burst length of lost packets [packet] |
| | mm_p_late⋆ | Prob. of late packets |
| | mm_p_lost⋆ | Prob. of lost packets |
| | mm_p_dup⋆ | Prob. of duplicate packets |
| | mm_p_oos⋆ | Prob. of out-of-sequence packets |
| | mm_n_oos⋆ | Length of out-of-sequence burst |
| | mm_oos_p⋆ | Total out-of-sequence packets |
| | mm_reord_p_n⋆ | Total reordered packets |
| | mm_reord_delay⋆ | Delay of reordered packets |
| | mm_avg_jitter⋆ | Average jitter [ms] |
| | mm_avg_ipg⋆ | Average IPG [ms] |
| | mm_avg_bitrate⋆ | Stream bitrate [kbit/s] |
| | mm_cl_b⋆ | Long stream flow length [bytes] |
| | mm_cl_p⋆ | Long stream flow length [packet] |
| | mm_cl_b_s⋆ | Short stream flow length [bytes] |
| | mm_cl_p_s⋆ | Short stream flow length [packet] |
| | mm_tot_time_s⋆ | Short stream flow lifetime [ms] |
| | mm_tot_time⋆ | Stream flow lifetime [s] |
| | mm_rtp_pt⋆ | RTP payload type |
| | mm_uni_multi⋆ | Unicast/multicast flows |
| | mm_type⋆ | Stream type |

# Bibliography

[1] Aitia. http://www.aitia.hu.

[2] Auckland-vi. http://www.wand.net.nz/wits/auck/6/auckland_vi.php.

[3] CAIDA, The Cooperative Association for Internet Data Analysis. http://www.caida.org/research/traffic-analysis/classification-overview/.

[4] Endace. http://www.endace.com.

[5] IANA, List of assigned port numbers. http://www.iana.org/assignments/port-numbers,.

[6] IPP2P home page. http://www.ipp2p.org/.

[7] Joost. http://www.joost.com.

[8] l7filter, Application layer packet classifier for Linux. http://l7-filter.clearfoundation.com/,.

[9] LEDBAT Mailing List Archives. http://www.ietf.org/mail-archive/web/ledbat,.

[10] LEDBAT ns2 code. http://perso.telecom-paristech.fr/~valenti/pmwiki/pmwiki.php?n=Main.LEDBAT.

[11] PPLive. http://www.pplive.com.

[12] RIPE's Routing Information Service Raw Data. http://data.ris.ripe.net/.

[13] SOPCast. http://www.sopcast.com.

[14] tcptrace - Official Homepage. http://jarok.cs.ohiou.edu/software/tcptrace/manual.html.

[15] The Network Simulator ns2. http://www.isi.edu/nsnam/ns/.

[16] Tstat, http://tstat.tlc.polito.it.

[17] TVAnts. http://www.tvants.com.

[18] Unibs traces. http://www.ing.unibs.it/ntw/tools/traces/.

[19] Univ. Brescia traces. http://www.ing.unibs.it/ntw/tools/traces/.

[20] Weka, http://www.cs.waikato.ac.nz/ml/weka/.

[21] Comcast throttles bittorrent, seeding impossible. `http://torrentfreak.com/comcast-throttles-bittorrent-traffic-seeding-impossible/`, Aug 2007.

[22] A. Abu and S. Gordon. A Dynamic Algorithm for Stabilising LEDBAT Congestion Window. In *2nd IEEE International Conference on Computer and Network Technology (ICCNT 2010)*, Bangkok, Thailand, Apr 2010.

[23] A. J. Abu and S. Gordon. Impact of Delay Variability on LEDBAT Performance. *Advanced Information Networking and Applications, International Conference on*, pages 708–715, 2011.

[24] Adobe. Real-time media flow protocol. `http://www.adobe.com/products/flashmediaserver/rtmfp_faq/`,.

[25] S. Alcock and R. Nelson. Application flow control in youtube video streams. *SIGCOMM Comput. Commun. Rev.*, 41:24–30, April 2011.

[26] E. Alessandria, M. Gallo, E. Leonardi, M. Mellia, and M. Meo. P2P-TV Systems under Adverse Network Conditions: A Measurement Study. In *INFOCOM 2009, IEEE*, 2009.

[27] M. Alllman and V. Paxson. Issues and etiquette concerning use of shared measurement data. In *Proceedings of the 7th ACM SIGCOMM conference on Internet measurement*, IMC ’07, pages 135–140, New York, NY, USA, 2007. ACM.

[28] P. Amer and L. Cassel. Management of sampled real-time network measurements. In *Proc. of IEEE LCN ’89*, Oct 1989.

[29] M. I. Andreica, N. Tapus, and J. Pouwelse. Performance evaluation of a Python implementation of the new LEDBAT congestion control algorithm. *International Conference on Automation, Quality and Testing, Robotics*, 2:1–6, 2010.

[30] F. Bakerand, B. Fosterand, and C. Sharp. Cisco Architecture for Lawful Intercept in IP Networks. IETF RFC 3924(Informational), Oct 2004.

[31] R. Bennett. The next Internet meltdown. `http://www.theregister.co.uk/2008/12/01/richard_bennett_utorrent_udp`, Dec 2008.

[32] P. Bermolen, M. Mellia, M. Meo, D. Rossi, and S. Valenti. Abacus: Accurate behavioral classification of P2P-TV traffic. *Elsevier Computer Networks*, 55(6):1394 – 1411, 2011.

[33] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian. Traffic classification on the fly. *SIGCOMM CCR*, 36(2):23–26, 2006.

[34] L. Bernaille, R. Teixeira, and K. Salamatian. Early application identification. In *Proc. of ACM CoNEXT 2006*, Lisboa, PT, December 2006.

[35] A. R. Bharambe, C. Herley, and V. N. Padmanabhan. Analyzing and Improving a BitTorrent Networks Performance Mechanisms. In *IEEE INFOCOM’06*, Barcelona, Spain, Apr 2006.

[36] A. Bhattacharyya. On a Measure of Divergence Between Two Statistical Populations Defined by Probability Distributions. *Bull. Calcutta Math. Soc.*, 35:99–109, 1943.

[37] R. Bindal, P. Cao, W. Chan, J. Medved, G. Suwala, T. Bates, and A. Zhang. Improving Traffic Locality in BitTorrent via Biased Neighbor Selection. In *IEEE ICDCS ’06*, Lisboa, Portugal, Jul 2006.

[38] D. Bonfiglio, M. Mellia, M. Meo, and D. Rossi. Detailed Analysis of Skype Traffic. *IEEE Transaction on Multimedia*, 11(1), Jan 2009.

[39] D. Bonfiglio, M. Mellia, M. Meo, D. Rossi, and P. Tofanelli. Revealing skype traffic: when randomness plays with you. In *ACM SIGCOMM'07*, Kyoto, Japan, August 2007.

[40] C. Boutremans, C. Boutremans, and J.-Y. L. Boudec. A note on the Fairness of TCP Vegas. In *In Proceedings of International Zurich Seminar on Broadband Communications*, pages 163–170, 2000.

[41] L. Brakmo, S. O'Malley, and L. Peterson. TCP Vegas: new techniques for congestion detection and avoidance. *ACM SIGCOMM Comp. Comm. Rev.*, 24(4):24–35, 1994.

[42] D. Brauckhoff, B. Tellenbach, A. Wagner, M. May, and A. Lakhina. Impact of packet sampling on anomaly detection metrics. In *Proc. of ACM SIGCOMM IMC '06*, Rio de Janeriro, Brazil, Oct 2006.

[43] V. Carela-Español, P. Barlet-Ros, A. Cabellos-Aparicio, and J. Solé-Pareta. Analysis of the impact of sampling on netflow traffic classification. *Computer Networks*, 55:1083–1099, April 2011.

[44] G. Carofiglio, L. Muscariello, D. Rossi, and C. Testa. A hands-on Assessment of Transport Protocols with Lower than Best Effort Priority. In *IEEE LCN'10*, Denver, CO, USA, Oct 2010.

[45] G. Carofiglio, L. Muscariello, D. Rossi, C. Testa, and S. Valenti. Rethinking the Low Extra Delay Background Transport (LEDBAT) Protocol. submitted to ACM/IEEE Transactions on Networking.

[46] G. Carofiglio, L. Muscariello, D. Rossi, and S. Valenti. The quest for LEDBAT fairness. In *IEEE Global Communication (GLOBECOM 2010)*, Miami, FL, Dec 2010.

[47] N. Cascarano, F. Risso, A. Este, F. Gringoli, L. Salgarelli, A. Finamore, and M. Mellia. Comparing P2PTV Traffic Classifiers. In *Communications (ICC), 2010 IEEE International Conference on*, pages 1 –6, may 2010.

[48] M. W. D. Center. TCP Receive Window Size and Window Scaling. `http://msdn.microsoft.com/en-us/library/ms819736.aspx`.

[49] Y. Chabchoub, C. Fricker, F. Guillemin, and P. Robert. Deterministic versus probabilistic packet sampling in the Internet. In *Managing Traffic Performance in Converged Networks(LNCS)*, Ottawa, Canada, Sep. 07.

[50] C. Chang and C. Lin. LIBSVM: A Library for Support Vector Machines. `http://www.csie.ntu.edu.tw/~cjlin/libsvm`.

[51] H. Chang, S. Jamin, and W. Wang. Live streaming performance of the Zattoo network. In *Proceedings of the 9th ACM SIGCOMM conference on Internet measurement conference*, IMC '09, pages 417–429, Chicago, Illinois, USA, 2009.

[52] D. Chiu and R. Jain. Analysis of the increase and decrease algorithms for congestion avoidance in computer networks. *Computer Networks and ISDN systems*, 17(1):1–14, 1989.

[53] B. Choi, J. Park, and Z. Zhang. Adaptive random sampling for load change detection. In *Proc. of ACM SIGMETRICS '02*, Marina Del Rey, CA, US, Jun 2002.

[54] D. Ciullo, M. Garcia, A. Horvath, E. Leonardi, M. Mellia, D. Rossi, M. Telek, and P. Veglia. Network awareness of p2p live streaming applications: A measurement study. *Multimedia, IEEE Transactions on*, 12(1):54 –63, Jan 2010.

[55] K. C. Claffy, G. C. Polyzos, and H. Braun. Application of sampling methodologies to network traffic characterization. In *Proc. of ACM SIGCOMM '93*, San Francisco, CA, USA, Sep 1993.

[56] B. Claise. Cisco Systems NetFlow Services Export Version 9. RFC 3954 (Informational), Oct 2004.

[57] B. Claise. Specification of the IP Flow Information Export Protocol for the Exchange of IP Traffic Flow Information. RFC 5101, Jan 2008.

[58] B. Cohen. Incentives build robustness in BitTorrent. In *Workshop on Economics of Peer-to-Peer systems*, Cambridge, MA, Jun 2003.

[59] B. Cohen and A. Norberg. Correcting for clock drift in uTP and LEDBAT. In *Invited talk at 9th USENIX International Workshop on Peer-to-Peer Systems (IPTPS 2010)*, San Jose, CA, Apr 2010.

[60] C. Cortes and V. Vapnik. Support-vector networks. *Machine Learning*, 20:273–297, 1995.

[61] T. M. Cover and J. A. Thomas. *Elements of information theory*. Wiley, New York, 1991.

[62] N. Cristianini and J. Shawe-Taylor. *An introduction to Support Vector Machines and Other Kernel-based Learning Methods*. Cambridge University Press, New York, NY, 1999.

[63] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):5–16, January 2007.

[64] A. Dainotti, W. de Donato, and A. Pescapé. Tie: A community-oriented traffic classification platform. In *Traffic Monitoring and Analysis*, volume 5537 of *Lecture Notes in Computer Science*, pages 64–74. 2009.

[65] L. De Cicco, S. Mascolo, and V. Palmisano. Skype video responsiveness to bandwidth variations. In *ACM NOSSDAV '08*, Braunschweig, Germany, May 2008.

[66] L. Deri. nProbe: an Open Source NetFlow Probe for Gigabit Networks. In *In Proc. of Terena TNC 2003*, Zagreb, Croatia, 2003.

[67] J. Drobisz and K. J. Christensen. Adaptive sampling methods to determine network traffic statistics including the hurst parameter. In *Proc. IEEE LCN '08*, Boston, USA, Oct 1998.

[68] N. Duffield. Sampling for passive internet measurement: A review. *Statistical Science*, 19:472–498, 2004.

[69] N. Duffield, C. Lund, and M. Thorup. Properties and prediction of flow statistics from sampled packet streams. In *Proc. of ACM SIGCOMM IMW '02*, Marseille, France, Nov 2002.

[70] N. G. Duffield and M. Grossglauser. Trajectory sampling for direct traffic observation. *SIGCOMM CCR*, 30(4):271–282, 2000.

[71] J. Erman, M. Arlitt, and A. Mahanti. Traffic classification using clustering algorithms. In *MineNet '06: Mining network data (MineNet) Workshop at ACM SIGCOMM '06*, Pisa, Italy, 2006.

[72] J. Erman, A. Mahanti, and M. Arlitt. Byte me: The case for byte accuracy in traffic classification. In *In SIGMETRICS '07 MineNet Workshop*, 2007.

[73] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson. Offline/realtime traffic classification using semi-supervised learning. *Perform. Eval.*, 64(9-12):1194–1213, 2007.

[74] J. Erman, A. Mahanti, M. Arlitt, and C. Williamson. Identifying and discriminating between web and peer-to-peer traffic in the network core. In *Proceedings of the 16th international conference on World Wide Web*, WWW '07, pages 883–892, Banff, Alberta, Canada, 2007.

[75] A. Este, F. Gringoli, and L. Salgarelli. On the stability of the information carried by traffic flow features at the packet level. *ACM SIGCOMM Comput. Commun. Rev.*, 39(3):13–18, 2009.

[76] A. Finamore, M. Mellia, M. Meo, M. Munafo, and D. Rossi. Experiences of internet traffic monitoring with tstat. *IEEE Network Magazine, Special Issue on Network Traffic Monitoring and Analysis*, May 2011.

[77] A. Finamore, M. Mellia, M. Meo, and D. Rossi. Kiss: Stochastic packet inspection classifier for udp traffic. *IEEE/ACM Trans. Netw.*, 18(5):1505–1515, 2010.

[78] A. Finamore, M. Mellia, M. Meo, D. Rossi, and S. Valenti. Peer-to-peer traffic classification: exploiting human communication dynamics. In *IEEE Globecom'10, Demo Session,*, Miami, FL, USA, 2010.

[79] A. Finamore, M. Meo, D. Rossi, and S. Valenti. Kiss to Abacus: A Comparison of P2P-TV Traffic Classifiers. In *Traffic Monitoring and Analysis, Springer Lecture Notes in Computer Science*, volume 6003, pages 115–126. 2010.

[80] T. Z. J. Fu, Y. Hu, X. Shi, D.-M. Chiu, and J. C. S. Lui. PBS: Periodic Behavioral Spectrum of P2P Applications. In *Proc. of PAM '09*, Seoul, South Korea, Apr 2009.

[81] Q. Gao and Q. Yin. Adaptive Vegas: A Solution of Unfairness Problem for TCP Vegas. In *Information Networking. Convergence in Broadband and Mobile Networking, Springer Lecture Notes in Computer Science*, volume 3391, pages 132–141. 2005.

[82] F. Gringoli, L. Salgarelli, M. Dusi, N. Cascarano, F. Risso, and k. c. claffy. GT: picking up the truth from the ground for internet traffic. *ACM SIGCOMM Comput. Commun. Rev.*, 39(5):12–18, 2009.

[83] P. Haffner, S. Sen, O. Spatscheck, and D. Wang. ACAS: automated construction of application signatures. In *ACM SIGCOMM Workshop on Mining Network Data (Minenet'05)*, Philadelphia, PA, August 2005.

[84] G. Hasegawa, M. Murata, and H. Miyahara. Fairness and stability of congestion control mechanisms of TCP. *Telecommunication Systems*, 15:167–184, 2000.

[85] G. Hazel. uTorrent Transport Protocol library. `http://github.com/bittorrent/libutp`, May 2010.

[86] X. Hei, C. Liang, J. Liang, Y. Liu, and K. W. Ross. A Measurement Study of a Large-Scale P2P IPTV System. *IEEE Transactions on Multimedia*, Dec. 2007.

[87] S. Hemminger et al. Network emulation with NetEm. In *Linux Conf Australia (LCA 2005)*, Canberra, Australia, Apr 2005.

[88] E. A. Hernandez, M. C. Chidester, and A. D. George. Adaptive sampling for network management. *J. Netw. Syst. Manage.*, 9(4):409–434, 2001.

[89] G. Huang. Experiences with pplive, 2007. Keynote speech at ACM SIGCOMM'07 Workshop on P2P-TV.

[90] Y. Huang, T. Z. J. Fu, D. M. Chiu, J. C. S. Lui, and C. Huang. Challenges, design and analysis of a large-scale p2p-vod system. In *Proc. of SIGCOMM '08*, Seattle, WA, USA, 2008.

[91] M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, and G. Varghese. Network monitoring using traffic dispersion graphs (tdgs). In *Proc. of IMC '07*, San Diego, California, USA, 2007.

[92] Ipoque. Internet Study 2007. `http://www.ipoque.com/resources/internet-studies/internet-study-2007`.

[93] M. Izal, G. Urvoy-Keller, E. Biersack, P. Felber, A. Al Hamra, and L. Garces-Erice. Dissecting bittorrent: Five months in a torrent's lifetime. In *Passive and Active Measurement (PAM 2004)*, Antibes, France, Apr 2004.

[94] V. Jacobson. Congestion avoidance and control. In *ACM SIGCOMM Comp. Comm. Rev.*, Stanford, CA, Aug 1988.

[95] V. Jacobson, R. Braden, and D. Borman. TCP Extensions for High Performance. IETF RFC 1323, May 1992.

[96] T. Jebara and R. Kondor. Bhattacharyya and Expected Likelihood Kernels. In *Proc. of Conference on Learning Theory (COLT'03)*, Washington D.C., US, August 2003.

[97] H. Jiang, A. W. Moore, Z. Ge, S. Jin, and J. Wang. Lightweight application classification for network management. In *Proceedings of the 2007 SIGCOMM workshop on Internet network management*, Kyoto, Japan, Aug 2007.

[98] Y. Jin, N. Duffield, P. Haffner, S. Sen, and Z.-L. Zhang. Inferring applications at the network layer using collective traffic statistics. *SIGMETRICS Perform. Eval. Rev.*, 38, June 2010.

[99] T. Kailath. The Divergence and Bhattacharyya Distance Measures in Signal Selection. *IEEE Transactions on Communication Technology*, 15(1):52 –60, 1967.

[100] T. Karagiannis, A. Broido, N. Brownlee, k. klaffy, and M. Faloutsos. Is P2P dying or just hiding? In *IEEE GLOBECOM '04.*, Dallas, Texas, US, 2004.

[101] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy. Transport layer identification of P2P traffic. In *4th ACM SIGCOMM Internet Measurement Conference (IMC'04)*, Taormina, IT, October 2004.

[102] T. Karagiannis, K. Papagiannaki, N. Taft, and M. Faloutsos. Profiling the end host. In *Proceedings of the 8th international conference on Passive and active network measurement*, PAM'07, Louvain-la-Neuve, Belgium, 2007.

[103] P. Key, L. Massoulié, and B. Wang. Emulating low-priority transport at the application layer: a background transfer service. In *ACM SIGMETRICS*, New York City, NY, Jun 2004.

[104] A. R. Khakpour and A. X. Liu. High-speed flow nature identification. In *Proceedings of the 2009 29th IEEE International Conference on Distributed Computing Systems*, ICDCS '09, 2009.

[105] H. Kim, K. Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee. Internet traffic classification demystified: myths, caveats, and the best practices. In *Proc. of ACM CoNEXT 2008*, Madrid, Spain, 2008.

[106] R. Kohavi and R. Quinlan. Decision tree discovery. In *IN HANDBOOK OF DATA MINING AND KNOWLEDGE DISCOVERY*, pages 267–276. University Press, 1999.

[107] S. B. Kotsiantis. Supervised machine learning: A review of classification techniques. In *Proceeding of the 2007 conference on Emerging Artificial Intelligence Applications in Computer Engineering: Real Word AI Systems with Applications in eHealth, HCI, Information Retrieval and Pervasive Technologies*, pages 3–24, Amsterdam, The Netherlands, The Netherlands, 2007. IOS Press.

[108] Y. Kulbak and D. Bickson. The eMule protocol specification. Technical Report TR-2005-03, Leibniz Center, 2005.

[109] A. Kumar and J. Xu. Sketch guided sampling - using on-line estimates of flow size for adaptive data collection. In *IEEE INFOCOM '06*, Barcelona, Spain, April 2006.

[110] S. Kumar and P. Crowley. Algorithms to accelerate multiple regular expressions matching for deep packet inspection. In *In Proceedings of the Annual Conference of the ACM Special Interest Group on Data Communication (SIGCOMM'06)*, pages 339–350, 2006.

[111] A. Kuzmanovic and E. Knightly. TCP-LP: low-priority service via end-point congestion control. *IEEE/ACM Transactions on Networking (TON)*, 14(4):752, 2006.

[112] C. Labovitz, S. Iekel-Johnson, D. McPherson, J. Oberheide, and F. Jahanian. Internet inter-domain traffic. *SIGCOMM Comput. Commun. Rev.*, 40:75–86, August 2010.

[113] A. Lakhina, M. Crovella, and C. Diot. Diagnosing network-wide traffic anomalies. In *ACM SIGCOMM '04*, Portland, OR, USA, 2004.

[114] D. Leith, R. Shorten, G. McCullagh, L. Dunn, and F. Baker. Making Available Base-RTT for Use in Congestion Control Applications. *IEEE Communications Letters*, 12:429–431, Jun 2008.

[115] E. Leonardi, M. Mellia, A. Horvath, L. Muscariello, S. Niccolini, and D. Rossi. Building a cooperative P2P-TV application over a wise network: the approach of the European FP-7 strep NAPA-WINE. *Communications Magazine, IEEE*, 46(4):20–22, 2008.

[116] W. Li, M. Canini, A. W. Moore, and R. Bolla. Efficient application identification and the temporal and spatial stability of classification schema. *Computer Networks*, 53(6):790–809, 2009.

[117] S. Liu, T. Basar, and R. Srikant. TCP-Illinois: A loss-and delay-based congestion control algorithm for high-speed networks. *ACM Performance Evaluation*, 65(6-7):417–440, 2008.

[118] S. Liu, M. Vojnovic, and D. Gunawardena. 4cp: Competitive and considerate congestion control protocol. In *ACM SIGCOMM*, Pisa, Italy, Sep 2006.

[119] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker. Unexpected means of protocol inference. In *6th ACM SIGCOMM Internet Measurement Conference (IMC'06)*, Rio de Janeiro, BR, October 2006.

[120] J. Mai, C. Chuah, A. Sridharan, T. Ye, and H. Zang. Is sampled data sufficient for anomaly detection? In *Proc. ACM SIGCOMM IMC '06*, Rio de Janeriro, Brazil, Oct 2006.

[121] G. L. Mantia, D. Rossi, A. Finamore, M. Mellia, and M. Meo. Stochastic Packet Inspection for TCP Traffic. In *IEEE International Conference on Communications (ICC'10)*, Cape Town, South Africa, May 2010.

[122] P. Marciniak, N. Liogkas, A. Legout, and E. Kohler. Small Is Not Always Beautiful. In *IPTPS'2008*, Tampa Bay, Florida United States, 2008.

[123] K. Matusita. A Distance and Related Statistics in Multivariate Analysis. In *Proc. of International Symposium on Multivariate Analysis, Academic Press P.R. Krishnaiah (ed.)*, pages 187–200, 1966.

[124] A. Mcgregor, M. Hall, P. Lorier, and J. Brunskill. Flow clustering using machine learning techniques. In *PAM'04*, Antibes Juan-les-Pins, Fr., April 2004.

[125] M. Mellia, R. Lo Cigno, and F. Neri. Measuring IP and TCP behavior on edge nodes with Tstat. *Computer Networks*, 47(1):1–21, January 2005.

[126] M. Mellia, M. Meo, L. Muscariello, and D. Rossi. Passive analysis of TCP anomalies. *Elsevier Computer Networks*, 52(14), October 2008.

[127] T. M. Mitchell. *Machine Learning*. McGraw-Hill, New York, 1997.

[128] J. Mo, R. J. La, V. Anantharam, and J. Walrand. Analysis and Comparison of TCP Reno and Vegas. In *In Proceedings of IEEE Infocom*, New York, NY, USA, Mar 1999.

[129] A. Moore, D. Zuev, and M. Crogan. Discriminators for use in flow-based classification. Technical report, University of Cambridge, 2005.

[130] A. W. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. In *ACM SIGMETRICS '05*, Banff, Alberta, Canada, 2005.

[131] D. Moore, K. Keys, R. Koga, E. Lagache, and K. C. Claffy. The coralreef software suite as a tool for system and network administrators. In *Proceedings of the 15th USENIX conference on System administration*, San Diego, California, 2001.

[132] Moore, Andrew. W. and Papagiannaki, Konstantina. Toward the Accurate Identification of Network Applications. In *Passive and Active Measurement (PAM'05)*, Boston, MA, US, March 2005.

[133] T. Mori, M. Uchida, R. Kawahara, J. Pan, and S. Goto. Identifying elephant flows through periodically sampled packets. In *Proc. of ACM SIGCOMM IMC '04*, Taormina, Italy, 2004.

[134] S. Morris. $\mu$Torrent release 1.9 alpha 13485. `http://forum.utorrent.com/viewtopic.php?pid=379206#p379206`, Dec 2008.

[135] T. T. T. Nguyen and G. Armitage. A survey of techniques for internet traffic classification using machine learning. *IEEE Communications Surveys & Tutorials*, 10(4):56–76, 2008.

[136] A. Norberg. BitTorrent Enhancement Proposals on uTorrent transport protocol. `http://www.bittorrent.org/beps/bep_0029.html`, 2009.

[137] I. Paredes-Oliva, P. Barlet-Ros, and J. Solé-Pareta. Portscan detection with sampled netflow. In *Traffic Measurement and Analysis (TMA), Springer-Verlag LNCS 5537*, May 2009.

[138] J. Park, H.-R. Tyan, and C.-C. Kuo. Internet Traffic Classification for Scalable QoS Provision. pages 1221 –1224, jul. 2006.

[139] V. Paxson. End-to-end routing behavior in the internet. *SIGCOMM CCR*, 26(4):25–38, 1996.

[140] V. Paxson. Bro: a system for detecting network intruders in real-time. *Elsevier Comput. Netw.*, 31:2435–2463, December 1999.

[141] M. Perényi, T. D. Dang, A. Gefferth, and S. Molnár. Identification and analysis of peer-to-peer traffic. *Journal of Communications*, 1(7):36–46, 2006.

[142] A. Pescapé, D. Rossi, D. Tammaro, and S. Valenti. On the impact of sampling on traffic monitoring and analysis. In *International Teletraffic Congress ITC22*, Sep. 2010.

[143] L. Peterson. Inter-AS traffic patterns and their implications. In *in Proc. IEEE GLOBECOM*, 1999.

[144] M. Pietrzyk, J.-L. Costeux, G. Urvoy-Keller, and T. En-Najjary. Challenging statistical classification for operational usage: the ADSL case. In *Proc. of IMC '09*, pages 122–135, Chicago, Illinois, USA, 2009.

[145] D. Qiu and R. Srikant. Modeling and performance analysis of BitTorrent-like peer-to-peer networks. In *ACM SIGCOMM'04*, Portland, Oregon, USA, Aug 2004.

[146] B. Ribeiro, D. Towsley, T. Ye, and J. C. Bolot. Fisher information of sampled packets: an application to flow size estimation. In *Proceedings of the 6th ACM SIGCOMM conference on Internet measurement*, Rio de Janeriro, Brazil, 2006.

[147] F. Risso, M. Baldi, O. Morandi, A. Baldini, and P. Monclus. Lightweight, payload-based traffic classification: An experimental evaluation. In *Proc. of IEEE ICC '08*, May 2008.

[148] F. Risso and N. Cascarano. Diffinder available at `http://netgroup.polito.it/research-projects/l7-traffic-classification`.

[149] M. Robnik-Šikonja and I. Kononenko. Theoretical and Empirical Analysis of ReliefF and RReliefF. *Mach. Learn.*, 53:23–69, October 2003.

[150] V. P. Roche and U. Arronategui. Behavioural characterization for network anomaly detection. *Transactions on Computational Science IV: Special Issue on Security in Computing*, pages 23–40, 2009.

[151] S. Romig, M. Fullmer, and L. Luman. The OSU Flow-tools Package and CISCO NetFlow Logs. In *USENIX LISA'00*, New Orleans, LA, US, Dec 2000.

[152] D. Rossi, C. Casetti, and M. Mellia. User patience and the web: a hands-on investigation. In *IEEE Globecom'03*, San Francisco, CA, USA, December 2003.

[153] D. Rossi, M. Mellia, and M. Meo. Understanding Skype signaling. *Elsevier Computer Networks*, 53(2):130–140, 2009.

[154] D. Rossi and E. Sottile. Sherlock: A framework for P2P traffic analysis. In *IEEE P2P'09*, Seattle, WA, USA, Sep 2009.

[155] D. Rossi, C. Testa, S. Valenti, , and L. Muscariello. LEDBAT: the new BitTorrent congestion control protocol. In *Proc. of ICCCN '10*, Zurich, Switzerland, Aug 2010.

[156] D. Rossi, C. Testa, and S. Valenti. Yes, we LEDBAT: Playing with the new BitTorrent congestion control algorithm. In *Passive and Active Measurement (PAM 2010)*, Zurich, Switzerland, Apr 2010.

[157] D. Rossi, C. Testa, S. Valenti, P. Veglia, and L. Muscariello. News from the internet congestion control world. Technical Report, Aug 2009.

[158] D. Rossi and S. Valenti. Fine-grained traffic classification with Netflow data. In *TRaffic Analysis and Classification (TRAC) Workshop at IWCMC '10*, Caen, France, Jun 2010.

[159] D. Rossi and S. Valenti. Fine-grained behavioral classification in the core: the issue of flow sampling. In *TRaffic Analysis and Classification (TRAC) Workshop at IWCMC '11*, Istambul,Turkey, Jul 2011.

[160] D. Rossi and S. Valenti. Identifying key features for P2P traffic classification. In *IEEE International Conference on Communications (ICC'11)*, Kyoto, Japan, Jun 2011.

[161] M. Roughan, S. Sen, O. Spatscheck, and N. Duffield. Class-of-service mapping for QoS: a statistical signature-based approach to IP traffic classification. In *ACM SIGCOMM Internet Measurement Conference (IMC'04)*, Taormina, IT, October 2004.

[162] I. S. Ha, Rhee and L. Xu. CUBIC: A new TCP-friendly high-speed TCP variant. In *ACM SIGOPS Operating System Review*, New York, NY, Jul 2008.

[163] L. Salgarelli, F. Gringoli, and T. Karagiannis. Comparing traffic classifiers. *ACM SIGCOMM Comp. Comm. Rev.*, 37(3):65–68, 2007.

[164] J. Schneider, J. Wagner, R. Winter, and H. Kolbe. Out of my way - evaluating Low Extra Delay Background Transport in an ADSL access network. In *Teletraffic Congress (ITC), 2010 22nd International*, pages 1 –8, sept. 2010.

[165] S. Sen, O. Spatscheck, and D. Wang. Accurate, scalable in-network identification of p2p traffic using application signatures. In *13th international conference on World Wide Web (WWW'04)*, New York, NY, US, May 2004.

[166] S. Shalunov. Low Extra Delay Background Transport (LEDBAT). IETF Draft, Mar 2010.

[167] T. Silverston and O. Fourmaux. Measuring P2P IPTV Systems. In *Proceedings of ACM NOSSDAV*, June 2007.

[168] R. Sommer and A. Feldmann. NetFlow: Information loss or win? In *ACM SIGCOMM Internet Measurment Workshop (IMW '02)*, marseille, france, Nov 2002.

[169] C. System. Cisco visual networking index: Forecast and methodology, 2010-2015. `http://www.cisco.com/en/US/solutions/collateral/ns341/ns525/ns537/ns705/ns827/white_paper_c11-481360_ns827_Networking_Solutions_White_Paper.html`.

[170] D. Tammaro, S. Valenti, D. Rossi, and A. Pescapé. Exploiting packet sampling measurements for traffic characterization and classification. submitted to International Journal of Network Monitoring.

[171] K. Tan, J. Song, Q. Zhang, and M. Sridharan. A compound TCP approach for high-speed and long distance networks. In *25th IEEE Conference on Computer Communications (INFOCOM 2006)*, Barcelona, Spain, Apr 2006.

[172] P.-N. Tan, M. Steinbach, and V. Kumar. *Introduction to Data Mining, (First Edition)*. Addison-Wesley Longman Publishing Co., Inc., Boston, MA, USA, 2005.

[173] S. Valenti, D. Rossi, M. Meo, M. Mellia, and P. Bermolen. Accurate, Fine-Grained Classification of P2P-TV Applications by Simply Counting Packets. In *Proc. of International Workshop on Traffic Monitoring and Analysis (TMA '09), Springer Lecture Notes on Computer Science*, volume 5537, pages 84–92, Aachen, Germany, 2009.

[174] S. Valenti, D. Rossi, M. Meo, M. Mellia, and P. Bermolen. An Abacus for P2P-TV traffic classification. In *IEEE INFOCOM 2009, Demo Session*, April 2009.

[175] A. Venkataramani, R. Kokku, and M. Dahlin. TCP Nice: A mechanism for background transfers. In *8th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2002)*, Boston, MA, Dec 2002.

[176] F. Verhulst. *Nonlinear differential equations and dynamical systems*. Springer-Verlag, New York, NY, USA, 1990.

[177] N. Williams, S. Zander, and G. Armitage. A preliminary performance comparison of five machine learning algorithms for practical IP traffic flow classification. *ACM SIGCOMM CCR*, 36(5):5–16, 2006.

[178] B. Wong, A. Slivkins, and E. Sirer. Meridian: A lightweight network location service without virtual coordinates. *ACM SIGCOMM Comp. Comm. Rev.*, 35(4):96, 2005.

[179] C. Wu, B. Li, and S. Zhao. Exploring large-scale peer-to-peer live streaming topologies. *ACM Trans. Multimedia Comput. Commun. Appl.*, 4:19:1–19:23, September 2008.

[180] C.-C. Wu, K.-T. Chen, Y.-C. Chang, and C.-L. Lei. Peer-to-peer application recognition based on signaling activity. In *Proc. of IEEE ICC '09*, Dresden, Germany, May 2009.

[181] W. A. Wulf and S. A. Mckee. Hitting the memory wall: Implications of the obvious. *Computer Architecture News*, 23:20–24, 1995.

[182] Y. Xia, D. Harrison, S. Kalyanaraman, K. Ramachandran, and A. Venkatesan. Accumulation-based congestion control. *IEEE/ACM Trans. Netw.*, 13:69–80, February 2005.

[183] H. Xie, A. Krishnamurthy, A. Silberschatz, and Y. R. Yang. P4P: Explicit Communications for Cooperative Control Between P2P and Network Providers.

[184] K. Xu, Z.-L. Zhang, and S. Bhattacharyya. Profiling internet backbone traffic: behavior models and applications. *ACM SIGCOMM Comput. Commun. Rev.*, 35(4):169–180, 2005.

[185] S. Yang, H. Jin, B. Li, and X. Liao. A modeling framework of content pollution in peer-to-peer video streaming systems. *Comput. Netw.*, 53:2703–2715, October 2009.

[186] S. Zander, T. Nguyen, and G. Armitage. Automated traffic classification and application identification using machine learning. In *Proc. of IEEE LCN ’05*, nov. 2005.

[187] X. Zhang, J. Liu, B. Li, and Y.-S. Yum. Coolstreaming/donet: a data-driven overlay network for peer-to-peer live media streaming. In *INFOCOM 2005. 24th Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings IEEE*, 2005.

[188] T. Zseby. Deployment of Sampling Methods for SLA. Validation with Non-Intrusive Measurements. In *Proc. of PAM ’02*, Fort Collins, Colorado, USA, Mar 2002.

[189] T. Zseby, M. Molina, N. Duffield, S. Niccolini, and F. Raspall. Sampling and Filtering Techniques for IP Packet Selection. RFC 5475 (Proposed Standard), Mar 2009.