Improving quality of experience in multimedia streaming by leveraging Information-Centric Networking

Jacques Samain

université PARIS-SACLAY

TELECOM ParisTech

NNT : 2019SACLT012

Thèse de doctorat

Improving quality of experience in multimedia streaming by leveraging Information-Centric Networking

Thèse de doctorat de l'Université Paris-Saclay
préparée à Télécom ParisTech

Ecole doctorale n°580 Sciences et technologies de l'information et de la communication (STIC)
Spécialité de doctorat : Réseaux, information et communications

Thèse présentée et soutenue à Paris, le 19 mars 2019, par

JACQUES SAMAIN

Composition du Jury :

Marco Cagnazzo Professeur, Télécom ParisTech	Président
Michael Zink Maitre de conférences, University of Massachusetts	Rapporteur
Hermann Hellwagner Professeur, Universität Klagenfurt	Rapporteur
Lucile Sassatelli Maitresse de conférences, Université Nice Sophia Antipolis	Examinateur
Gwendal Simon Professeur, IMT Atlantique	Examinateur
Jeff Burke Professeur, University of California Los Angeles	Examinateur
Gérard Memmi Professeur, Télécom ParisTech	Directeur de thèse
Giovanna Carofiglio Distinguished Engineer, Cisco Systems France	Co-encadrant

Improving Quality of Experience in Multimedia Streaming by Leveraging Information-Centric Networking

Jacques Samain

2019

 This work was conducted while the author was employed by Cisco Systems France. It benefited from support from French CIFRE grant #2015/1309.

Abstract

Information-Centric Networking (ICN) is a promising architecture to address today Internet multimedia traffic explosion and increasing user mobility: not only to enhance the user's quality of experience, but also to naturally and seamlessly extend video support deeper in the network functions. However, to the best of our knowledge, a thorough assessment of the benefits brought by ICN to multimedia delivery has not been done yet. In this thesis, we aim at reducing the gap to such assessment, by considering ICN in various multimedia delivery scenarios.

First, we assess the benefits brought by an ICN-based Dynamic Adaptive Streaming (DAS) compared to TCP/IP based streaming, by means of an experimental campaign that includes multiple channels (e.g., emulated Wi-Fi and LTE, real 3G/4G traces), multiple clients (homogeneous vs heterogeneous mixture, synchronous vs asynchronous arrivals) and carefully selected DAS adaptation logics to cover the broad families of available adaptation algorithms. We also warn about potential pitfalls that are nonetheless easily avoidable.

Second, we show how network assistance helps improving the users' quality of experience. To do so, we leverage the in-network caching feature of ICN and propose a simple periodical network signal from the cache (i.e., per-quality hit ratio) to be exploited by DAS adaptation logic to enhance further the user's quality of experience by avoiding the known cache-induced quality oscillations. We confirm the soundness of our approach through experiments.

Finally, as live multimedia delivery is gaining momentum, we propose hICN-RTC by integrating hICN (hybrid ICN), an ICN-in-IP solution, to WebRTC and we design RICTP (Realtime Information Centric Transport Protocol), a content-aware transport that minimizes the communication latency. Although still in development, the results we gathered from early experiments are promising as they show that hICN-RTC scales with the number of active speakers rather than the total number of participants.

Résumé

Les réseaux centrés sur l'information (ICN) sont une architecture prometteuse pour faire face à l'explosion du trafic multimedia sur internet et à la mobilité croissante des utilisateurs: non seulement ICN peut améliorer la qualité d'expérience de l'utilisateur, mais ICN peut également étendre naturelle et de façon transparente la prise en charge du trafic vidéo dans les fonctions réseau. Cependant, à notre connaissance, une évaluation approfondie des avantages apportés par ICN à la diffusion multimédia n'a pas encore été réalisée. Dans cette thèse, nous voulons réduire l'écart qui nous sépare d'une telle évaluation en prenant en compte ICN dans divers scénarios de diffusion multimédia.

Tout d'abord, nous évaluons les avantages apportés par du DAS (Dynamic Adaptive Streaming) basé sur ICN par rapport au streaming basé sur TCP/IP, au moyen d'une campagne expérimentale comprenant plusieurs canaux (des émulations Wi-Fi et LTE, des traçes 3G/4G), plusieurs clients (mélange homogène et hétérogène, arrivées synchrones et asynchrones) et des logiques d'adaptation DAS soigneusement sélectionnées pour couvrir les deux grandes familles d'algorithmes disponibles. Nous mettons aussi en exergue les pièges potentiels qui sont néanmoins facilement évitables.

Ensuite, nous montrons comment l'assistance du réseau contribue à améliorer la qualité d'expérience des ultilisateurs. Pour ce faire, nous tirons parti de la fonctionnalité de mise en cache réseau d'ICN et proposons un signal réseau simple envoyé périodiquement par le cache à exploiter par l'algorithme d'adaptation DAS pour optimiser la qualité d'expérience de l'utilisateur en évitant le phénomène bien connu des oscillations induites par le cache. Des expériences nous permettent de valider le bien-fondé de notre approche.

Enfin, puisque la diffusion multimedia en direct gagne du terrain, nous proposons hICN-RTC, en intégrant hICN (hybrid ICN), une solution ICN-dans-IP, à WebRTC, accompagné du protocole RICTP (Realtime Information Centric Transport Protocol), un protocole de transport basé sur le contenu, qui minimise la latence. Bien que toujours en développement, les résultats des premières expériences sont prometteurs car ils montrent que le trafic induit par hICN-RTC ne croit qu'avec le nombre de locuteurs actifs plutôt qu'avec le nombre total de participants.

Remerciements

Tout d'abord, mes remerciements vont à Dario Rossi, pour son soutien sans faille, ses conseils avisés et son aide précieuse en qualité de directeur de thèse durant la majorité de ces trois ans. Mes remerciements vont aussi à Gérard Memmi qui a pu prendre le relais quand cela fut nécessaire. Je remercie aussi chaudement Giovanna Carofiglio qui m'a permis d'effectuer ma thèse au sein de Cisco, et qui a toujours été disponible et de bon conseil tout au long de ces trois années.

Je remercie mes rapporteurs, Hermann Hellwagner et Michael Zink, d'avoir pris le temps de lire mon manuscrit et d'écrire des rapports aussi détaillés que pertinents. Merci aussi aux autres membres du jury, Jeff Burke, Marco Cagnazzo, Lucille Sassatelli et Gwendal Simon pour leur participation à ma soutenance et aux questions qu'ils m'ont posés.

Je tiens ensuite à remercier les autres thésards qui ont partagé cette aventure avec moi: Yoann Desmouceaux, Guillaume Ruty, Marcel Enguehard et Mohammed Hawari. Toujours le petit mot qui va bien, autour d'un café ou du babyfoot. Merci aussi à toute l'équipe ICN de Cisco de m'avoir accueilli et initié aux joies des démos pour MWC ou CLUS: Luca Muscariello, Jordan Augé, Michele Papalini, Alberto Compagno, Mauro Sardara, Xuan Zeng, Angelo Mantellini, Giovanni Conte et Massoud Hemmatpour. Merci aussi à tout les membres du PIRL avec qui j'ai pu échanger durant ces trois ans, il m'est impossible de tous les nommer ici, mais je pense particulièrement à Alain Fiocco, Guillaume Sauvage de Saint Marc, André Surcouf, Jérome Tollet, Pierre Pfister, Victor Nguyen, Nathan Skrzypcczak, Aloÿs Augustin, Enzo Fenoglio, avec une mention spéciale pour Carole Reynaud et Mark Townsley, sans qui rien n'aurait été possible.

Merci aussi à mes collègues du LINCS pour ces mercredis matins PPT, et tout particulièrement Michele Tortelli pour son aide et son soutien lors de la première partie de ma thèse.

Enfin, je remercie ma famille, ma mère, mon père, mes frères et ma soeur, qui ont été à mes cotés dans tous les moments de cette longue aventure, débutée presque dix ans auparavant sur les bancs de la prépa. Merci aussi à tous ceux qui ont pu venir à ma soutenance, notamment mes grand-parents, ainsi que Charles et Chantal, venu de loin pour y assister. Enfin, merci à tous mes amis pour leur soutien, avec une mention spéciale pour le groupe :D et les adeptes du 20h20 multiple.

Finalement, un immense merci à Claire d'avoir été présente tout au long de ma thèse, de m'avoir supporté pendant ces trois ans, simplement d'avoir partagé ces trois merveilleuses années ensemble.

Contents

List of Figures

Chapter 1

General Introduction

1.1 Video streaming to shape future networks

Video streaming, and more generally multimedia streaming over the internet, has gained tremendous popularity in the last few years, and is most likely to be the predominant application in the future years. Indeed, according to Cisco VNI forecast [104], in 2022, 82% of the IP traffic is expected to be video. This increase of traffic goes hand in hand with ever evolving video services: video contents are now moving from Full HD (FHD) to Ultra HD (UHD, or 4K) or even 8K resolution, while Virtual and Augmented Reality equipments are emerging, with high bandwidth requirements. Furthermore, still according to Cisco VNI forecast, two-third of all Internet traffic will be generated from wireless and mobile devices, supported by heterogeneous and high speed 5G wireless access. This corroborates the observed shift in video consumption: from TV broadcasted programs, we are switching to streaming sessions over connected devices such as smartphones, tablets and laptops relying on social platforms. This consistent move towards online viewing has been illustrated during the last Rio 2016 Olympic games where NBC claimed that "the first six days of [*their*] Rio coverage generated 153.8 million 'social media engagements', 10 times larger than the total of NCAA Basketball March Madness – held over 19 days – and outpaced the 32-day total of the entire 2014 soccer World Cup in Brazil"[1]. These factors, altogether, drive future 5G networks design to meet new mobile video usage with very-high bandwidth requirement under ultra-low latency constraints.

More specifically, these factors highlight the critical role of future 5G networks in the support of Dynamic Adaptive Streaming (DAS). Here, DAS refers to the various techniques that have emerged over the last years to realize an efficient mul-

[1] `https://www.kark.com/road-to-the-olympics/new-multiplatform-media-consumption-drives-nbcs-olympic-strategy/530225621`

timedia delivery over the Internet, leveraging HTTP in most cases: many popular ones are proprietary (e.g., Microsoft HSS, Apple HLS, Adobe HDS), while MPEG Dynamic Adaptive Streaming over HTTP (DASH) recently became a standard. DAS techniques rely on segmenting the video in several segments of equal temporal length and encode each segment at different qualities (e.g., various resolutions, various video bitrate). The client then requests the segments at the quality that best matches its characteristics (both internal, such as her screen resolution, and external, such as network condition). DAS techniques were initially designed for CDN/OTT content delivery, and their interactions with the underlying network has been only superficially studied so far. However, in a 5G environment where mobility and heterogenous network access are important features, DAS interaction with the network along with providing caching and computing capabilities to the network edge should be considered in order to enable efficient mobile video delivery.

In parallel, the Internet usage has significantly evolved over the past decades, resulting in a *mismatch between Internet usage and its architecture*: on the one hand, today's Internet usage is mostly centered on information dissemination and retrieval. Users upload and watch more and more multimedia contents on social platforms, such as YouTube, Facebook or Netflix, turning the network in an instrument to connect people with content. Moreover, the popularity of certain contents (e.g., Olympic Games, the soccer World Cup, the super Bowl, ...) results in having millions of users requesting the same content at the same time. On the other hand, the network architecture still relies on the founding host-to-host IP architecture principles, resulting in an end-to-end model that appears to be unsuited to deal with multimedia delivery.

To overcome this mismatch, over-the-top (OTT) solutions, such as Content Delivery Networks (CDNs) and peer-to-peer networks, have been designed and widely deployed, carrying a large fraction of today's Internet traffic. The drawback of using such overlay models lies in the added complexity in the network ecosystem as many application-layer technologies and players, such as CDN providers, ISP and content providers, are involved, network management is also harder to do from either a technical or business point of view. This complexity is illustrated by the important technical inefficiencies that arise from the overlay model, such as mobility management, dynamic content-to-location bindings, multicast, multihoming, etc.

1.2 ICN: a promising architecture

The tremendous stress put on the network by the ever increasing multimedia data being transmitted over the Internet led, over the last years, to the emergence of a

new research domain, called *Future Internet*, to investigate various approaches to tackle actual network inefficiencies and to handle the traffic increase.

Among the Future Internet proposals, Information-Centric Networking (ICN) offers a new networking paradigm leveraging content-awareness at the network layer. More specifically, ICN is based on location-independent network names: each content is addressed over the network by a unique network name, rather than being addressed using locations (e.g., the IP address of the hosting server). The network operations in ICN are then driven by content names, rather location identifiers and thus enable a user-to-content communication. Addressing the content directly by its network name allows for a more agile connectionless transport model, driven by the end-user and not bounded to a network addressable interface. As a result, this simplifies the management of mobile multi-homed communications and brings to the ISPs a finer-grained control over carried data. Finally, thanks to the content-awareness at the network level, ICN routers are able to route Interest packets (requests for a network name specified in the packet) towards the nearest content replicas, exploiting dynamic in-network caches and adaptive request routing for a more efficient and cost-effective data delivery.

Interestingly enough, DAS techniques and ICN share some characteristics, such as the pull-based approach: in both cases, it is the client that initiates the content retrieval, by sending an Interest packet (ICN case) or an HTTP GET request (DAS case); or the location-independent feature: in DAS, the segments are not necessarily located on the same server and can potentially be retrieved from any location, thus enabling potential multi source locations. To ensure the effective HTTP download of the segment, its location is given to the client through a manifest file, but it is needed by the nature of HTTP transfer over TCP/IP rather than a necessity from DAS techniques. Therefore, ICN appears as a natural candidate to support the evolution of multimedia delivery by leveraging content-aware capabilities at the network level to achieve a joint video/network optimization, as there are several features offered by ICN that are very appealing for DAS, such as network-level caching, seamless mobility support or multi-path forwarding.

1.3 Thesis statement

The contribution of this thesis is threefold. First, ***this thesis proposes a comprehensive comparison of DAS systems over a TCP/IP stack versus an ICN stack***. The goal is here to assess the benefits that can be brought by an ICN networking stack with respect to a TCP/IP one, along with the potential pitfalls to avoid. To do so, as DAS performance are influenced by the adaptation logics, we selected three adaptation logics from the state-of-the-art that are representative of the whole design space, and we then assess the benefits coming from ICN features,

such as enhanced rate adaptation, load-balancing on heterogeneous interfaces and in-network loss detection and recovery. We show (Chapter 3) that DAS performance over an ICN stack can easily match and possibly significantly outperform (especially in a multi-homed scenario) DAS performance over a TCP/IP stack. We also show that the ICN features should be used *jointly* in order to achieve the best performance, while DAS performance over an ICN stack with only selected features (for example an enhanced rate adaptation without in-network loss detection and recovery) would result in a performance comparable or worse than the DAS performance over a TCP/IP stack. This comparison is only focused on the client-side, and does not consider the interactions with the network, other than load-balancing done at the client over heterogeneous interfaces, as it would be the case in a multi-homed client.

Secondly, ***this thesis investigates and proposes a network-assisted approach to DAS***, by investigating further a particular ICN feature, in-network caching, and how it can be leveraged to improve DAS performance. Specifically, we explore (Chapter 4) the boundaries of the design space for network and client interactions, highlighting both the benefits that come from in-network caching and its well-known drawbacks, such as the cache-induced quality oscillations. We then design a network-aware evolution of an existing adaptation logic (AdapTech, [15]), that leverages a simple signal from the network (per-quality cache hit-rates) to take educated decisions at the client-side. We confirm the soundness of this approach by an experimental campaign that shows a significant increase of the DAS performance when using our network-aware enhanced adaptation logic with respect to a network-blind approach, where decisions are taken solely on client information (such as estimated throughput and buffer level).

Third, we investigated DAS usage in an ICN network, but DAS techniques do not fit all the video use cases over the Internet. More specifically, the latency between the content source and the clients induced by a DAS system is typically of, at least, a few seconds, imposed by the chunk granularity. Therefore, DAS techniques are not usually used for ultra low-latency media streaming, when real-time latency is required for both media distribution and live interaction/feedback from the clients, such as online gaming, betting, auctioning services or simply web-conferencing. In such use-cases, WebRTC (Web Real-Time Communication) emerges as a promising option. ***This thesis explores the scalability benefits of integrating ICN to WebRTC***. Indeed, WebRTC was not designed for large scale and thus, to confirm WebRTC as a credible candidate for low-latency multimedia streaming, it requires to have some scalability guarantees. More specifically, we (Chapter 5) design and implement a WebRTC system over hICN (hybrid ICN, an incrementally deployable ICN-in-IP solution [97]), along with a new Realtime Information Centric Transport Protocol (RICTP), a content-aware transport that

minimizes the latency. We show that WebRTC and ICN together provide an architecture that scales with content (i.e., the number of active media streams in a video conference) rather than end hosts (i.e., the number of participants) which is the case with WebRTC over IP.

1.4 List of contributions

Journal paper

- [122] *Jacques Samain*, Giovanna Carofiglio, Luca Muscariello, Michele Papalini, Mauro Sardara, Michele Tortelli, and Dario Rossi. Dynamic adaptive video streaming: Towards a systematic comparison of ICN and TCP/IP. In *IEEE Transactions on Multimedia*, 19(10):2166-2181, 2017.

Workshop paper

- [123] *Jacques Samain*, Giovanna Carofiglio, Michele Tortelli, and Dario Rossi. A simple yet effective network-assisted signal for enhanced dash quality of experience. In *Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 55-60. ACM, 2018. (**Best paper award**).

Conference paper

- [103] Michele Papalini, Giovanna Carofiglio, Alberto Compagno, Angelo Mantellini, Luca Muscariello, *Jacques Samain*, and Mauro Sardara. Scaling WebRTC using hybrid information-centric networking (**under submission**). In ACM ICN, 2019.

Demos

- [121] *Jacques Samain*, Jordan Augé, Giovanna Carofiglio, Luca Muscariello, Michele Papalini, and Mauro Sardara. Enhancing mobile video delivery over an heterogeneous network access with information-centric networking. In *Proceedings of the SIGCOMM Posters and Demos*, pages 22-24. ACM, August 2017.

- Deutsch Telekom Campus Fair, Bonn, Germany, May 2017.

- MWC demo, Barcelona, Spain, February 2017.

- [27] Giovanna Carofiglio, Jordan Augé, Luca Muscariello, Michele Papalini, Jacques Samain, and Mauro Sardara. Using ICN to simplify data delivery, mobility management and secure transmission over an heterogeneous network access. In *FG IMT-2020 Workshop and Demo Day: Wireline Technology Enablers for 5G*. ITU, December 2016.

Talks

- Adaptive Video Streaming: Towards a systematic comparison of ICN and TCP/IP, weekly seminar at LINCS, Paris, France, October 2017.
- Mobile video over ICN, Cisco École Polytechnique Symposium, Paris, France, March 2017.

White Paper

- [26] Giovanna Carofiglio, Jordan Augé, Luca Muscariello, Michele Papalini, and *Jacques Samain*. Mobile video delivery with hybrid ICN. Cisco, White Paper, 2016.

Open-source

- [2] Contributions to the open-source CICN project, especially to VIPER, a dual-stack (ICN and TCP/IP) video player. A more detailed description of VIPER is given in Appendix B.

Chapter 2

Background

Contents

2.1 Video Delivery

As the multimedia demand over the Internet increased over the years, video delivery techniques also evolved to meet with the new video requirements. From broadcast TV, streaming video contents took over the Internet through the years to finally become the predominant application in IP traffic: according to Cisco VNI forecast [104], more than 82% of IP traffic will be video by 2022. Following on from this, the video delivery techniques evolved over the years, to face the increasing demand from the users. Broadly, video streaming over the Internet can be divided into two categories, depending on the nature of the desired content: either Video on Demand (VoD) or live content. In the VoD case, the content is already fully produced (e.g., a movie) while in the live case, the content is produced while the client is watching (e.g., a sport event) and therefore the latency between the source and the client should be kept to a minimum (especially in the case of web-conferencing). The next sections describe further the different delivery protocols that can be used to address the different use-cases.

2.1.1 Video on Demand

For VoD, the latency between the source and the client is not an issue, since the content is already fully produced. There are several techniques to achieve VoD delivery over the Internet. The simplest one is to download the full video at the client side and then play it locally. The video is, in this case, treated as a data file and is downloaded by the client using file transfer protocols such as FTP [112], HTTP [41] or more recently using peer-to-peer protocols, such as BitTorrent [31]. However, these protocols were designed to download the video locally and not to stream the video: with these, the client has to wait for the download to be completed before being able to start watching the video. Therefore, the startup delay, that is, the time elapsed between the decision to watch the video, e.g., the click on the play button, and the first frame of the video being displayed to the user, is equal to, at minimum, the downloading time of the video and therefore can be arbitrary long as it depends on the size of the video file, the network characteristics (available throughput) and the protocol used (especially BitTorrent for non-popular content). Furthermore, this requires to have locally the whole video file, which can be of several GBs for UHD (4K) movies and thus can create storage issues, especially on mobile devices such as smartphones or tablets.

To reduce the startup delay, HTTP progressive download was introduced: by allowing the client to start the video playback before the download is completed, the startup delay is significantly decreased. Naturally, this requires to have a linear download of the video file (for instance, this cannot be achieved using BitTorrent, as it allows to download chunks of video in random order) and it also requires the video being encoded in a video format that allows a linear decoding of the video, meaning that the decoding does not need to make a first pass on the whole video file before starting. Although this reduces the startup delay, therefore bringing the video quicker to the user, it introduces new challenges to address, such as rebuffering events, that can have a tremendous impact on the user's quality of experience, as shown in [53]. Indeed, by starting the playback of the video before the end of the download, the user assumes that the video downloading rate will be equal or exceed the playout rate: for example, while one second of video is played, at least one second worth of video will be downloaded. If the network conditions change (e.g., a drop in the available throughput), the downloading rate can drastically decrease, and thus the player will eventually run out of video data to display, resulting in a rebuffering event: the video is frozen while the player waits for data to be downloaded. Therefore, even though HTTP progressive download efficiently reduces the startup delay, using it in a mobile environment where the available bandwidth fluctuates would lead to rebuffering events during the playback of the video, which would negatively impact the user's quality of experience.

To take into account the network variations (and thus the resulting rebuf-

fering events in HTTP progressive download), the last few years witnessed the introduction of Dynamic Adaptive Streaming (DAS) techniques, proprietary solutions such as Microsoft Smooth Streaming [142], Adobe HTTP Dynamic Streaming (HDS) [12] and Apple HTTP Live Streaming (HLS) [101], while MPEG-DASH (Dynamic Adaptive Streaming over HTTP) [130] was recently standardized. The main idea behind these techniques lies in segmentation: instead of downloading the full video at once, the video is divided into segments of equal temporal length and segments are then retrieved by the clients. To make it more resilient to network variations, several qualities (different resolutions, different video bitrates) of each segment are encoded. Then, depending on its local characteristics (e.g., available throughput), the client selects the segment quality that would best fit her needs. More details on DAS techniques, and especially on MPEG-DASH, can be found in section 2.2.

2.1.2 Live video streaming

In the case of live video streaming, the source is producing the media while the user asks for it. The main goal is therefore to minimize the latency between the two while providing a smooth quality of experience to the user. Albeit DAS techniques can provide a good user's quality of experience, the minimum latency they can achieve is bounded by the segment granularity: a segment must first be fully encoded before being available for download. To achieve low-latency streaming, several real-time streaming protocols were developed, such as RTMP and RTP.

Real-Time Messaging Protocol (RTMP) is a protocol that was initially a proprietary protocol, but was partially released in [105]. It was developed by Macromedia (now owned by Adobe) for streaming multimedia content over the Internet. RTMP is a TCP-based protocol that establishes persistent connections between Flash players and servers. it splits multimedia streams into fragments of size dynamically negotiated between the two parties.

Real-time Transport Protocol (RTP) [126] is a protocol that provides end-to-end real-time data (e.g., audio, video) delivery services in a push-based fashion. Typically, RTP runs on top of UDP, but it can be used on top of other underlying transport or network protocols. If the underlying protocol is supporting multicast, then RTP also supports multicast distribution. Its primary goal is to support web-conferencing (audio and/or video) and to do so, it had to accommodate to the different media formats available and therefore, a payload type identification is provided in the RTP header to indicate to the receiver the encoded format of the data carried in this RTP packet. Furthermore, as a receiver may receive multimedia contents from several sources, a synchronization source (SSRC) identifier is present in the RTP header to uniquely identify the media source of this packet. The RTP header also presents a sequence number, that is incremented by one for each RTP

packet sent, for the receiver to detect packet loss and packet reordering. RTP is only designed to transport media data, and thus relies on session establishment protocols, such as SIP (Session Initiation protocol), RTSP (Real-Time Streaming Protocol), or SDP (Session Description Protocol).

It is worth noting that while RTP achieves near real-time experience, it presents some scalability issues: for instance, the rate adaptation, unlike DAS techniques, does not rely on predefined and preprocessed video segments encoded at different qualities that the user chooses, but is rather done dynamically and continuously, relying on feedback from the receiver and the sender (e.g., packet loss, measured latency, etc). Indeed, such algorithms, like Google Congestion Control (GCC) [55], Network Aware Dynamic Adaptation (NADA) [145] and Self-Clocked Rate Adaptation for Multimedia (SCReAM) [68] dynamically adjust the video encoding rate at the sender. Consequently, when multiple users wish to see the same multimedia stream, the sender can either use (i) only one encoder or (ii) one encoder per user. When the sender uses only one encoder, the resulting rate adaptation has to take into account the feedback from all the users, resulting in the selection of the worst encoding rate among the rates achievable based on the feedback of each user taken separately, leading to a sub-optimal quality of experience for the users. But when the sender uses one video encoder per user, although each encoder receives feedback from one user and can therefore pick the optimal encoding rate for this user, the CPU load generated by the different encoders prevents this solution to scale above a certain (small) number of clients. To mitigate this issue, recent works [50, 108, 22] use Simulcast to limit the number of encoders at the sender side, while increasing the quality of experience of the users.

Therefore, when some latency is allowed (tens of seconds between the source and the playback at the user), e.g., typically when user's interactions is not required, like sporting events or TV shows, large content distribution still relies on DAS techniques rather than RTP as it offers better scalability performances since it uses CDN infrastructures. Furthermore, as DAS techniques leverage HTTP, NAT traversal is not an issue, while RTP-based streaming techniques need to rely on other protocols to achieve NAT traversal, such as STUN [120], TURN [88] or ICE [119].

2.2 MPEG-DASH

In this section, we give a detailed description of one particular DAS techniques, MPEG-DASH (Dynamic Adaptive Streaming over HTTP) [130], that was recently standardized. While the details concern only one DAS technique, the core ideas behind are shared with the others. We first give a general overview of MPEG-DASH (Section 2.2.1) and then we review the adaptation logics used by the clients

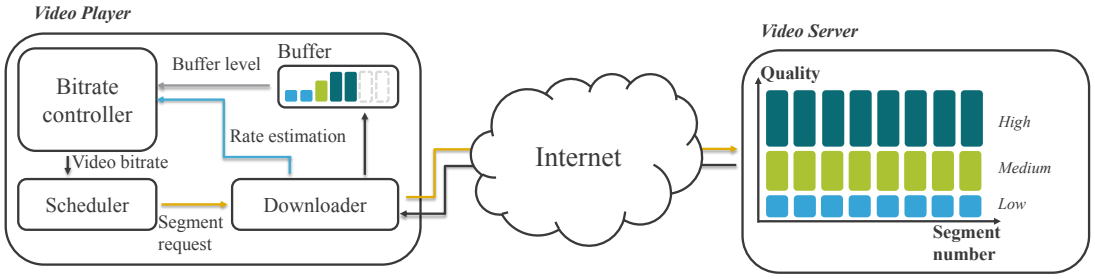


Figure 2.1: Dynamic Adaptive Streaming over HTTP: end-to-end scheme, presenting the video player and the video server.

(Section 2.2.2).

2.2.1 General

Figure 2.1 presents the different parties at play in a DAS scenario: a video player requests the video from a server over the Internet. As described in Section 2.1.1, the multimedia content is divided, either physically (different files) or logically (same files, but the player only requests part of the file by means of HTTP range requests) into segments of equal temporal length[1]. Each segment is encoded at different qualities (e.g., different video bitrates and/or video resolutions) and made available on the server through HTTP standard compliant GET requests. To watch the content, the player downloads the multimedia segments sequentially. Before the downloading of a segment, the client uses local parameters, such as buffer level (the number of segments it has downloaded but not yet played), throughput estimation, screen size, to select the best quality at which to download the segment. The quality is selected following an algorithm, called the adaptation logic, which

[1]Note that all the multimedia content (video and audio) is chunked into segments, therefore either audio and video are interleaved and divided into segments carrying both audio and video data, or they are separated into two different streams of segments, thus requiring the player to download both video and audio segments and to synchronize them to ensure smooth playback.

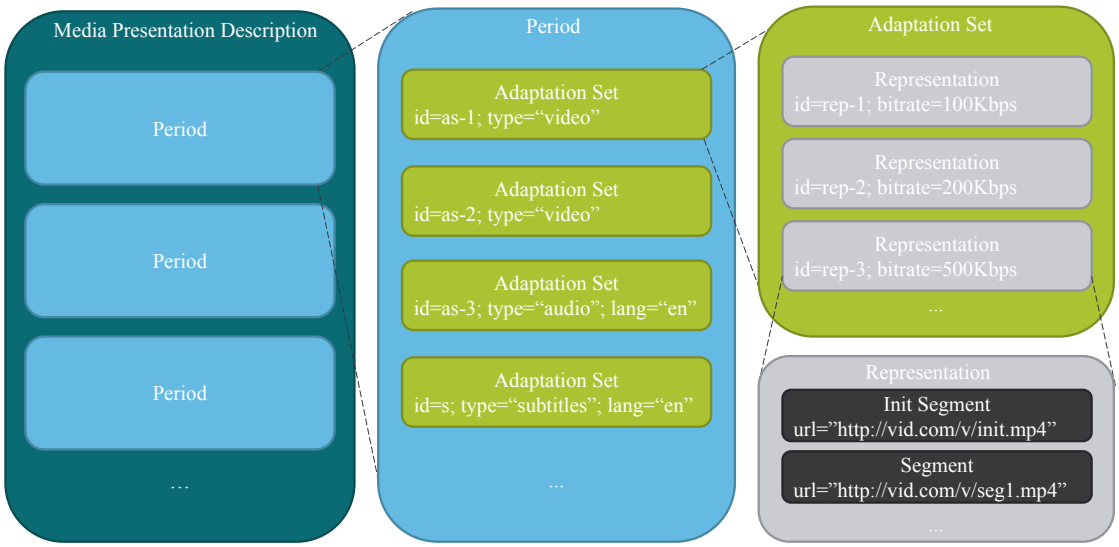


Figure 2.2: Media Presentation Description

allows the client to *adapt* its streaming experience to its capabilities (network characteristics, buffer level, screen size, etc).

To select the best quality, the client needs to be aware of the available qualities for a given content. Such information, like also the URLs of the segments and their relationships to one another, are gathered in one XML file, called a manifest, or, in the case of MPEG-DASH, a Media Presentation Description (MPD). This manifest is stored at the server side, and is the first entity to be downloaded by the client. It describes all the adaptation possibilities for a given video. In the MPEG-DASH case, the MPD presents a hierarchical structure, as depicted in Figure 2.2. It is divided into *Period* components, which describe a part of the content: each Period has a start time and a duration and several Periods can be used to divide a video into chapters, but it can also be used to insert advertisements in the content without having to integrate the advertisement into the video first.

Each Period is divided into *Adaption Sets*, which contain the multimedia streams (video, audio and subtitles/captions). While a single Adaptation Set may contain all the media streams for the content, it is common to have different media streams in different Adaptation Sets, one per media. Besides, a given media stream can be described by several Adaptation Sets: for instance, the audio stream can be represented by different Adaptation Sets, one per available language, or per audio channel format (mono, stereo, 5.1, etc).

Each Adaptation Set is further divided into *Representations*, describing all the

available qualities for the media stream contained in this Adaptation Set. Since the main goal of adaptive streaming is to match the media bitrate to the available bandwidth, it is common to have only one Representation for audio and subtitles streams, as the impact of these streams on the used bandwidth is low, while the video stream offers several Representations, as varying the video bitrate and video resolution can greatly impact the used bandwidth.

Finally, each Representation contains a list of *Media Segments*, which describes the media segments that construct the media stream. The Media Segment contains the URL associated to this segment, and, if the segments are stored in a single file, a byte range. Switching between two Representations could be achieved at any point in the video by requesting a Media Segment from another Representation, however, due to coding dependencies, jumping from one Representation to another at arbitrary positions could prove complicated. To avoid this, MPEG-DASH introduces Stream Access Points (SAP), where a client can effectively switch between Representations. The simplest and most common case is to have a SAP at the beginning of each Segment, that way, the client can switch from one Representation to another at the end of each Segment.

2.2.2 Adaptation logics

The main goal of an adaptation logic is to optimize a chosen metric: for example, the simplest adaptation logic is to always select the lowest quality available, that way, the startup delay and the rebuffering probability are minimized. Most of the adaptation logics focus on optimizing the quality of experience of the client, by relying on simple metrics such as average quality, number of quality switches, rebuffering events, or more elaborated ones, such as linear combinations of simple metrics. These adaptation logics take into account parameters from the client (local parameters) and/or from network entities (e.g., cache, video server, see chapter 4 for more information).

The local parameters depend only on the client and its environment, such as its screen resolution, the available bandwidth and the buffer level. These parameters are then used by the adaptation logic to select the most fitting segment quality. For example, if the screen resolution is FHD, then downloading 4K segments would require downscaling by the video player, therefore yielding the same quality of experience as downloading FHD segments, but would use more bandwidth, as 4K segments are bigger with respect to FHD ones. Therefore the adaptation logic would select a segment quality for screen resolutions up to FHD.

The dynamics of the video buffer level, i.e., the amount of video (given in unit of time, usually seconds) that can be played at the client-side at a given moment t, as found in the literature, is modelled as follows: when the video is playing, the buffer (denoted by $B(t)$) is drained linearly, as shown in Eq. 2.1, and can not be

negative, hence the presence of the Max function.

$$B(t + \delta t) = Max(0, B(t) - \delta t) \tag{2.1}$$

When a segment is fully downloaded, it is appended to the buffer, and thus its duration (denoted by τ) is added to the video buffer, Eq. 2.2.

$$B(t + \delta t) = Max(0, B(t) - \delta t) + \tau \tag{2.2}$$

Therefore, the variation of the buffer $\Delta(B)$ during the download of a segment can be expressed as: $\Delta(B) = \tau - \Delta_{download}$, where $\Delta_{download}$ is the downloading time of the segment. From this, we see that the buffer depletes when the downloading time is higher than the segment duration ($\Delta_{download} > \tau$) and the buffer increases when the downloading time is less than the segment duration ($\Delta_{download} < \tau$). Rebuffering events occur when the buffer depletes fully, $B(t) = 0$, at which point the video player stops and waits for the next segment to be fully downloaded. Finally, the buffer has a maximum capacity, B_{max}. When the buffer level reaches the maximum capacity ($B(t) = B_{max}$), the player stops downloading video segments until there is room in the buffer (drained because the video is playing, or emptied, e.g., the client wants to seek the video). This creates an ON-OFF downloading pattern inherent to DAS systems and well described in the literature [58, 83].

There are numerous works in the DAS literature that focus on application-level and client-side adaptation logics for the video bitrate [15,34,59,66,83,91,129,131, 141], and, more recently, on their systematic comparison in mobile networks [71]. While most of the DAS literature only considers TCP/IP as the underlying protocols, some works consider *in-network functionalities* offered by an ICN paradigm to support DAS [21,48,75,76,80,86,107,111,115]. The main goal of an adaptation logic is to maximize the quality of experience of the user, by (a) minimizing the rebuffering events, (b) maximizing the average quality of the video, (c) minimizinge number of quality switches, (d) minimizing the start-up delay.

Tab. 2.1 presents a summary of the most relevant work in the literature, and explicitly separates the work done in the TCP/IP domain (top) versus in the ICN domain. Following a consolidated taxonomy [133], DAS adaptation logics can be categorized into one of two big families: **rate-based (RB)** or **buffer-based (BB)**, meaning that the adaptation logic mainly[2] relies on either the *estimated throughput* or the *buffer level* to take a decision. This is referred to as "main approach" in Tab. 2.1.

[2]Despite this coarse distinction, in all the surveyed adaptation logics, both metrics (i.e., throughput and buffer level) are often jointly considered in order to obtain a finer adaptation. However, according to the importance that each metric has in the whole decision process, it is still possible to classify the strategy of interest as either *mainly RB* or *mainly BB*.

Table 2.1: State of the art in dynamic adaptive video streaming.

	Reference	Tool	Main Approach	Buffer Level	Avg Bitrate	Quality Switches	Rebuffering	Start-up Latency	Throughput	Delay	Fairness
TCP/IP	FESTIVE [66]	*Experiments*	*RB*	C	O	O			C		O
	PANDA [83]	*Experiments*	*RB*	C	O	O	O		C		O
	BOLA [131]	*Experiments*	*BB*	C	C	O	O				
	AdapTech [15]	*Experiments*	*BB*	M	M				M		
	ELASTIC [34]	*Experiments*	*BB*	C	O	O	O		C		O
	BBA-x [59]	*Experiments*	*BB*	C	O	O	O		C		
	Miller('12) [91]	*Experiments*	*BB*	C	O	O	O		C		
	BIEB [129]	*Heuristic*	*BB*	C	C/O	O	O	O	O		
	Essaili('13) [38]	*Simulation*	*INA*		M						
	QFF [44]	*Optimization*	*INA*			O			C		O
	Thang('14) [133]	*Experiments*	*Investigation*	M	M				M		
	Huang('12) [58]	*Experiments*	*Investigation*		M				M		
	Thang('12) [132]	*Experiments*	*Investigation*		M				M		
	Akhshabi('13) [14]	*Experiments*	*Investigation*		M	M			M		
	Dobrian('11) [37]	*Conviva*	*Measurements*		M		M	M			
	YouSlow [53]	*Chrome*	*Measurements*		M	M	M	M			
	xMPC [141]	*Optimization*	*BB/RB*	C	C	O	O	C	C		
	LCC [113]	*Optimization*	*Offline*						O	C/O	
	Pensieve [90]	*Experiments*	*Neural Network*		M	M	M		C		
ICN	Lederer('14) [76]	*Emulation*	*Investigation*	M					M		
	Lederer('13) [75]	*Emulation*	*Investigation*	M	M	M			M		
	DASC [86]	*Simulation*	*Investigation*		O				C		
	Petrangeli('15) [107]	*Simulation*	*Investigation*	M	M	M					
	DASH-INC [48]	*Model*	*Characterization*		M						
	Bath('15) [21]	*Experiments*	*INA*							M	
	INA [111]	*Simulation*	*INA+BB*						C		O
	DASCache [80]	*Optimization*	*Offline*						O	C	
	Rainer('16) [115]	*Simulation*	*Investigation*		O				C		

Legend: O: objective metric; C: control metric; M: measured metric.
BB: buffer-based; RB: rate-based; INA: in-network adaptation.

Additionally, for each work we report in this table the tools adopted to design the proposed DAS adaptation logic (or to carry out the proposed analysis), along with a set of Key Performance Indicators (KPI), including buffer level, throughput, quality switches, rebuffering events, start-up latency, fairness, etc., that are used either as a *control (**C**)* knob, an *objective (**O**)* of the algorithm, or a *measured (**M**)* metric. The following sections will give an overview of the full landscape, and provide some details about a few adaptation logics that we select as representative of each class. Specifically, we select Probe AND Adapt (PANDA) [83] (mostly RB) and Buffer Occupancy based Lyapunov Algorithm (BOLA) [131] (mostly BB), as they are very popular and often used as reference benchmarks in the literature,

and AdapTech [15], which provides an equal balance between BB and RB classes.

In the following sections, the buffer level, at a given moment t, will be denoted by $B(t)$, $j \in [\![1, N]\!]$, will represent the j^{th} representation, while b_j will represent its associated average video bitrate. Furthermore, $\widetilde{C}_k$ denotes the measured throughput of the k^{th} segment, the size of the k^{th} segment in the j^{th} representation will be denoted by $S_{k,j}$ and the average segment size for the j^{th} representation will be denoted by S_j.

Rate-based strategies (TCP/IP)

Rate-based (RB) algorithms [66, 83], as their names indicate, rely on using the measured throughput of the last segment, $\widetilde{C}_k$, as an *estimate* for the throughput of the next segment $\widehat{C}_{k+1}$. Based on this knowledge, the adaptation logic can infer the highest affordable quality to be requested, as the highest quality with a bitrate less or equal to the estimated throughput,:

$$\underset{j \in [\![1,N]\!]}{\text{ArgMax}}(b_j \leq \widehat{C}_{k+1})$$

However, pure rate-based algorithms suffer from inefficiencies [58], such as *rebufferings* events, since the bitrate of the representations in the manifest is an average, some segments may be significantly bigger than the average size ($S_{k,j} > \tau \times b_j$) and therefore, the actual downloading time, given by: $\Delta_{download} = S_{k,j}/\widetilde{C}_k$ (assuming that the estimation is accurate), may be significantly longer than the estimated one, $\widehat{\Delta_{download}} = S_j/\widehat{C}_k$, and may be higher than the segment duration, thus depleting the buffer and eventually leading to rebuffering events.

The throughput estimation can also be the source of some inefficiencies, such as bandwidth *underutilization* or *overestimation*, which would in both cases lead to bad quality of experience for the user: in the former, the client would get a poor average video quality with respect to the one it could have had and in the latter, the client would get some rebuffering events and quality switches. Both underutilization and overestimation cases derive from the ON-OFF downloading pattern. [58] explores the bandwidth underutilization case and presents the downward-spiral effect: when competing with a TCP flow, a DAS video client suffers from the ON-OFF pattern. When the client is in an OFF period, the competing flow uses all the available bandwidth, and when the client starts to download a new segment, i.e., enters an ON period, it never reaches its fair share of the bandwidth, because it takes some time for the TCP slow-start to ramp up to it. From this, the resulting throughput is low and thus, a pure rate-based adaptation logic tends to select low qualities for the segments. The overestimation case can be observed when two or more DAS players compete for bandwidth [13, 83]: due to the ON-OFF patterns

of the different clients, one client can be the only one in the ON phase at a given point, thus getting the full bandwidth instead of sharing it with the other clients. This leads to an overestimation of the available throughput at the client level, leading to the selection of a quality too high to be sustainable, which will either induce rebuffering events or quality switches. Moreover, the ON-OFF pattern could also lead to unfairness between the different competing DAS clients: some might be forced to request a lower quality with respect to their fair share. Finally, the short-term variations of the bandwidth can induce fluctuation in the throughput estimates and thus leading to quality switches at the client.

Several proposals exist to address the aforementioned issues at client [83, 66], server [14], and/or network [38, 44] points of view.

Buffer-based strategies (TCP/IP)

Buffer-based (BB) algorithms [15, 34, 59, 91, 129, 131] rely on the current buffer occupancy $B(t)$ to select the video quality.

Typically, buffer-based algorithms divide the buffer into multiple ranges and take different actions, according to which range the current buffer level is currently at. Generally, the lowest quality is requested when the buffer is almost empty or below a minimum threshold B_{min} as to avoid rebuffering events; while the highest quality is requested when the buffer is above a maximum threshold B_{up}. To handle the remaining cases (i.e., $B_{min} \leq B(t) \leq B_{up}$), a proper function (e.g., monotonically increasing) is needed to map any possible combination between buffer occupancy and requested video quality inside the feasible region. Segments that accumulate into the buffer can act as a cushion to absorb the effects of small bandwidth variations; however, if the mapping spacing between two consecutive bitrates is too narrow (e.g., number of available qualities too high compared to the buffer range), unwanted quality switches could arise.

Beyond single-stack client-based adaptation

As previously stated, an adaptive video streaming service might take advantage, at a relatively low cost, from built-in features of Information Centric Networking (ICN) [89]. For this reason, despite some initial work assessing the performance of rate-based algorithms for Named Data Networking (NDN) [76], most of the literature on video streaming and ICN has proposed and investigated *in network* adaptation mechanisms [21, 48, 75, 80, 86, 107, 111]. Studies range from the possibility to dynamically select the best performing link (i.e., between 4G and Wifi) when downloading a video segment in a mobile scenario [75] (thus reaching better performance than the classic scenario with a single link), to the usefulness of caching in the presence of multiple clients fetching the same content [86] (thus

resulting in an increment of the retrieved video quality over time). The picture is however far from being complete. For instance, some argue [107] that the presence of in-network caching may favor the use of Scalable Video Coding (SVC) for an ICN-based adaptive streaming service, since the layered approach could increase the efficiency and the flexibility of the adaptation process (i.e., as base layers can be prioritized over enhancement layers in order to guarantee a continuous video playout if the latter ones cannot be retrieved). At the same time, others point out that this could induce some inefficiencies, like quality oscillations [48, 21] due to hit/miss events interfering with the bandwidth estimation process, or even client starvation [111]. Possible solutions propose to increase the decisional and computational power of intermediate nodes (e.g., by altering the media description according to cached bitrates or transcoding the cached qualities [48], or by letting ICN routers perform some form of access control [111]). However, when the in-network adaptation envelope is pushed too far, scalability issues may be encountered (e.g., as in [80], where the orchestrating entity has to solve an Integer Linear Programming (ILP) optimization problem).

2.3 Information-Centric Networking

Information-Centric Networking (ICN) is a novel networking architecture that was introduced by [62]. It proposes a content-centric communication paradigm that leverages location-independent data names along with a content-aware connectionless transport. Namely, ICN is based on location-independent data names, where each content is addressed over the network by a unique data name, rather than by a location (e.g., the IP address of the content-hosting server). The data retrieval is thus one of the key difference between ICN and traditional host-centric networking as, in ICN, the data is retrieved using their names rather than their locations. Moreover, thanks to content-awareness carried by content names, an ICN network is able to route content requests towards nearest content replicas, exploiting in-network caching and adaptive request routing to achieve a more efficient and cost-effective data delivery. The following sections expand on the key aspects of an ICN architecture, such as Named Data (Section 2.3.1), name-based routing and forwarding (Section 2.3.2)

2.3.1 Named Data

The core idea of ICN is to use content-awareness directly at the network-layer to perform network operations (such as forwarding and caching) on topology-independent data names rather than on IP addresses. To achieve this, each Data (video, file, sensor reading, etc) is divided into a sequence of chunks uniquely

identified by a data name and stored in one and more servers. Naming data chunks gives to ICN networks content-awareness and such networks can then take content-aware decisions at the network-layer, without requiring any Deep Packet Inspection (DPI) nor delegation to the application-layer.

The naming convention does not need to be specified and can therefore be application-specific. The only requirement is to have a hierarchical naming structure, like the one already adopted by HTTP URLs, in order to achieve entry aggregation in name-based routing tables. A data name following such naming structure is composed by a variable number of components (not necessarily human-readable), hierarchically ordered.

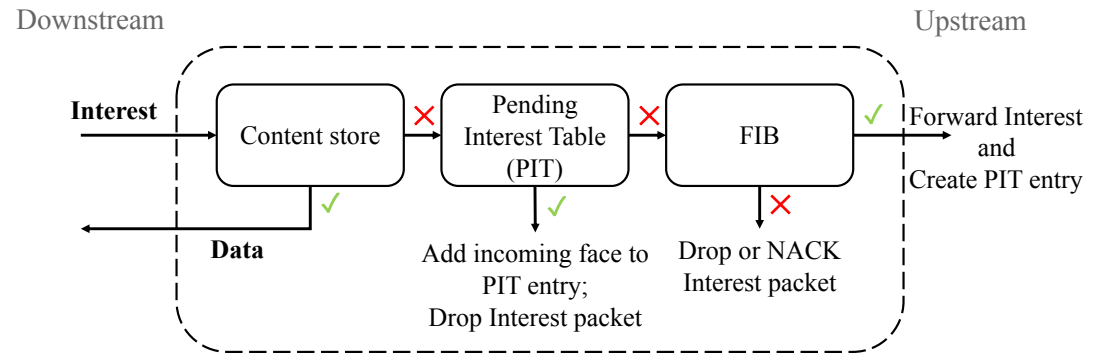
Furthermore, ICN data names can potentially be unbounded (in contrast to, e.g., IP addresses that are limited to either 32 – v4 – or 128 bits – v6) and therefore an efficient name encoding scheme is an important challenge to (i) achieve a fast name parsing and (ii) limit the space needed for carrying the data name in ICN packets.

2.3.2 Name-based routing

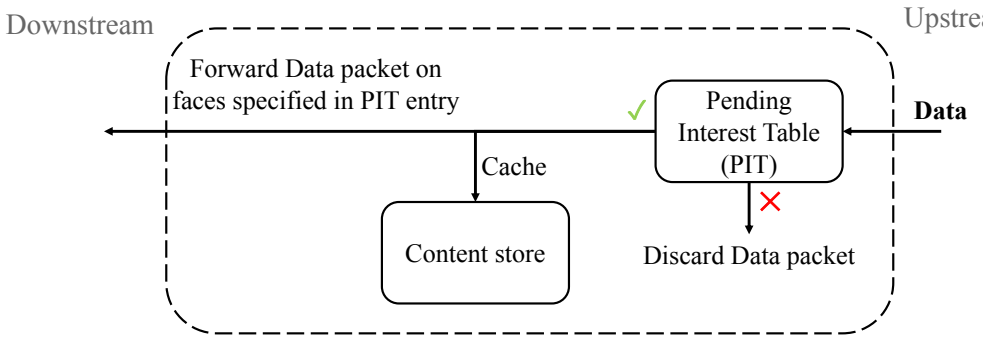
ICN packets are divided in two categories, *Interest* and *Data* packets. Interest packets are packets sent by the users to retrieve data chunks, while Data packets are the data chunks. Each Data packet is identified by the data name of the corresponding data chunk. To request a given data chunk, a user sends out an Interest packet for the data name associated to this data chunk. ICN routers then process the received Interest packets by name in a hop-by-hop fashion towards a permanent copy of the requested content. Symmetric routing is used to route back the requested Data packets back to the user: ICN routers keep track of received Interest packets to return data chunks to the user following the reverse request path. To achieve the aforementioned, each ICN node maintains three structures inside its forwarding engine:

- a **Forwarding Information Base (FIB)**, that is used to forward Interest packets towards potential sources of the requested data. It works like an IP FIB, with the sole difference that it can have multiple outgoing *faces*[3] for a given name prefix rather than a single one.
- a **Content Store (CS)**, that acts as a cache: when a Data packet arrives at the router, it can decide to store this Data packet in its content store. Subsequent Interest packets requesting the same data will then be satisfied

[3]Here, *face* designates where the packets can be sent/received. It can be a hardware *interface*, but packets can also be exchanged with application processes within a computer.



(a) Interest packet forwarding.



(b) Data packet forwarding.

Figure 2.3: ICN forwarding engine.

by the content store directly, rather than forwarding the Interest packet to the data source.

- a **Pending Interest Table (PIT)**, that is used to route the Data packets back to the user. The PIT keeps track of all forwarded Interest packets as PIT entries. A PIT entry corresponds to a name and a list of *face(s)* on which an Interest packet for this name has been received. When a Data packet arrives at the node, it is forwarded on all the *faces* associated to the data name in the corresponding PIT entry. As the routing is done only for Interest packets, the PIT ensures that the Data packets are sent to the requesting user.

Figure 2.3 presents how ICN packets are handled by an ICN node, differentiating the two types of packets in subfigures (a) and (b). The following paragraphs

describe in more depth the routing operations of an ICN node.

When an *Interest* packet is received on a face in an ICN node (Figure 2.3-(a)), the node first checks if the requested data chunk is locally stored. An exact match lookup on the content store is performed for the data name carried by the Interest packet. If there is a match, the corresponding Data packet (stored in the CS) is send back to the user via the face on which the Interest packet was received. If there is no matching data chunk in the content store, the node then performs an exact match lookup on the PIT. If there is a matching PIT entry, the incoming face (the face on which the Interest packet was received) is added to the list of *faces* in the PIT entry and the Interest packet is discarded. This phenomenon is known as PIT aggregation and will be described in more details later. If there is no matching PIT entry, the node performs a longest prefix match lookup on the FIB. If there is a match, a PIT entry for the data name carried by the Interest is created and the Interest packet is forwarded to the *face(s)* given by the FIB match. If there is no match, the Interest packet is discarded as the node does not know how to handle it.

When a *Data* packet is received on a face in an ICN node (Figure 2.3-(b)), the node checks if this data chunk was requested, by performing an exact match lookup on the PIT. If there is no corresponding PIT entry for this data name, this Data packet was not solicited from this node and is therefore discarded. If there is a match in the PIT, the Data packet may be stored by the node in its content store and it is then forwarded on each *face* listed in the matching PIT entry. Finally, the PIT entry is removed, as all the pending requests (recorded by the PIT entry) have been satisfied.

2.3.3 ICN features

ICN offers several features, and the key ones are described in the following paragraphs.

Connectionless, pull-based, client-driven transport

In contrast to traditional sender-based TCP/IP model, the ICN data retrieval model is driven by user requests (Interest packets) sent out for data chunks of the requested content. Therefore, rate and congestion control are left to the end user through a transport protocol offering the following characteristics:

- connectionless: no connection is established between the end points prior to the data transfer.
- supports data retrieval from multiple sources: data chunks of a given content may be retrieved from several locations in the network, such as intermediate

caches or hosting server.

- supports multi-path communication: in the case of multi-homed users, multiple links may be used to retrieve data chunks, in order to perform traffic load-balancing and potentially increase user performance.

Mobility support

As interfaces have no network addresses, physical mobility in ICN does not necessarily translate in a change of address in the data plane. Furthermore, thanks to the connectionless and pull-based nature of the transport layer, the consumer mobility is fully integrated in ICN. The consumer emits Interest packets that are routed through the network towards data chunks and Data packets flow back to the consumer following the trail of breadcrumbs left in the PIT entries. If the consumer moves during the data transfer, the consumer simply re-emits Interest packets for the data chunks that were not yet received and they will be routed through the network towards the data chunks, that may be found in local caches.

Producer mobility is more challenging, however, some protocols were recently proposed to tackle it, such as MAP-Me [18], an anchorless solution to manage mobility of content producers in an ICN environment.

PIT aggregation

When an ICN node receives an Interest packet for a data name that is already present in the PIT, the PIT entry is updated by adding the incoming interface to the list of faces associated to this data name and the Interest is then discarded. This exploits the ICN symmetric routing: as the Data packet follows the reverse request path, the requested Data packet will traverse this node. Therefore, rather than sending another Interest packet upstream (and thus increasing the traffic load for a data chunk that was previously requested), the node adds the Interest incoming face to the existing PIT entry, so that when the Data packet arrives, it is forwarded on all the faces on which an Interest for this Data was received.

It is important to note here that each PIT entry is eventually removed, either by reception of the solicited Data packet, or by the expiration of a timer. This timer ensures that if the Interest packet was lost upstream, the communication is not prevented: the PIT entry will eventually time out and the consumer is responsible to re-issue Interest packets.

In-network caching

ICN nodes present a content store, that give them the ability to temporarily store data chunks in order to serve future requests for these: this is the first step executed

by an ICN node when it receives an Interest packet, it checks for the requested data chunk in its content store. The decision to whether or not cache the data chunk depends on caching algorithms such as *Cache Everything Everywhere* (CE^2) [62], *Leave Copy Down* (LCD) [74], *ProbCache* [114] or *StreamCache* [81]. Moreover, cache replacement is decided upon cache policies that can be enforced, such as FIFO (First In First Out, the first data chunk cached is the first to be replaced), LRU (Least Recently Used), LFU (Least Frequently Used) or TTL (Time To Live) [92].

Furthermore, coupled with the content-awareness provided by the data names, this caching can be leveraged by mechanisms such as WLDR [28] to recover packet losses from the network, without requiring the sender to identify and retransmit lost packets.

Load-balancing

If there are several faces for a FIB entry, an ICN node may use some forwarding strategy to choose on which face to forward the Interest packet. Such forwarding strategies include *broadcast*, where the Interest packet is forwarded on each face of the FIB entry, or *Load-balancing* strategies, where the face on which the Interest packet is forwarded is selected based on some metric, such as the load on a link, its RTT, the channel utilisation, etc...

Security considerations

Current Internet security is provided by protocol extensions such as IPsec and TLS. TLS encrypts a layer 4 connection between two hosts to provide web security: the server authentication relies on certification authorities and a public key infrastructure, while the transmitted data is encrypted thanks to symmetric cypher on the end-points, based on a negotiated key.

In ICN, as there is no connections anymore, the security model is de facto different : ICN security model is based on content encryption at the network layer, based on asymmetric keys, along with a web of trust built upon certification authorities and a public key infrastructure. The authentication of the data (i.e., the verification of the publisher of this data) is done by including the producer signature of the data and its name.

The atomic security service provided by ICN guarantees that the producer has published a piece of data with the name available in the packet, enabling location-independent secured content access. Denial of service attacks based on cache poisoning can be blocked using signature verification techniques, however, the cost is not negligible and some recent work [45] has started to build network layer trust management that does not require in-network verification by using the

concept of Interest-key binding.

Although ICN security framework permits content distribution, several services require to be redesigned, such as, for example, access control: it requires to manage and distribute keys to the group of users with granted access to the controlled data. Furthermore, content revocation requires data version management and policy enforcement to delete obsolete content from the network when need be.

2.3.4 ICN architectures

There are in the literature different ICN architectures, such as Content-Centric Networking (CCN) [62], Named Data Networking (NDN) [143], Publish/Subscribe Networking (PSIRP) [35], Community ICN (CICN) [2] and hybrid ICN (hICN) [97]. As the NDN, CICN and hICN architectures were used in this thesis, the remainder of this section will focus on these architectures.

CICN and NDN share common roots in CCNx 1.0 (an implementation of CCN), as they both originated from the CCN project, and an on-going effort at the IRTF [1] aims for the convergence of NDN and CCN, while CICN focuses on the CCNx 1.0 specification [94], but will evolve by keeping track of the work done in the convergence group at the IRTF. Both CICN and NDN are open-source projects[4], and CICN offers a VPP plugin, to benefit from VPP [84] (Vector Packet Processing), a high-performance packet-processing stack. However, both architectures require a full ICN-enabled network to properly work, and therefore, hICN [97] was recently introduced to offer an incremental deployment solution for ICN into existing IP networks.

hICN proposes a solution to deploy ICN inside IP, rather than an overlay of IP. It makes use of IPv4 or IPv6 RFC compliant packet formats and guarantees transparent interconnection of standard IP routers and hybrid ICN-IP routers while preserving pure ICN behaviour at layer 3 and above. In a nutshell, hICN embeds data names in IP addresses and overloads regular IP packets (including higher layers such as transport) with ICN semantics. The main appeal for these changes is that such packets would be unnoticed by ICN-unaware equipments and thus treated as a regular IP packet, while it will be processed as an ICN packet in hICN-aware equipment.

More information on hICN is available in Chapter 5.

[4]CICN: `https://wiki.fd.io/view/Cicn`, NDN: `https://github.com/named-data`

Chapter 3

Assessing ICN benefits in video delivery

Contents

3.1 Introduction

In this chapter, we assess the benefits of video streaming over an ICN network stack by comparing it against video streaming over a legacy TCP/IP network stack. Since DAS techniques were initially designed for CDN/OTT content delivery, their interaction with the network has been only superficially studied so far. In the 5G mobile and heterogeneous network access, it seems of utmost importance to consider DAS application-network interaction, and to move caching and computing capabilities to the network edge in order to enable efficient mobile video delivery [17]. Given this context, ICN appears as a natural network substrate for DAS [21, 48, 75, 76, 80, 86, 107, 111, 115]. The features offered by ICN, such as network-level caching, multi-path forwarding capabilities and seamless mobility support are all very appealing for DAS systems, and we argue that using such features would help improving the user quality of experience. However, the potential for ICN application in adaptive streaming services as an alternative to relieve from some of the recognized inefficiencies of standard TCP/IP transport has been only partially explored (refer to [140] for an overview of ICN aspects related to video delivery). Recently, valuable work started to appear [21,48,75,76,80,86,107,111,115], which gives hints on the potential benefits coming by exploiting capabilities of an ICN content-aware architectures to assist DAS rate adaptation inside the network, rather than only at the client side. At the same time, the literature currently lacks a systematic approach for testing the interplay of ICN and DAS. Similarly, a quantification of the benefits ICN could bring over the current TCP/IP solutions in realistic environments is far from being complete.

As a first step to evaluate such benefits, we selected three state-of-the-art DASH adaptation logics (PANDA, AdapTech and BOLA) that are representative of the whole design space and we compared the performance of video streaming over an ICN network stack with video streaming over a TCP/IP network stack. This comparison was based on an experimental campaign using tools that we developed and made available as open-source software. As there is no default transport protocol in ICN, we resorted to previous work and used ICP, the Interest Control Protocol, introduced in [25]. We considered a wide range of scenarios, varying the number of clients (from single clients to multiples ones), the network stack (especially when several clients are involved, we considered an homogeneous case, where all clients have the same network stack type, ICN or TCP/IP, and a heterogeneous case), the arrival rate of clients (synchronous versus asynchronous), the channels used (emulated LTE and Wi-Fi, DASH profiles, real 3G/4G traces) and the different levels of integration with an ICN network (vanilla NDN, wireless loss detection and recovery at the access point, load-balancing). To the best of our knowledge, a systematic comparison is very rare, already in the TCP/IP DASH world, where [71] represents the most notable exception. As such, the broader picture of a system-

atic DAS comparison under both TCP/IP and ICP/NDN stacks that this work addresses is still totally unexplored.

This chapter is divided as follow: Section 3.2 presents the three adaptation logics that we selected and introduces the architecture, the emulation platform and scenarios, Sections 3.3 and 3.4 report experimental results and finally Section 3.5 summarizes the main lessons.

The results presented in this chapter were published in [122] and a working demo was shown at [121] and at MWC'17[1].

3.2 Methodology

3.2.1 Adaptation logics used

As described in Section 2.2.2, the DAS adaptation logics can be divided into two families: the *rate-based* strategies and the *buffer-based* strategies. For each family, we select one adaptation logic to compare its performance over a TCP/IP stack versus an ICN one. Namely, we selected PANDA [83] (rate-based), BOLA [131] (buffer-based) and AdapTech [15] (rate-based and buffer-based), since we identified these strategies as representative of their families. They are described in more details in the next paragraphs.

PANDA

The strategy proposed in [83], namely Probe and Adapt (PANDA), is a rate-based adaptation logic that takes inspiration from TCP congestion control, implementing the same principles at the application layer (i.e., operating at a video-segment rather than at RTT timescale). The main observation is that throughput estimates are accurate (i.e., they reflect the fair-share bandwidth) only when links are oversubscribed and with no OFF periods (i.e., the client is always downloading). In the remaining cases, overestimations occur. The idea is then to constantly *probe* the available bandwidth by varying the requested bitrate. Since bitrates associated to available video qualities are discrete, intervals between consecutive requests for video segments are fine-tuned in order to obtain a *continuous* average data rate sent over the network: the average data rate is used to probe the bandwidth until congestion (i.e., when the network conditions cannot sustain the requested bitrate, and a back off should occur), and determine inter-request time.

PANDA comprises four main steps:

1. the target average data rate is computed using an additive increase multiplicative decrease (AIMD)-like bandwidth estimation;

[1] `https://www.gsma.com/gsmaeurope/event/gsma-mobile-world-congress-2017/`

2. the target rate is smoothed using an exponential weighted moving average (EWMA);

3. the smoothed target rate is quantized in order to compute the quality to be requested, this is achieved using a dead-zone quantizer with an up-shift (Δ_{up}) and a down-shift (Δ_{down}), which act as safety margins to mitigate frequent quality switching between two adjacent video bitrates;

4. the next segment request is scheduled to comply with a target inter-request time: if the actual download time is smaller than this target, the client will wait a time equal to their difference before requesting the next segment.

Compared to other rate-based players, PANDA is shown to have the best stability-responsiveness trade-off, which is why we selected it as representative of the rate-based adaptation logics.

BOLA

[131] proposes BOLA, a buffer-based adaptation logic that addresses the bitrate adaptation using an utility maximization problem. The goal of this algorithm is to maximize a joint utility defined by:

$$\bar{v}_N + \gamma \bar{s}_N \tag{3.1}$$

$\bar{v}_N$ represents the time-average playback quality computed over the N segments of the video (a logarithmic function is used to compute each single term), and $\bar{s}_N$ represents the average playback smoothness (i.e., the fraction of time spent not rebuffering). γ is a weighting parameter which allows to prioritize between the two metrics. Through problem relaxation, the authors conceive an online version of BOLA, where, at each time-slot, adaptation is made by monitoring the current buffer level and by solving a deterministic optimization problem, whose constraints are those of keeping the buffer as much stable as possible, and maximizing the aforementioned utility function. Different variants of the main strategy are also proposed in order to either minimize the number of quality shifts (i.e., since a bitrate capping is introduced by monitoring the available bandwidth, utility can be sacrificed), or maximize the utility (with more quality variations). BOLA is the default strategy implemented in the DASH.js player [4], which makes it a good representative of the buffer-based adaptation logics.

AdapTech

[15] introduces AdapTech, an adaptation logic that offers characteristics from both buffer-based and rate-based strategies. The main goal of AdapTech is to stabilize

the buffer level around a target value, B_{steady}, while keeping the video quality as smooth as possible, by avoiding to react to short term bandwidth spikes, which would trigger unnecessary quality switches. The algorithm defines two thresholds, θ_1 and θ_2, which divide the buffer in three regions: the *panic* zone ($0 \leq B(t) \leq \theta_1$), the *buffering-state* zone ($\theta_1 \leq B(t) \leq \theta_2$) and the *cushion-state* zone ($\theta_2 \leq B(t) \leq B_{max}$). The algorithm keeps track of two different bandwidth estimates: the throughput of the last segment, A to which we will refer as the instantaneous throughput, and its smoothed version, $\hat{A}$, computed via an EWMA ($\hat{A(n)} = \alpha \times \hat{A(n-1)} + (1-\alpha) \times A(n)$), to which we will refer as the average throughput.

The selection of the quality of the next segment is done following one of the three modes described below. The used mode depends on which zone the buffer level is and the current quality (q_k):

- **Panic mode:** when the buffer level is in the panic zone, the lowest video quality is selected;
- **Buffering-state mode:** when the buffer level is in the buffering-state zone, the video quality is selected using the instantaneous throughput: if the instantaneous throughput is higher than the video bitrate of the next quality ($A > b_{k+1}$), then the next quality, q_{k+1} is selected. If the instantaneous throughput is lower than the current video bitrate ($A \leq b_k$), then the quality is decreased and q_{k-1} is selected (assuming that the current quality is not the lowest one);
- **Cushion-state mode:** when the buffer level is in the cushion-state zone, the video quality is never decreased, and can be increased if over the last T seconds, the average throughput is higher than the video bitrate of the next quality ($\hat{A} > b_{k+1}$) and the instantaneous throughput is also higher than the video bitrate of the next quality.

The panic mode aims at quickly building up the buffer to avoid rebuffering events. The buffering-state mode aims also at increasing the buffer, but as the panic mode partially filled up the buffer, it is not critical to have the video segments as quickly as possible, therefore the quality selection can be done following the instantaneous throughput variations. Therefore, the quality selection rapidly adapts to network conditions by quickly switching to sustainable qualities. Finally, in the cushion-state mode, on the one hand, the video quality can not be decreased to avoid negative short-term fluctuations of the bandwidth, as the already built-up buffer can absorb the downloading time variations induced by a negative spike in

the bandwidth. On the other hand, the T parameter is introduced to prevent positive short-term fluctuation of the bandwidth to trigger unwanted quality switch.

As AdapTech offers characteristics from both rate-based and buffer-based strategies, we select it to consider the impact of a hybrid adaptation logic.

3.2.2 Architecture

Figure 3.1 presents the reference architecture we consider to compare TCP/IP pull-push and NDN pull-pull approaches in a DAS scenario[2]. To conduct our comparison, we used the MPEG-DASH standard [130] as the streaming system for our emulations, while the videos were encoded using the H.264/MPEG-4 video coding standard. The framework we used to orchestrate the experimental campaign is called VICN and was open-sourced as part of a Linux foundation project [2] and a bit more detailed in Appendix A, while the instructions to reproduce our results can be found in Appendix C.

Client and server

The client *controller* drives the video segment request process, which consists of a series of requests for video segments, encapsulated in HTTP request/response pairs (the orange/black arrows). As previously indicated, we select state of the art representatives for all possible adaptation logic families, namely rate-based (PANDA), buffer-based (BOLA), or hybrid (AdapTech). For each adaptation strategy, we perform a thorough calibration in Section 3.3. Furthermore, clients have the option to select one of the two network stacks: the TCP/IP and the NDN one. When the clients use a TCP/IP network stack, the video is served by an Apache HTTP server, while an NDN repository [8] is used when the clients use an NDN network stack. For our comparison, we consider both *single* and *multiple* clients scenarios, in both *homogeneous* and *heterogeneous* settings, with either *synchronous* or *asynchronous* start times.

Congestion control

While investigation of the congestion control flavor is not among the main goal of our comparison, it is worth pointing out some differences among the two stacks under investigation. In the TCP/IP case, congestion control of video-segment transmissions is exerted by the server according to the well known Cubic TCP flavor. In the NDN case, control over video-segment transmissions is exerted by the client, by

[2]The client always selects which segment to *pull* from the server, but the segment retrieval is achieved either via a *push* from the server (TCP/IP), or via a*pull* from the client (NDN : each data chunk of the segment is requested via Interest packets).

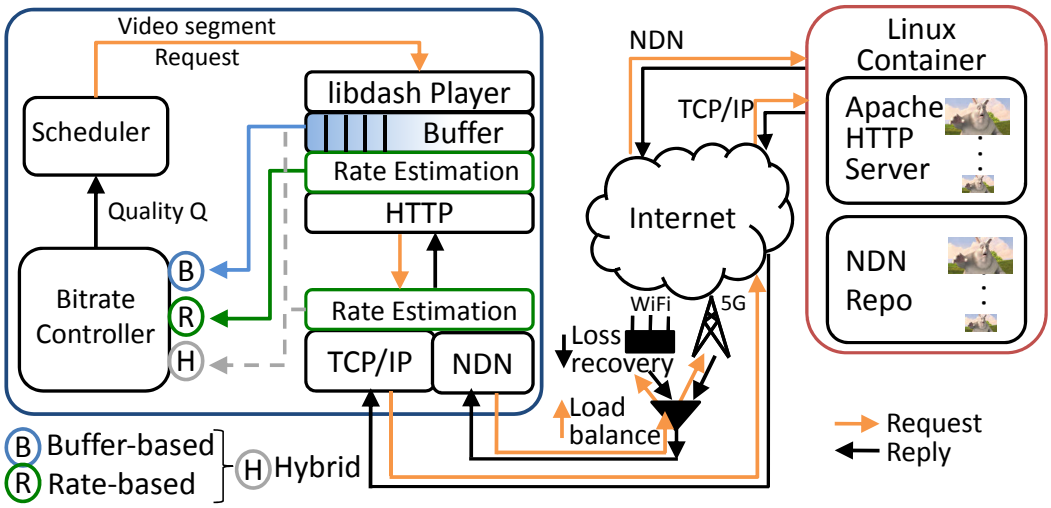


Figure 3.1: Synoptic of the DAS video streaming architecture used for the ICP/NDN vs TCP/IP comparison.

means of *Interest* control. Since in the NDN world there is neither a TCP equivalent, nor a protocol considered as the de facto "default" one, we resort to previous work [25]. ICP, Interest Control Protocol, uses an *AIMD* mechanism to control the window growth, which is regulated according to *delay* measurements – hence we expect ICP to be no more aggressive than MIMD and loss-based TCP Cubic. It is worth noting also that, unlike TCP, ICP does not support FastRetransmit, therefore ICP recovers losses via timeouts; and ICP does not support slow-start either, thus starting with AIMD congestion avoidance. Given this differences, we also need to assess to what extent these differences impact the performance gap between TCP/IP versus ICP/NDN, which we address in Sections 3.3.2 and 3.4.1.

Bandwidth estimation

While buffer level estimation is agnostic to the network stack used, and would be therefore the same for both stacks, the throughput estimation differs between the two network stacks we use here. TCP/IP clients only have estimates of the download rate at the video segment level, which is the throughput of the TCP connection carrying the video segment over an HTTP reply. As the bandwidth is controlled at the server (sender) side, the client cannot have finer-grained estimations out of the box. Using sub-segment level estimations of the throughput would require support from the TCP/IP stack at the server side, along with an out-of-band protocol for signalling. This mismatch does not appear when the client uses an NDN network stack, as all the control is exerted on the client side, and thus the local client stack can leverage NDN-chunk level information to com-

pute finer-grained bandwidth estimates, typically sub-segment level estimates. We investigate further bandwidth estimation granularity in Section 3.4.2.

In-network loss recovery

As some previous work show, the NDN model offers new opportunities [115] for the deployment of an efficient video streaming service, especially in mobile environments [75]: since NDN leverages the use of caches inside nodes and offers a security model where contents themselves are secured, instead of the client-server connection, Data packets can be, in principle, retrieved from multiple locations (i.e., multipath support) and from any node in the network (thus implicitly building a multicast-transmission tree). To produce a fair comparison against TCP/IP, we let aside large and long-lasting NDN caches, but an additional advantage of NDN over TCP/IP is the fact that even small buffer memories can be used as temporary caches. Using these would enable *Wireless Loss Detection and Recovery* (WLDR) [28] of NDN Data packets at the first hop. WLDR induces a faster and cheaper, in terms of network resources, loss recovery mechanism than retransmissions (at the client side for NDN or at the server side for TCP/IP stack).

WLDR is a mechanism that works between two directly interconnected nodes and implemented at face level. By adding a sequence number in the header of the NDN Interest and NDN Data packets and keeping track of the number of packets sent on the face, WLDR is able to quickly detect losses. On the sender side, a per-face counter is used to keep track of packets (Interest and Data) sent through a given face. The counter gives the sequence number to be added in the header of the packet and is then increased when the packet is sent. On the receiver side, the node keeps an expected sequence number value, when a packet arrives, the sequence number on that packet is compared to the expected value. If they are the same, the expected value is incremented, if they differ, the receiver infers that there was a loss on the link and can directly notify the sender about the lost packets: the sequence number of the missing packets is known from the difference between the received sequence number and the received one. The impact of WLDR on DAS is investigated in Section 3.4.1.

Multi-cast/Multi-path support

In NDN, multi-cast and multi-path functions remain transparent to the application, whose controller still operates on the aggregate rate. Unlike in IP, NDN naturally supports multicast via PIT aggregation (and caching). Additionally, since TCP/IP only supports a connection oriented mode, multi-path support must be enforced at application level; at the same time, we are not aware of any DAS video controller explicitly supporting multiple paths. Similarly, whereas Multi-path TCP

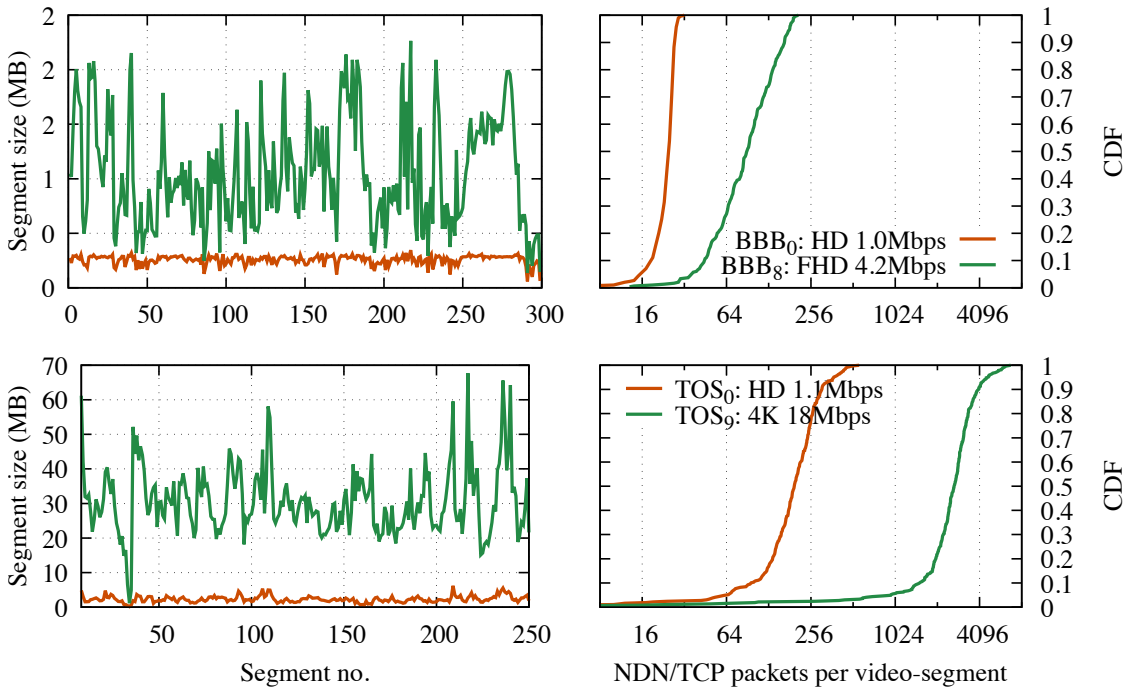


Figure 3.2: Highest and lowest quality representations for the BBB (top) and TOS (bottom) video: temporal evolution of video segment size (left) and cumulative distribution function of the number of TCP/NDN messages per video-segment (right).

(MTPTC) deployment is growing, a number of studies [64, 32] points to MPTCP as actually *harming* user experience. Conversely, the ICP/NDN model allows a very simple mean to support for multiple path, which can be implemented at NDN-chunk level as a simple *Load-Balancing* (LB) function among all available faces, and, thus, applied directly by the client. Notice that the load balancing is applied to *Interest* packets, but due to NDN symmetric routing where *Data* follows back the trail of breadcrumbs left in the PIT by *Interest* packets, the load balancing consequently applies also to the corresponding video *Data* packets. Additionally, we consider two granularities for the LB function: namely, at *transport-segment* (i.e., at packet levelm easy in NDN, but hard in TCP) vs *video-segment* level (possible in both NDN and TCP). We report multi-cast and multi-path results respectively in Secs.3.3.3 and 3.4.4.

3.2.3 Scenario description

Video sources

In this chapter, we use two different videos: Big Buck Bunny (BBB) and Tears of Steel (TOS), both with a segment duration of 2 seconds. Both videos can be found in the dataset of [77]. As our study focuses only on high-quality streaming, we selected only a subset of the available representations of these videos. As a result, we use BBB encoded in 9 video representations, 3 of which are encoded at 1280x720p HD resolution (1, 1.2, and 1.5 Mbps) and the rest at 1920x1080p FHD resolution (2.1, 2.5, 3.1, 3.5, 3.8, and 4.2 Mbps). Similarly, for TOS we only consider bitrates higher than 1Mbps, selecting 7 representations from the dataset, namely at 1280x720p (1.1, 1.5, and 2.4Mbps) and at 1920x1080p (3, 4, 6, and 10Mbps). Aiming at supporting even higher qualities, for the TOS video we encoded three new representations/qualities, i.e., 1920x1080p (FHD, 12Mbps), 2560x1440p (QHD, 15Mbps), and 3840x2160p (UHD or 4K, 18Mbps), that we appended to the existing ones, thus obtaining a total of 10 representations. For the sake of illustration, Figure 3.2 presents, for both the lowest and the highest representations of each video, the size of the segments. It also depicts the distribution of the number of TCP *segments* (or NDN *Data* packets) per video segment in order to give an idea of the granularity, in bytes, of the controller decision. It is worth highlighting that a video segment can be composed of hundreds to thousands packets for the highest qualities.

Network scenarios

In this section, we present the increasingly complex scenarios that we defined to compare DAS over TCP/IP versus NDN. In these scenarios, we vary video (BBB, TOS), bandwidth (DASH profile, heterogeneous access), NDN network features enabled (vanilla, WLDR, LB), and the controller logic and settings. The DASH profiles, defined in [3], are emulated using the Token Bucket Filter (TBF) of the Linux traffic control suite (tc) [3], while the characteristics of the access network are either emulated using the ns3 channel models in MiniNet (Wi-Fi and LTE), or enforced using real 3G/4G traces [117, 136]. To perform a fair comparison of NDN against TCP/IP, we left out scenarios involving routers equipped with caches.

Besides, while our framework is able to support the deployment of complex network scenarios, we focus in this chapter on simpler topologies: the Internet cloud depicted in Fig. 3.1 is modelled as a simple dumbbell topology connecting the access point, either the Wi-Fi access point or the 3G/4G base station, to the origin video server; thus assessing ICN capabilities at the network access. This

[3] `https://linux.die.net/man/8/tc`

simplistic setting alreay allows to assess implementing functions at network level, as opposite to an over-the-top approach as CDN would do in a TCP/IP case (given that Wi-Fi AP and 3G/4G base-station are managed by the ISPs, and currently out of the reach of CDN providers). More specifically, the following cases were considered:

(A) Calibration: A single client downloads the BBB video through a single network channel with a bell-shaped bandwidth profile, taken from the DASH profiles [3], used to calibrate the three selected adaptation logics (BOLA, PANDA and AdapTech). The aim is to contrast their performance under (i) TCP/IP vs (ii) vanilla NDN stacks (i.e., neither LB, nor WLDR). Results are presented in Section 3.3.2.

(B) Multi-clients: An *homogeneous* (either all TCP/IP or all NDN) or an *heterogeneous* (half TCP/IP, half NDN) population of clients download the TOS video, where clients start time are either *synchronized* (live streaming) or *desynchronized* (VoD case). Results are presented in Section 3.3.3.

(C) Transport: A single client downloads the TOS video through a single emulated Wi-Fi channel. We contrast (i) TCP/IP against (ii) vanilla NDN or (iii) NDN with WLDR, furthermore varying the granularity of the bandwidth estimation technique at either (iv) video-segment or (v) NDN-chunk levels. Results are presented in Sections 3.4.1 and 3.4.2.

(D) Network Access: A single client downloads the TOS video, contrasting different access types and emulation techniques: model-based Wi-Fi/LTE vs trace-driven 3G/4G. Results are presented in Section 3.4.3.

(E) Load balance: A single client downloads the TOS video in a multi-homed Wi-Fi and LTE environment. In this scenario we add a LB beyond the WLDR capabilities, and contrast LB operations at (i) fine-grained, i.e., per *Interest* vs (ii) coarse-grained, i.e., per video-segment level. Results are presented in Section 3.4.4.

3.3 Calibration Results

In this section, we carry out a preliminary calibration of the selected DAS algorithms over TCP/IP and NDN stacks. Rather than exhaustively present the full quantitative details of the sensitivity of each parameter of each selected adaptation logic, we show insights about the qualitative behaviour of the selected strategies, while contrasting their performance under a TCP/IP and a bare-bone NDN stack. Furthermore, we perform a careful tuning of the best algorithmic settings for each

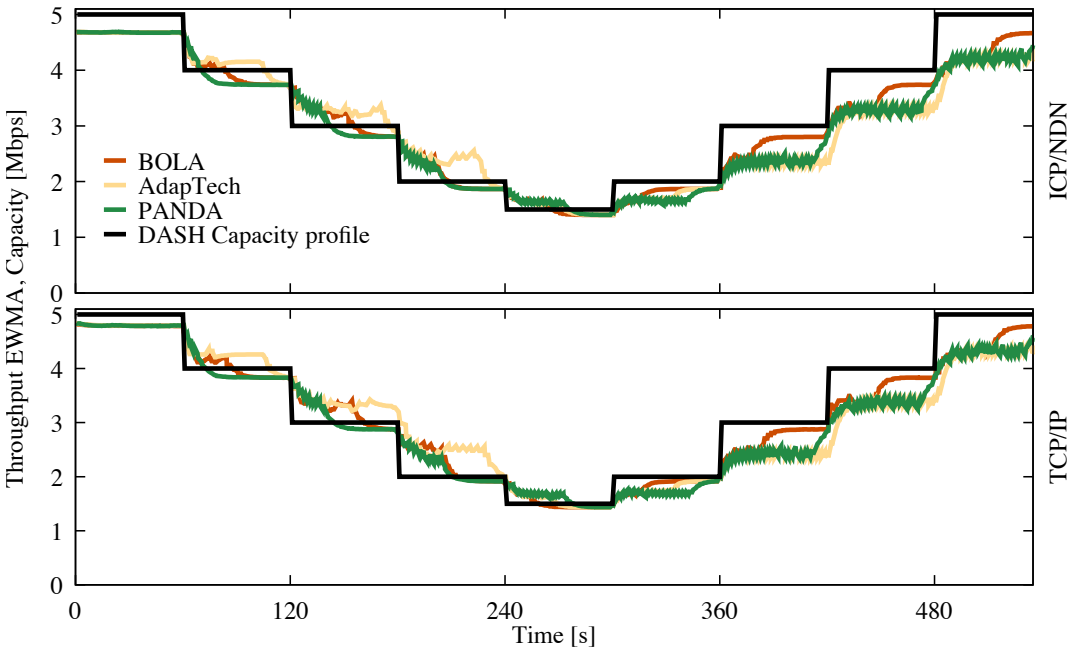


Figure 3.3: Time evolution of the estimated throughput for the three selected strategies (EWMA smoothed version) and DASH capacity profile.

selected adaptation logic, settings that will be used for the remainder of the experimental campaign. The instructions to reproduce can be found in Appendix C.

We start our analysis by showing, at a glance, the behaviour of the three selected DAS adaptation logics in their best configuration (Section 3.3.1), before detailing the finding of these best configurations (Section 3.3.2).

3.3.1 Adaptation logic behaviour

We instrument the simple client-server scenario (A) with a client connected through a wired link to a server, from which she requests video segments from BBB. The bandwidth and delay of the wired link are varied following a standard DASH profile, namely the 2a profile in [3], with 60 seconds variations. Introducing bandwidth and delay variations allows us to illustrate the different operational points reached by PANDA, BOLA and AdapTech, and also to assess the interplay between the DASH client adaptation logic at network, i.e., IP vs NDN, and transport layers, TCP vs ICP, under both stacks.

Figure 3.4 presents the time evolution of the *requested quality* (red line) and the *buffer level* (green line) for the three selected strategies, and for the two stacks. Each plot is annotated with a tuple: $(\bar{q},\#QS,\bar{f}_{QS},|\Delta(QS)|,R,RTime)$, represent-

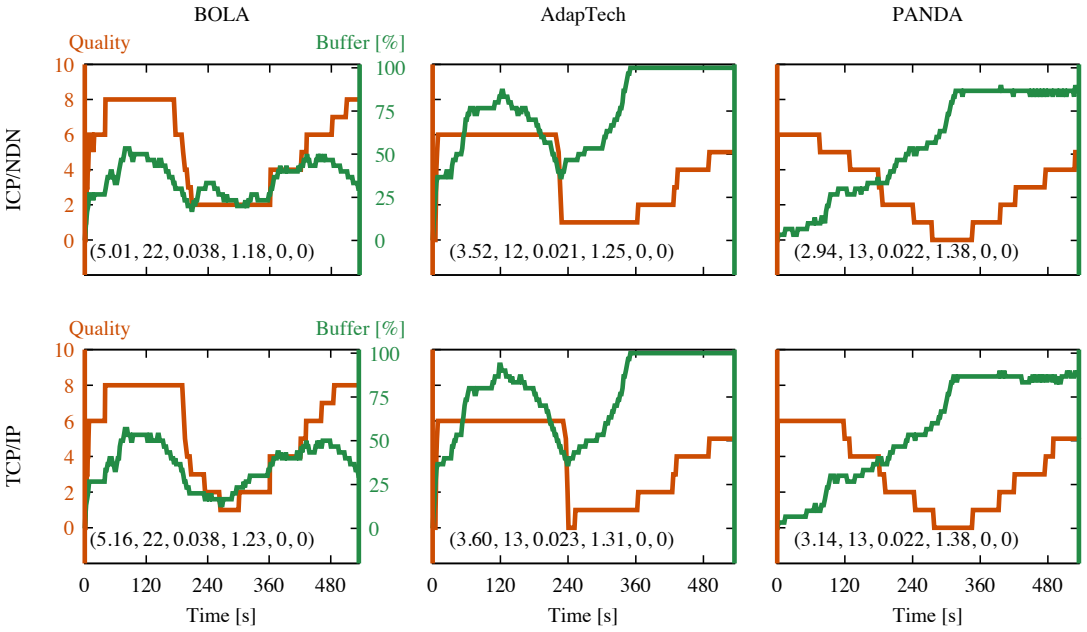


Figure 3.4: Scenario (A) with BBB video: time evolution of requested quality (i.e., the correspondent video representation requested by the client, where "0" is the lowest and "8" is the highest one) and buffer level for the best settings of the three selected strategies (BOLA, AdapTech, and PANDA, on each different columns), running on top of both ICP/NDN (top) vs TCP/IP (bottom) stacks. The picture is annotated with a tuple ($\bar{q}$,$\#QS$,$\bar{f}_{QS}$,$|\Delta(QS)|$,R,$RTime$) representing the main KPIs, namely: average quality $\bar{q}$, number $\#QS$, frequency $\bar{f}_{QS}$, and amplitude $|\Delta(QS)|$ of quality switches; number R and duration $RTime$ of rebufferings. Out of the box, in simple DASH settings, ICP/NDN performance matches that of TCP/IP for all DAS strategies.

ing the main key performance indicators: the average quality $\bar{q}$, the number of quality switches $\#QS$, their frequency $\bar{f}_{QS}$ and their amplitude $|\Delta(QS)|$, along with the number of rebuffering events R and the time spent in rebuffering $RTime$. These key performance indicators are detailed in the next section. Furthermore, the corresponding DASH capacity profile and the Exponential Weighted Moving Average (EWMA) of the estimated throughput at the client is reported in Figure 3.3. Two main messages arise from these results.

First, for this basic scenario (A) with no packet losses, no difference appears between the two stacks: each algorithm, being either prevalently buffer-based (e.g., BOLA and AdapTech) or rate-based (e.g., PANDA), behaves exactly the same, regardless of the network stack. This is especially reassuring since ICP and TCP

are two similar but not identical congestion control protocols, that are furthermore exerted in opposite pull vs push modes. For instance, while both ICP and TCP use AIMD to govern the window growth, TCP reacts on losses, whereas ICP reacts primarily on delay variations; additionally, TCP recovers losses mainly via FastRecovery (if the cwnd is large enough), whereas ICP recovers losses via Timeouts; finally, TCP implements slow-start, whereas ICP does not (in the current implementation). Still, it can be seen that transport-layer differences do not result in noticeable changes in the DAS algorithm behaviour.

Next, consider the specific behaviour of each algorithm. One can clearly see a trend going from left (BOLA) to middle (AdapTech) and right (PANDA) in both the quality and buffer level. Specifically, BOLA more aggressively follows the bandwidth profile: this results in a higher average quality than the one in AdapTech and PANDA. As a consequence, the buffer level is lower in BOLA with respect to AdapTech and PANDA, since the former fully exploits the available bandwidth to download at higher qualities, using the built buffer as a safety net (notice that when the available bandwidth drops, the requested quality remains high for a while, at the cost of buffer level), whereas the latter one use the available bandwidth to increase the buffer and be more resilient against varying conditions.

3.3.2 Sensitivity analysis

The results presented in the previous section are gathered with DAS settings found with an empiric sensitivity analysis, that we report in this section. More precisely, we start from suggested configurations, either taken from reference papers, or open source codebase when they are not available in the reference papers, and we vary the most prominent parameters of each selected algorithm.

Specifically, we vary BOLA's *stable buffer threshold*, which states the difference between startup and steady state [131], in the range [6,24] seconds (the suggested default value in the DASH.js implementation [4] is 12 seconds). Concerning the AdapTech strategy , we vary the two thresholds, θ_1 and θ_2 (expressed as percentage of the buffer size [15]), which affect the behaviour of AdapTech, exploring $\theta_1 \in \{10\%, 20\%, 30\%\}$ and $\theta_2 \in \{40\%, 60\%, 80\%\}$; but we keep the T parameter to its default value of 10 seconds [15]. Finally, for PANDA, we tune the B_{min} parameter, which we adapt to the length of the buffer in our experiments (i.e., 60 seconds), and vary as $B_{min} \in \{34, 44, 54\}$ seconds. Additionally, we use two separate configurations: a more aggressive one, which follows the settings for the thresholds Δ_{up} and Δ_{down} suggested in [83], while we obtain a more conservative behaviour with the settings described in [82].

In order to comprehensively compare the three selected DAS algorithms, we consider six different metrics, among the many available, to estimate the user quality of experience [128]:

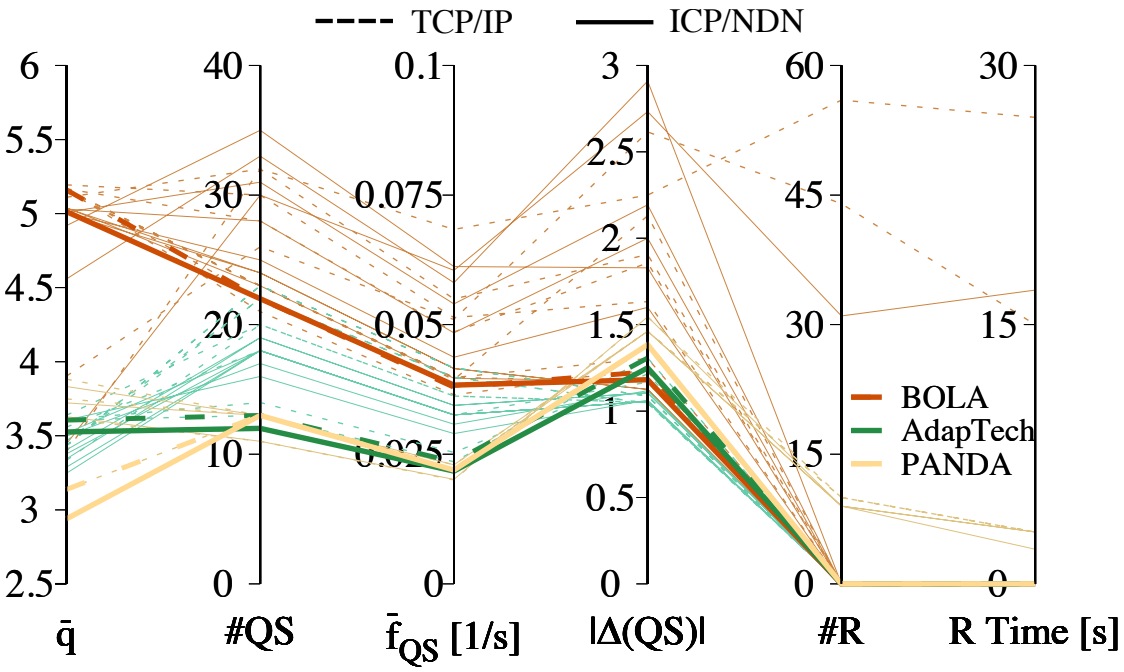


Figure 3.5: Calibration of selected adaptation strategies in a simple client-server scenario with bandwidth and delay variations.

- **Average Video Quality** $\bar{q}$: the average quality of the downloaded video segments over all chunks for the selected algorithm.
  It is computed as $\bar{q} = \frac{1}{K}\sum_{k=1}^{K} q_k$.

- **Number of Quality Switches** #QS: the total number of times the adaptation logic changes the requested quality.

- **Average Quality Switch Frequency** $\bar{f}_{QS}$: the inverse of the average continuous quality playback (i.e., lapse of time at which successive segments are requested at the same quality).
  It is computed as $\bar{f}_{QS} = 1/[\frac{1}{S-1} \times \sum_{z=1}^{S} t(QS_z) - t(QS_{z-1})]$, where $t(QS_z)$ is the time instant of the z-th quality switch, and $t(QS_0) = 0s$.

- **Average Quality Variations** $|\Delta(QS)|$: the average magnitude of a quality switch between consecutive segments.
  It is computed as $|\Delta(QS)| = \frac{1}{K-1}\sum_{k=1}^{K}|q_{k+1} - q_k|$.

- **Number of Rebuffering Events** $\#R$: the number of times the video playout is interrupted due to buffer depletion (i.e., rebuffering events).

- **Total Time Rebuffering** $RTime$: total amount of time spent rebuffering.

In order to efficiently represent the above 6 key performance indicators for the combination of the 46 explored settings, we depict results as a parallel coordinate plot in Figure 3.5, which allows to grasp the correlation between KPIs for specific settings. Each line in the plot presents the performance achieved by a DAS algorithm with specific settings: in the parallel coordinate representation, lines are a pure representation artefact that joins values taken by a DAS algorithm using specific settings over the different KPIs. To distinguish between the different adaptation logics used, each line is assigned a colour, depending on the strategy it represents. Namely, a brown line depicts results for given settings of BOLA, while green depicts AdapTech and gold, PANDA. Furthermore, a specific line type is assigned for each network stack: a dashed line depicts a DAS experiment over a TCP/IP stack, while a ICP/NDN stack is presented using a solid line. Finally, the thicker lines indicate the best selected settings. Note that there are, for each strategy, two thick line, indicating the best combination for this strategy over both TCP/IP and ICP/NDN stacks. To select the best settings for each strategy, we first select settings that avoid rebuffering events, because they represent the major factor in user disengagement [37]. Among the remaining settings, we select the one that maximizes the average video quality while minimizing the number of quality switches: when the average video quality difference between two settings is less than 5%, we select the setting that reduces the number of quality switches.

The results shown in Figure 3.5 reveal that there is not any significant difference between the best cases of TCP/IP and ICP/NDN stacks for each strategy, at least under the scenario used in this calibration phase. These results also confirm, to a greater extent, the different behaviours observed in the previous section: BOLA presents a more *aggressive* behaviour, while PANDA and AdapTech both present a more *conservative* one. Indeed, the parallel curves associated to BOLA (the brown ones in the plot), presents an adaptation strategy able to provide a higher average quality ($\bar{q}$) to the detriment of rebuffering events (in some cases) and quality switches: both their number ($\#QS$) and their frequency ($\bar{f}_{QS}$) are, on average, higher with respect to PANDA and AdapTech. Furthermore, BOLA presents the largest magnitude of quality switches; this outcome is linked to the higher $\bar{f}_{QS}$ and to the way $|\Delta(QS)|$ is computed (i.e., since quality switches are more frequent, it is less likely that the requested quality remains the same for a considerable number of consecutive segments, which would, in that case, reduce $|\Delta(QS)|$ by adding null terms). Nevertheless, in the best BOLA setting, corresponding to a stable buffer threshold of 18 seconds, the drawbacks are limited: there are no rebuffering events occurring, the average quality is high, while the number and the frequency of quality switches are significantly reduced compared to other BOLA settings, while the average magnitude of the quality switches is almost in par with AdapTech and PANDA.

AdapTech and PANDA, on their side, offer a greater *stability*, which translates into (i) a better quality smoothness, by virtue of less frequent quality switches of furthermore smaller amplitudes and (ii) a general absence of rebuffering events, with although the noticeable exception of two configurations of the aggressive version of PANDA [83]. However, the tradeoff for this increased stability of the video playout is a *smaller average quality* with respect to BOLA. To select the optimal settings for AdapTech, we noticed from Figure 3.5 that varying θ_1 and θ_2 induces much more variability in the number of quality switches than in the average quality, therefore, we select the setting configuration ($\theta_1 = 30$, $\theta_2 = 40$) that minimizes the frequency of quality switches, $\bar{f}_{QS}$. Finally, we rule out the aggressive configuration of PANDA as it introduces rebuffering events, which we want our selected strategies to totally avoid, and we select the least aggressive version [82] with $B_{min} = 44$s as the best PANDA setting configuration.

3.3.3 Multi-client scenarios

We next consider multi-client scenarios to assess if the selected calibration settings yield to consistent results. The topology used in these scenarios is described in Figure 3.6: four clients are connected to a router via wired links of capacity equal to 15 Mbps. The router is connected to a video server via a wired link of capacity 30 Mbps. The clients run an instance of our video player, while the server serves video segments of the TOS video.

More specifically, we design scenarios to study DAS performance with (i) a *homogeneous* client population (i.e., all the clients are using the same network stack, either TCP or NDN), as well as (ii) a *heterogeneous* population (i.e., half of

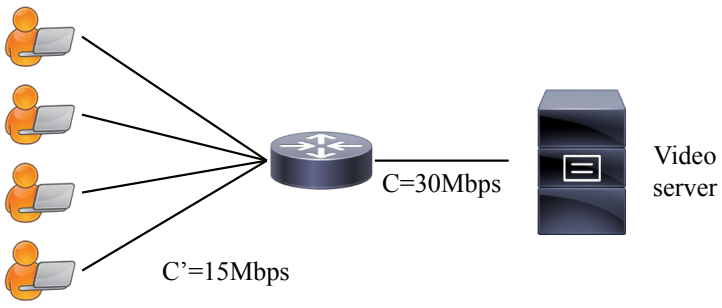


Figure 3.6: Topology of the multi-clients scenarios.

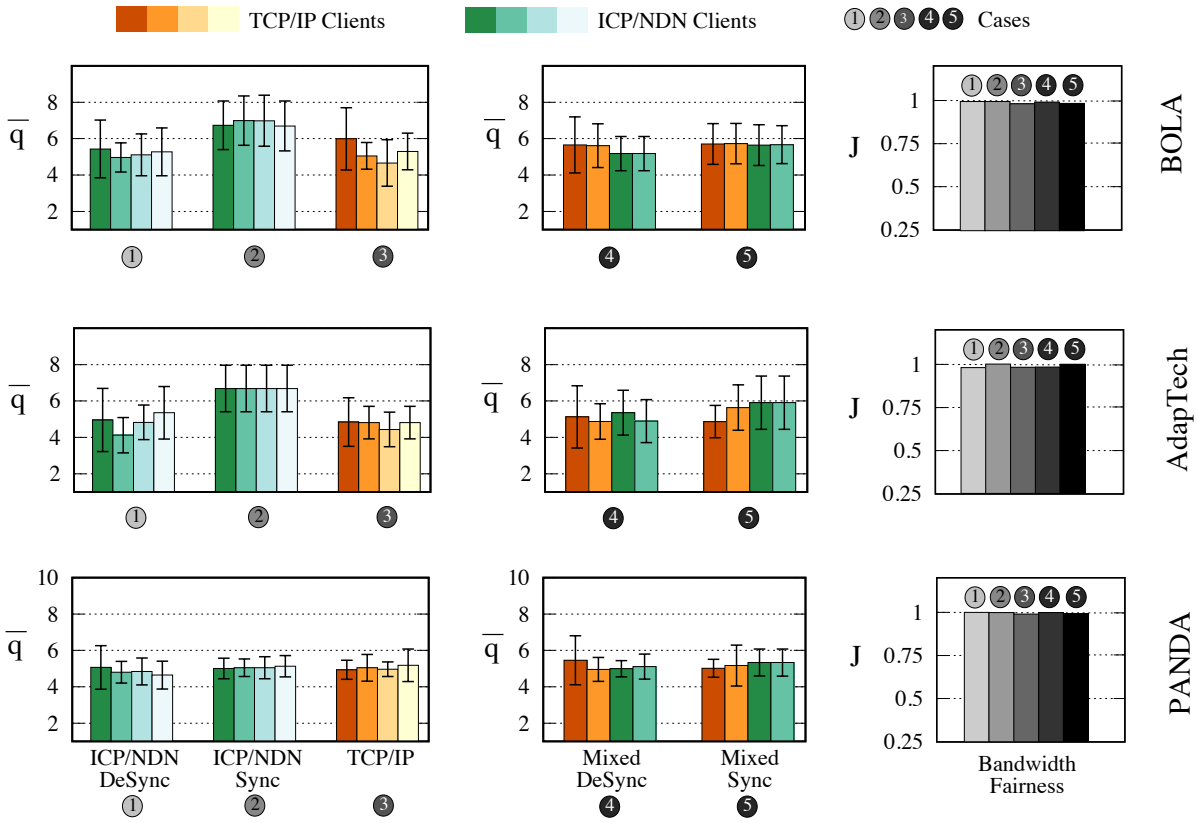


Figure 3.7: Multi-client Scenarios. *Average quality* and *bandwidth fairness* (with 95% confidence intervals over 10 runs) under homogeneous/heterogeneous populations, with synchronized/desynchronized client arrivals for BOLA (top), AdapTech (middle), and PANDA (bottom).

the clients use a TCP stack and the other half use an NDN stack), and we further distinguish the DAS performance between (i) *synchronized* and (ii) *desynchronized* client arrival patterns. Simultaneous arrivals (the synchronized case) closely represent a live-streaming case, while asynchronous independent client requests (the desynchronized case) correspond to a VoD case. Both arrival patterns are relevant from practical viewpoints.

In particular, we do not expect the synchronization scenario to have noticeable effect for TCP/IP: each client opens a connection to the server and starts requesting video segments over it. Therefore, the arrival patterns of the clients will have a minor impact on the quality of experience at the client, since the number of clients (4) in our scenarios is low, an overload of the server is highly improbable. On the contrary, in the ICP/NDN case, the synchronization of the client arrivals is expected to be beneficial, since it would gain from the Interest packets aggregation at

the PIT: in our case, the bottleneck is the link connecting the router to the server, when all the clients are downloading video segments from the server, the resulting bandwidth fair-share is $30/4 = 7.5$ Mbps, we expect PIT aggregation to form a multicast tree, leading the ICP/NDN clients to use the bottleneck bandwidth more efficiently.

The five cases we consider are reported in Figure 3.7. The picture reports, for each selected adaptation logic, the average quality, $\bar{q}$, for a homogeneous population of ① ICP/NDN asynchronous clients, ② ICP/NDN synchronous clients and ③ TCP/IP clients, as well as for a heterogeneous TCP and NDN population with either ④ asynchronous or ⑤ synchronous clients. The picture also reports (gray bars) the Jain fairness index [63] of the bandwidth share, useful to assess if some of the $N = 4$ clients starves the other ($J \approx \frac{1}{N}$), or if clients equally compete for resources ($J \approx 1$).

From these plots, we can gather three very important takeaways. *First*, as expected, NDN synchronous clients benefit from PIT aggregation as they increase their average video quality without increasing the upstream bandwidth: this is particularly visible for the BOLA and AdapTech strategies contrasting ② against ① and ③, where at least one quality level can be consistently gained in the emulation settings. It is worth noting that as the clients start at the same time, they request the same video segments (in a temporal viewpoint), and furthermore, as all the adaptation logics start by requesting the lowest quality, all the clients request the first video segment at the same quality at the same time and thus PIT aggregation happens at the router and only one request for the video segment is sent upstream towards the server. When the Data packets of this video segment is sent back by the server, the router sends them back to all the clients. As the clients receive the segments at the same time, their buffer level are the same, and thus when using mainly buffer-based adaptation logics (AdapTech, BOLA), they will request the next segment at the same quality, thus maintaining the PIT aggregation. However, the PIT aggregation requires all the clients to send Interest packets for the same Data packets at approximately the same time (one RTT $\approx 5 - 10$ ms in our settings), therefore, if there is a shift in the request patterns of the clients of more than one RTT, the PIT aggregation will not be observed anymore. Such a shift can happen when the clients introduce OFF periods in their downloading patterns, due either to a full buffer (and thus the client has to wait for room in the buffer) or to scheduling induced by the adaptation logic (which is the case in PANDA).

Second, from ④ one can easily observe that ICP/NDN appears to be no more aggressive than TCP/IP: the bandwidth share is fair and the average quality is in par or slightly lower, which is expected due to the differences in the window growth dynamics (delay-based and AIMD in ICP vs loss-based and MIMD in TCP

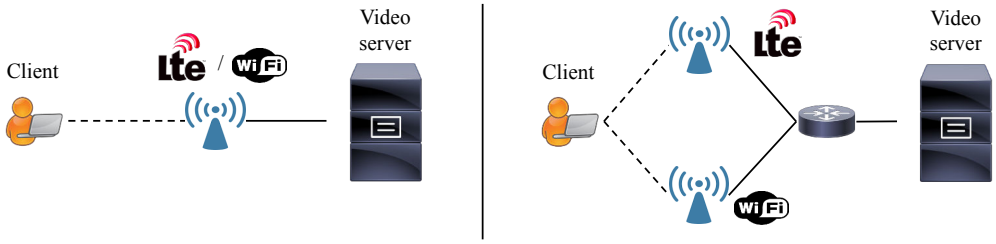
Cubic).

Third, and most important, from ⑤ one can gather that the previous properties combine: especially noticeable under AdapTech, the PIT aggregation makes synchronous ICP/NDN clients consume content as leafs of a multi-cast tree. This, on the one hand, improves the quality for ICP/NDN, and, on the other hand, reduces the used upstream bandwidth, which now becomes available for TCP/IP as well. Notice also that no side effects appear, as bandwidth share is still fair also under this circumstance.

To conclude this section, we validated that the expected benefits of ICP/NDN hold, and we additionally observe that the selected settings from the calibration for each of the adaptation logics are robust to multi-client scenarios as well.

3.4 Experimental Results

In this section, we carry out a fair comparison of DAS performance over TCP/IP contrasted to what is achievable on ICP/NDN. We incrementally take into consideration different features such as in-network loss recovery (Section 3.4.1), different granularities of the bandwidth estimation (Section 3.4.2), heterogeneous access technologies (Scetion 3.4.3) and in-network load-balancing among multiple paths (Section 3.4.4). The different scenarios explored in this section are depicted in Figure 3.8: a single client is connected to a server using various access technologies, emulated Wi-Fi, emulated LTE or 3G/4G traces, (Figure 3.8-(a)) or a single client is connected both to Wi-Fi and LTE access points in a multi-homed fashion (Figure 3.8-(b)).



(a)Single path: a single client is connected to a server via various access technologies (emulated Wi-Fi or LTE, 3G/4G traces)

(b)Multi-path: a single client is connected to a server in a multi-homed scenario, using emulated Wi-Fi and LTE access.

Figure 3.8: Topologies used during our experimental campaign.

3.4.1 In-network loss recovery

Using the NS-3 Wi-Fi model, we emulate a realistic lossy link for the topology described in Figure 3.8-(a). In order to get a bandwidth of approximately 6 Mbps, we set the distance between the access point and the client to 60 m. While in this case the TCP stack has decades of optimizations in recovering losses in an end-to-end fashion, a vanilla NDN stack presents additional challenges: thanks to its point-to-point nature, TCP can cope with losses by exploiting duplicated acknowledgments received by the sender endpoint, while the NDN *Data* packets sender endpoint might vary over time, making it difficult to learn about losses, even when piggybacking control information in subsequent *Interest* packets. Following from this, the simplest option for an NDN stack is to let the application handle losses by re-issuing requests after a timeout. However, this solution, albeit simple, is suboptimal, not only because it places the burden on the DAS application, but also because the selection of a proper value of the timeout is far from being trivial: note that the RTT may vary significantly as the *Data* packets can be retrieved from different endpoints. Therefore, a more suitable option is not to rely on endpoints to recover from losses but rather perform in-network loss recovery, which is especially useful for the first wireless hop. In this case, the access point can detect losses and retransmit earlier (up to one RTT) than in the TCP case. Without any loss of generality, we use the Wireless Loss Detection and Recovery (referred hereafter as WLDR) mechanism described in [28] (a brief description is also available in Section 3.2.2). It is important to note that this mechanism does not require any additional caches at network nodes, as it only leverages buffers on routers' linecards (of about 1 MB).

Figure 3.9-(a) reports, for all the selected adaptation logics, the selected video quality (left) and the player buffer occupancy (right) over time, both for TCP/IP (green curves), vanilla ICP/NDN (i.e., without WLDR, brown curves) and ICP/NDN with WLDR (gold curves). From this plot, the impact of in-network loss recovery is clearly visible: for all the selected strategies, vanilla ICP/NDN does not guarantee the same performance as TCP/IP in terms of selected video quality (which is consistently one level lower). Conversely, the ICP/NDN stack with WLDR equals the performance of TCP/IP, showing that the loss recovery mechanism, albeit needed, can be easily implemented directly at the network layer in NDN, so outside the transport layer (as opposed to TCP/IP). Furthermore, an in-network loss recovery mechanism such as WLDR would reduce the communication overhead with respect to a TCP retransmission, as an in-network loss recovery occurs directly between two adjacent nodes, while a TCP retransmission needs to traverse the whole path from client to server.

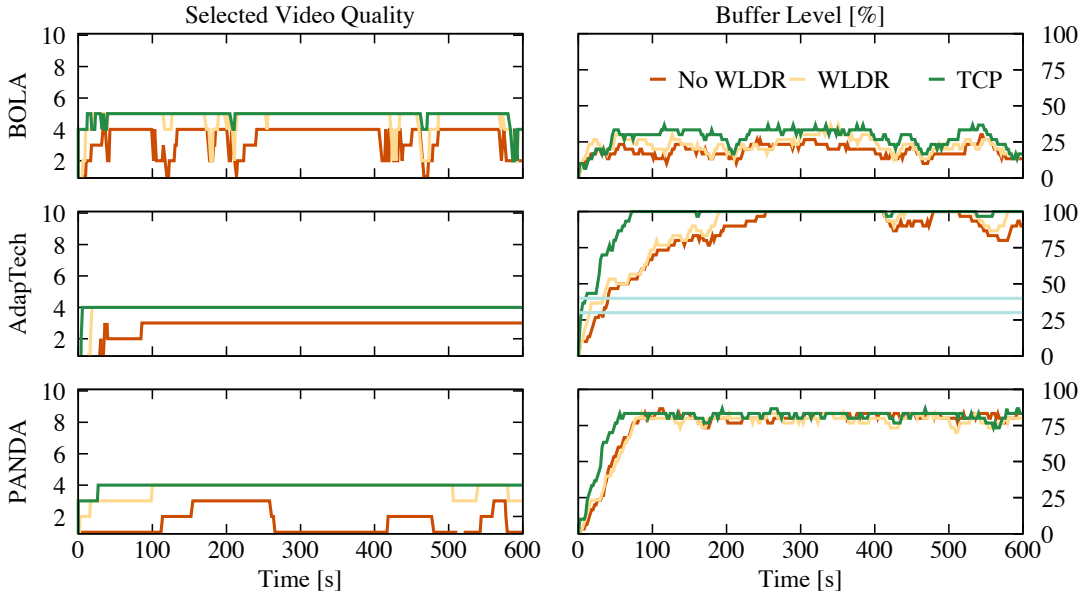
3.4.2 Bandwidth estimation granularity

Figures 3.9-(a) and 3.9-(b) present the impact of bandwidth estimation on the DAS performance by contrasting a coarse-grained throughput estimation against a fine-grained one. Specifically, in Figure 3.9-(a), each video segment constitues a bandwidth sample, while a sample in Figure 3.9-(b) results from averaging estimates over 50 NDN packets, as also proposed in [138]. As a consequence, the number of available samples in the NDN case can grow up to two orders of magnitude more with respect to the TCP segment-based estimate (recall Figure 3.2), thus resulting in valuable extra information that can be used to implement a more timely and refined estimate of the available bandwidth.

While we are aware that more sophisticated approaches would be possible (e.g., packet-pair for capacity [70], train or chirps for available bandwidth [99, 47, 116], possibly in band with the data transfer [102]), our main interest here is not to quantitatively assess a specific mechanism, but to point out qualitative properties that can be expected from this building block. Furthermore, provided the availability of the complex aforementioned techniques for the TCP case, the resulting estimate would, however, be available only at the server side, requiring out-of-band protocols to signal it to DAS clients (differently from the NDN case).

When comparing Figure 3.9-(a) to Figure 3.9-(b), we notice that the instantaneous bandwidth variations are better tracked by a fine-grained estimate, which allows DAS clients to better exploit the available capacity. However, a more responsive adaptation logic might result in an increased aggressiveness, as shown by AdapTech and BOLA, where the number of quality switches increases, if it is not smoothed by the strategy itself, as shown by the conservative version of PANDA, which takes advantage of the finer-grained estimate by increasing the selected video quality (with respect to the TCP/IP case) without any side effect. It is also worth noticing that while a fine-grained bandwidth estimate allows to a better use of the available capacity and thus downloading at a higher rate, the buffer level is lower with respect to the TCP/IP case and can, in some cases, lead to rebuffering events, for instance when using PANDA with a vanilla NDN network stack (without WLDR). Therefore, we are not advocating to indiscriminately use fine-grained bandwidth estimates (e.g., see the increase in quality shifts in some cases), we consider more accurate techniques to estimate the available bandwidth as a useful building block when coupled to, for example, in-network load balance, where the availability of multiple (independent) channels can be exploited to either increase the selected video quality, or guarantee the same quality if some of them experience bad conditions: in these cases, being able to closely track channel evolution would allow to fully benefit from the aggregate capacity.

(a) Coarse granularity, per video segment bandwidth estimation (TCP-like)



(b) Fine granularity, per NDN-chunk bandwidth estimation (NDN-like)

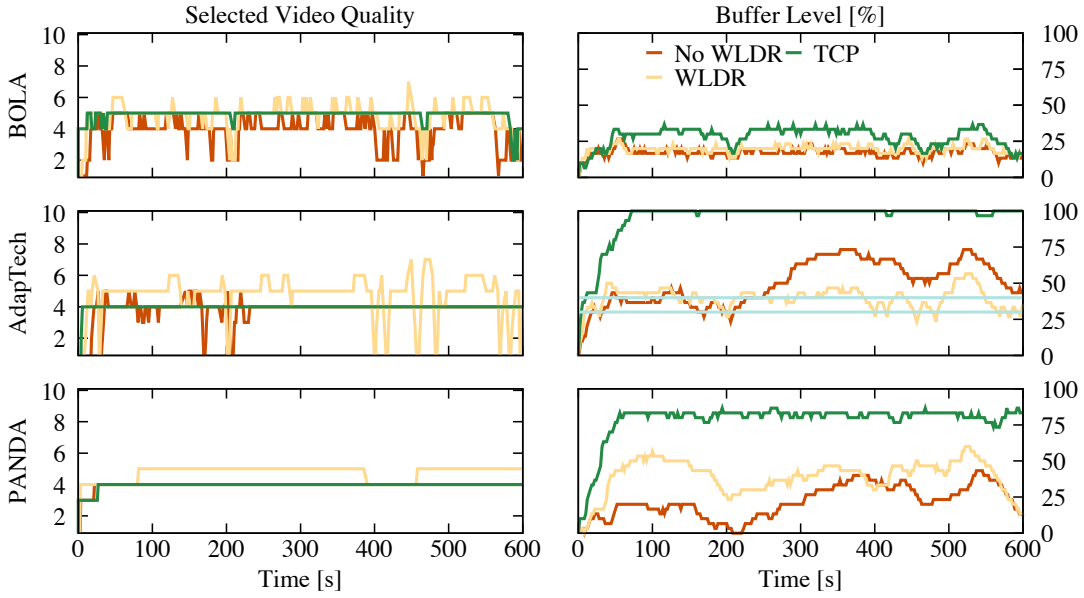


Figure 3.9: Impact of in-network loss recovery and bandwidth estimation granularity: (a) when *a coarse video-segment* granularity is used (for both NDN and TCP), NDN+WLDR performance matches that of TCP. However (b) when *a per-packet granularity* is used (for NDN only), it can be seen that more bandwidth can be exploited, making the protocol more aggressive and thus either better performing (PANDA) or prone to more quality switches (AdapTech).

3.4.3 Access technology and emulation technique

In order to confirm the findings discussed in the previous sections in more realistic conditions, we now contrast DAS performance gathered via *emulated channel models* (Wi-Fi and LTE) against those collected using *real traces*. In particular, we use both 3G [4] and 4G [5] real traces, available at [117, 136]. For the emulated cases, we set the distance between the client and the Wi-Fi access point to 60 m, resulting in a downstream capacity of approximately 6 Mbps, while the distance between the client and the LTE base station is set to 1200 m, thus offering a downstream capacity of approximately 18 Mbps. We remark that if, on the one hand, emulated models have the benefits of yielding arbitrarily long stochastic processes – which ensure statistical relevance of the experiments over multiple independent repetitions, real traces, on the other hand, represent samples of finite length, but of real conditions – without requiring complex calibrations.

Figure 3.10 reports the DAS performance under the various access technologies: the bottom plot in the figure illustrates the available bandwidth for the different cases. Top plots report the detailed time evolution of the requested quality of the video segments along with the buffer occupancy at the client using AdapTech as its adaptation logic for each considered network stack (both TCP/IP and ICP/NDN) under each access technology: left plots present results for emulated Wi-Fi and LTE, while the right plots present results for 3G and 4G traces. Each time plot is annotated with a tuple summarizing the performance details. To recall, the tuple presents the average quality ($\bar{q}$), the number of quality switches ($\#QS$), the frequency (f_{QS}) and the magnitude of the quality switches ($|\Delta(QS)|$), along with the number of rebuffering events (R) and the time spent rebuffering ($RTime$). Furthermore, for each access technology, a bar chart shows the average requested quality (along with 95% confidence intervals bars over 10 repetitions) for all the selected adaptation strategies. It can be observed that, despite differences in the stochastic nature of these processes, there is an agreement between the available bandwidth and the average qualities: e.g., emulated Wi-Fi and trace-driven 3G performance are similar for all DAS strategies, and the same holds for emulated LTE vs trace-driven 4G. Overall, the comparison of both methodologies allows to conclude that performance gathered over emulated models are not only statistically relevant, but also qualitatively and quantitatively in agreement with real trace driven conditions.

[4] `http://home.ifi.uio.no/paalh/dataset/hsdpa-tcp-logs/bus.ljansbakken-oslo/report.2010-09-28_1407CEST.log`

[5] `http://users.ugent.be/~jvdrhoof/dataset-4g/logs/report_bus_0006.log`

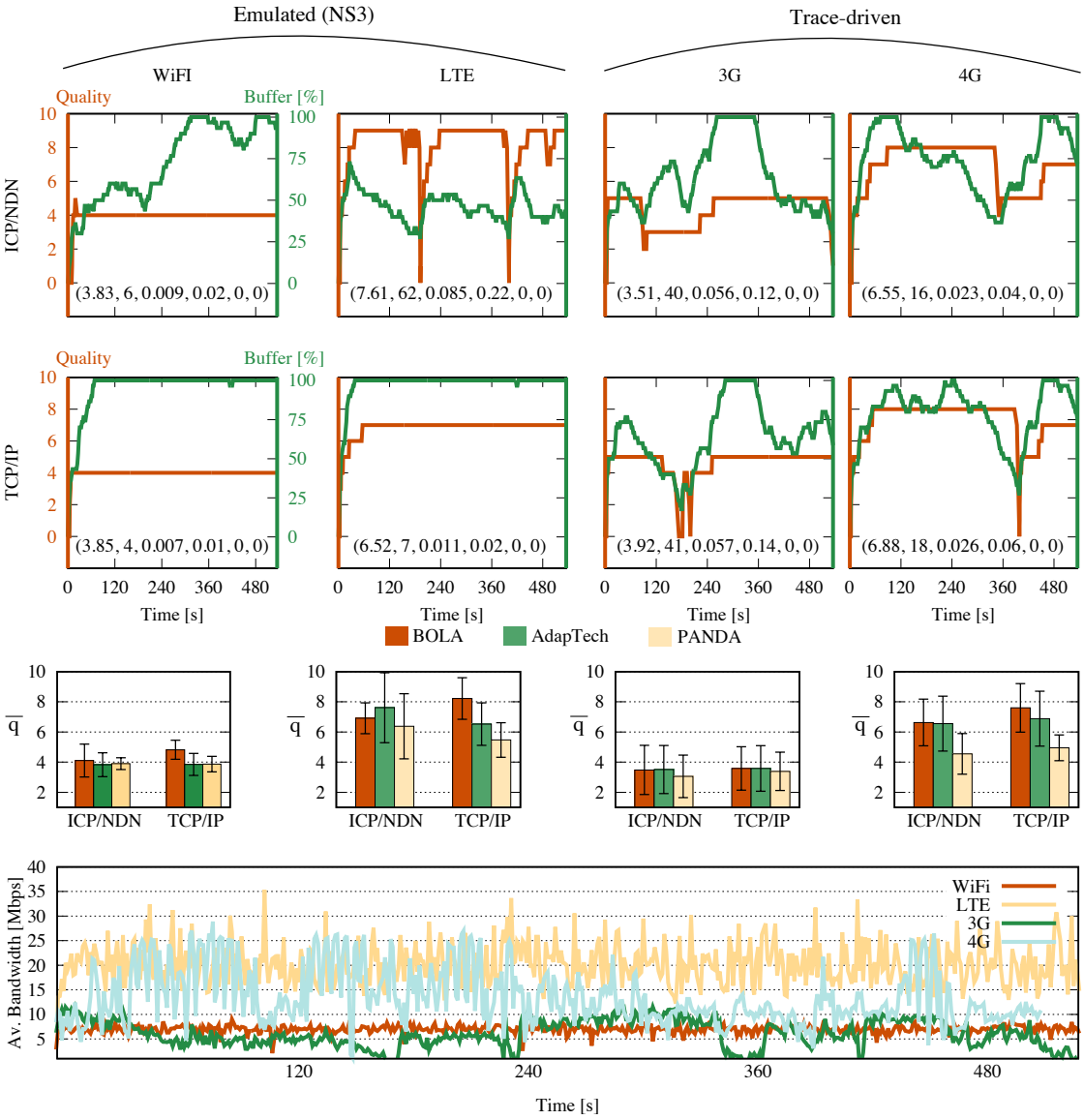


Figure 3.10: Access technologies: Emulated NS3 Wi-Fi and LTE (left columns) vs Trace-Driven 3G/4G (right columns). Top picture reports the AdapTech case as an example, annotated with $(\bar{q},\#QS,f_{QS},|\Delta(QS)|,R,RTime)$ performance details. Middle bar charts report average quality (with 95% CI over 10 runs) for all DAS strategies. Plot in the bottom row illustrates the available bandwidth with the different access technologies and emulation techniques.

3.4.4 In-network load-balancing

In this section, we consider the case where a client, using a ICP/NDN network stack with WLDR enabled, is multi-homed with heterogeneous wireless technologies, as depicted in Figure 3.8-(b). Specifically, we restrict our attention to emulated WiFi and LTE conditions, which we expect to be both statistically relevant and with a sufficient degree of realism for our purposes. The distance to the Wi-Fi access point is set as in Section 3.4.1, while the LTE base station is placed at 1400 m, offering a bandwidth of approximately 16 Mbps.

The NDN client performs load balancing of *Interest* requests, and as the corresponding Data packets are routed back to the user using the PIT, they are load balanced as well. We consider a simple algorithm [25], where clients monitor the number of Pending Interests (PIs), which are the sent *Interest* packets that are not satisfied yet, for each prefix associated to a face. Any new Interest packet for a given name is sent on a face for the matching prefix with a probability that is inversely proportional to the PIs of that face (normalized over all faces). This algorithm is motivated by the intuition that a face with many pending Interests is slow to respond, whereas a face with no pending Interest is likely underutilized.

We do not engineer load balancing on the TCP/IP case, as it would be significantly complex: this is well explained in [57], which testifies the complexity that would entail an architecture using range-requests to load balance requests at sub-video-segment level. At the same time, we argue that a TCP/IP load balancing would, as for the bandwidth estimation, likely be performed at video-segment level. Since ICP/NDN with WLDR roughly matches TCP/IP performance in the single-path case (recall Section 3.4.1), we argue that ICP/NDN with WLDR performing load-balancing at the video segment level would roughly match a DAS system performing load-balancing at video segment level over multiple-paths via a TCP/IP stack, at a smaller implementation cost. We therefore implemented two load-balancing algorithms, a per-Interest one, that is described earlier, and a per-video-segment one: when a new video segment is to be downloaded, the client randomly picks one face matching the requested name and all the subsequent Interest packets for that video segment will be sent on that face.

Figure 3.11 reports the DAS performance using the two load-balancing algorithms: the per-Interest load-balancing (identified by the red curves) and the per-video-segment load-balancing (represented by the green curves). The plots in the top row present the requested quality for the video segments, while the bottom plots report the EWMA of the split ratio over the LTE interface, i.e., the percentage of Interest packets (resp. video segments) that are sent (resp. requested) over the LTE interface. Specifically, two curves for the split ratio are shown: the light-colored one gives more weight to the instantaneous sample ($\alpha = 0.7$) in order to gauge the variability of the split ratio, whereas the thick-colored line is a heavily

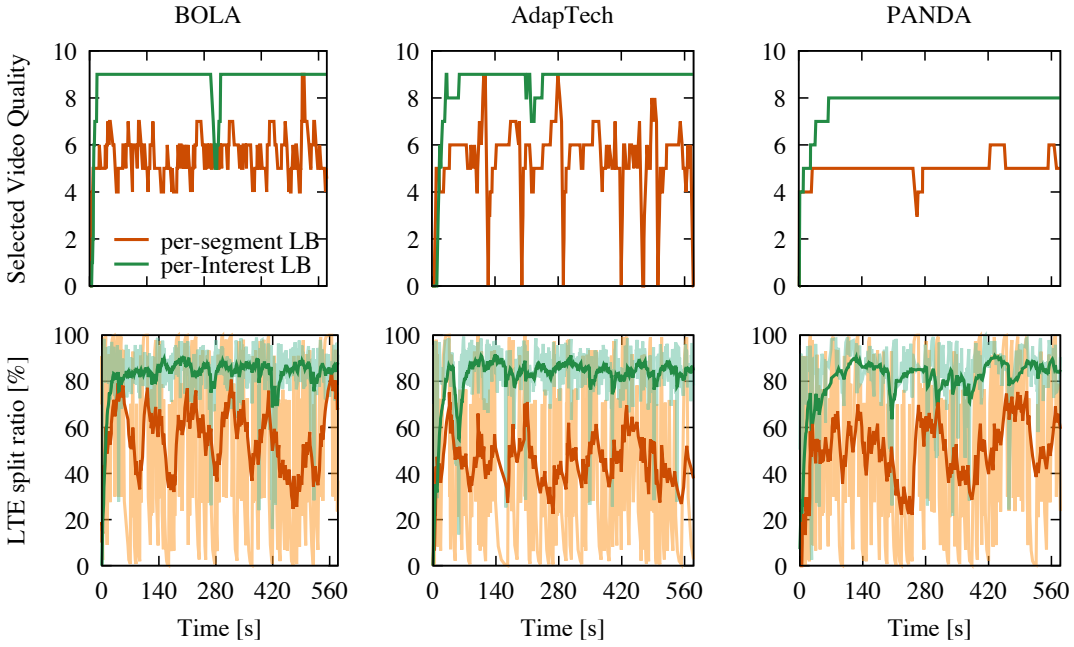


Figure 3.11: In-network support: load-balancing among WiFi and LTE interfaces. Top plots show the instantaneous requested quality for segment vs Interest-level load balance. Bottom plots show the percentage of segments vs *Interest* packets aired over the LTE interface using EWMA smoothing.

smoothed version ($\alpha = 0.1$) to make the average split clearly readable.

Figure 3.11 shows that only per-Interest load balancing allows to profit from the aggregate bandwidth, while segment-level one is only partly helpful, and often even counter-productive. Notice that, by performing fine-grained load balancing decisions, BOLA, AdapTech, and PANDA not only exhibit a tremendous gain in terms of the average quality increase, but also in terms of stability. This is due to the fact that (i) fine-grained bandwidth estimation, coupled with (ii) fine-grained forwarding decisions, make these algorithms able to aggressively and promptly react to changes in the channel. Additionally, the stochastic variability that negatively affected stability of the requested quality in the single channel Wi-Fi case, is no longer a problem, since channels are independent. Conversely, per-video-segment decisions forbid these algorithms to fully exploit the aggregate capacity, since entire segments are downloaded over a single channel; this means that, even in case of severe channel variations, the algorithm has to finish the current segment download before switching interface, thus leading to undesirable quality switches. It is worth noticing, in the end, that PANDA turns out to be the less aggress-

ive adaptation logic[6], being in line with results shown previously. In particular, when load-balancing decisions are taken per video segment, the quality shifts are drastically reduced with respect to BOLA and AdapTech, while in the case of per-Interest load-balancing the average quality remains constantly at one level below compared to BOLA and AdapTech.

3.4.5 Summary

Qualitative summary

Figure 3.12 summarizes the main findings of the experimental campaign, for the AdapTech strategy. We choose to present only the results for AdapTech to avoid cluttering the picture and the results presented with AdapTech still hold for the two remaining adaptation logics (BOLA and PANDA). The picture is a scatter plot where the axes represent two important KPIs: the x-axis represents the average quality $\bar{q}$, while the y-axis represents the number of quality switches $\#QS$ for different TCP/IP or ICP/NDN configurations. To ease off the comparison, the origin of the axes is set as the values gathered in the TCP/IP case (red square). To ease off further the reading of this plot, we annotate each point with the actual averages ($\bar{q}$, $\#QS$). The picture shows that vanilla configurations of ICP/NDN, i.e.,when no in-network loss recovery capabilities are used, and irrespectively to the granularity of the bandwidth estimation technique ① and ② can hurt the performance of DAS systems; however, the use of in-network loss recovery ③ puts ICP/NDN in par with TCP/IP when the bandwidth estimation is performed at video-segment level. Additionally, an NDN sender has the opportunity of tracking more closely the bandwidth variations, thereby being more aggressive in the requested quality, which increases both the average quality as well as the quality switch rate ④. This is expected on a single channel, while when adding multi-path functionalities, which are very simply implemented in NDN, one can leverage statistical multiplexing to smooth out variability of bandwidth and losses. The gain in average quality is already sizeable when load-balancing is performed at video-segment level ⑤, which could also possibly be implemented (with some significant effort) in TCP/IP; however, the very large size of video-segments (several thousand packets at the highest quality level) may play against multi-path capabilities, still forcing undesirable quality switches. Conversely, when a fine-grained load balancing (i.e., NDN-chunk level) is used, the DAS system is able to fully exploit the available bandwidth with no penalty, i.e., almost doubling the quality with a minimal amount of quality switches ⑥ – interestingly, a packet-level tech-

[6]While it is possible to use the more aggressive PANDA settings, possibly exploiting at maximum the extra capacity, this is not an angle we deem of interest, because of the downsides (i.e., rebuffering events) we highlighted in the single channel scenario.

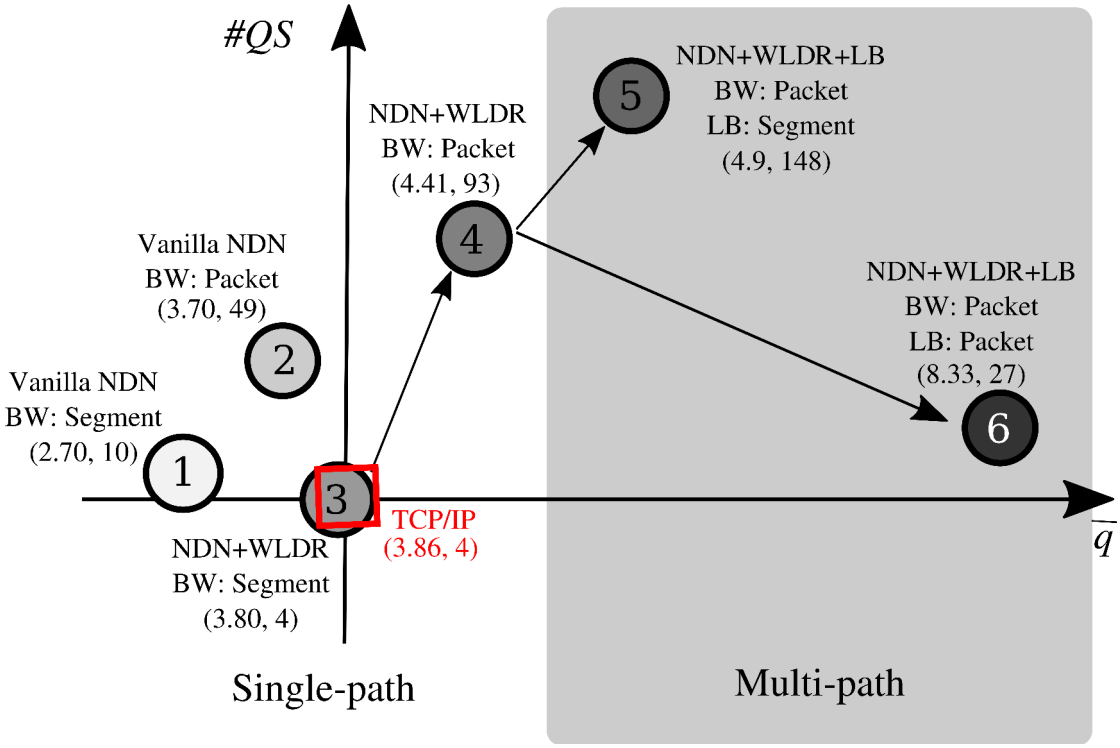


Figure 3.12: Scatter plot illustrating the effect of different ICP/NDN settings w.r.t. TCP/IP for the average video quality (x-axis) and number of quality shifts (y-axis) for AdapTech.

nique would not be advisable in the case of connection-oriented TCP, where letting packets follow disjoint paths with different bandwidth and latency characteristics would cause significant amount of out-of-order packet delivery, jeopardizing TCP congestion control. Clearly, cases ①-⑤ are the pitfalls to be avoided in order to attain the desirable ICP/NDN operational point ⑥.

Quantitative summary

We finally present in Figure 3.13 and in Table 3.1 the average performance of the different NDN settings ①–⑥ for the KPIs early used in the sensitivity analysis (with the noticeable exception of the number and duration of rebuffering events, as no rebuffering events occurred with our settings). To gather results that are not tied to a specific DAS strategy, Figure 3.13 reports results *averaged over all DAS strategies*.

Interestingly, the best ICP/NDN setting ⑥ significantly increases the average quality – by almost a factor of two. This means that one can expect consistent and considerable quality gains, that furthermore hold across strategies. Next, notice

that the quality increase for ⑥ does not mechanically translate into a higher number of quality switches, which remain close to that experienced in the TCP/IP stack. As such, one can definitively confirm the interest in a carefully configured ICP/NDN stack to enhance the performance of video streaming systems in future networks: the necessary building blocks to achieve this goal are (i) fine-grained bandwidth estimation at the ICP transport layer, coupled to (ii) fine-grained load-balancing decisions among heterogeneous interfaces at the NDN client side, and (iii) in-network loss recovery through the use of WLDR.

Conversely, other NDN settings (e.g., ④ and ⑤) lead to a more modest increases in the average quality, at the price of a significant increase of the quality switches. In line with studies that model how these objective metrics translate into user Quality of Experience (QoE) [16, 56], we observe that a high number of quality switches may not be desirable since it can offset the gain in the average quality. Particularly interesting is the fact that setting ⑤ employs all ingredients of ⑥ with a single difference: i.e., the granularity of the load balancing decisions, that are taken at video-segment level. We can thus argue that the use of multiple paths could be difficult in the TCP/IP world, where decisions are likely to happen at this level of granularity [57], as this may ultimately harm user experience as remarked in [64, 32].

Finally, other naive ICP/NDN settings are less interesting as they either match ③, or even worsen ②-① performance with respect to TCP/IP. These settings correspond to a poor use of bandwidth estimation (①,③), or to the lack of network support for loss recovery (①,②).

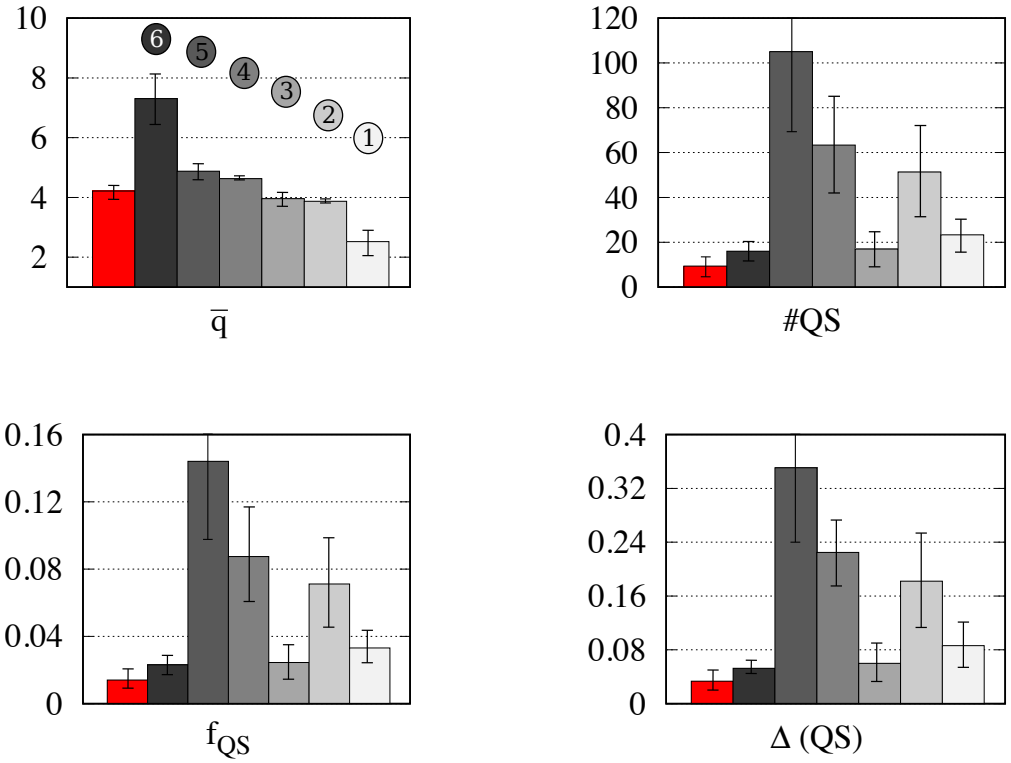


Figure 3.13: Bar chart illustrating the effects of different ICP/NDN settings (shaded gray) w.r.t. TCP/IP (red) for all the considered metrics: average video quality $\bar{q}$, number $\#QS$, frequency f_{QS}, and amplitude $|\Delta(QS)|$ of quality shifts. We report bars representing averages over the three strategies, along with standard deviations.

Table 3.1: The performance of the different ICP/NDN settings w.r.t. TCP/IP, decomposed in all the considered metrics, averaged over all adaptation logics (BOLA, PANDA and AdapTech). Specifically: average video quality $\bar{q}$, number $\#QS$, frequency f_{QS}, and amplitude $|\Delta(QS)|$ of quality shifts, number of rebufferings R, and rebuffering time $RTime$.

	TCP	6	5	4	3	2	1
$\overline{q}$	4.21	7.30	4.88	4.63	3.96	3.87	2.52
$\#QS$	9.33	16.00	105.00	63.33	17.00	51.33	23.33
f_{QS}	0.014	0.023	0.144	0.087	0.024	0.071	0.033
ΔQS	0.033	0.052	0.350	0.225	0.060	0.182	0.086
$\#R$	0	0	0	0	0	0	0
$R\ Time[s]$	0	0	0	0	0	0	0

3.5 Conclusion

In this chapter, we contrasted the performance achievable by DAS systems using rate-based vs buffer-based adaptation logics developed on top of an ICP/NDN or a TCP/IP network stack. Our approach is experimental and based on a real prototype, which we make available as open source software (Appendix A), along with the necessary scripts to seamlessly repeat part of our evaluation.

Our experimental campaign includes multiple videos (up to 4K resolution at 18Mbps), multiple channels (including DASH profiles, as well as Wi-Fi and LTE access emulated via NS-3, or real 3G/4G traces), multiple clients (in homogeneous and heterogeneous population mixture, with synchronous and asynchronous arrival patterns) and multiple adaptation logics (PANDA, AdapTech, and BOLA). Concerning the ICP/NDN settings, we experiment with several building blocks that include bandwidth estimation, use of multiple heterogeneous interfaces, and in-network loss recovery. Our findings are that performance of ICP/NDN easily matchs and possibly significantly outperforms that of TCP/IP. While this is achievable by combining relatively simple building blocks, we also find that all these blocks are *jointly* needed, and that ICP/NDN performance can just match or even worsen with respect to TCP/IP in the other cases. We argue that our findings do not depend on the specific architecture design we used (NDN) and still hold under another architecture design for ICN. For instance, [121] results are in par with those presented in this chapter, and were gathered using the CICN architecture.

Overall, we believe the work done in this chapter constitutes a first milestone towards a fair and complete assessment of fully fledged ICN video distribution systems, and their comparison with state of the art CDN technologies implemented over a classic TCP/IP stack. The following step to achieve this more ambitious goal, would be that of contrasting the two alternatives in more realistic scenarios (e.g., more complex topologies with several origin servers, multiple videos, realistic user arrival and mobility patterns, etc.). This would allow to better grasp pros and cons of the two architectures (e.g., CDN request redirection and load balancing vs ICN multicast and multipath support), as well as to assess their impact on the overall performance from the user viewpoint.

Finally, in this chapter, we purposely left aside the presence of in-network caches that can be leveraged in an ICN environment to have a fair comparison with TCP/IP stacks. In the next chapter, we explore how in-network caching can be exploited in an ICN environment to improve further the quality of experience of DAS clients.

Chapter 4

Network-Assistance in video delivery

Contents

4.1 Introduction

In this chapter we focus on how assistance from the network can affect dynamic adaptive streaming. *Network functions*, such as in-network caching and multi-path forwarding, along with *network frameworks* (such as Server and Network Assisted DASH [135] or ICN), are good candidates to reduce the pressure put on the network infrastructure by video streaming. As multi-path forwarding was addressed in the previous chapter, we will concentrate here on in-network caching. While the existing DAS literature abounds with adaptation logics [71], there is no systematic study of the controller[1] interaction with network functions, but Section 4.2 will give an overview of the existing works in this area. In particular, we argue that it would be desirable for any network-controller mechanism to be as lightweight as possible, thus limiting the *amount* of information needed to be collected from the network and disseminated to all the clients, as well as the *rate* at which the information need to be disseminated. Additionally, such signal should be easily pluggable within existing DASH controller, to naturally extend their logic by fitting the additional information provided by the network, as opposite to require a complete redesign around it. Our goal here is indeed not to propose another adaptation logic but rather propose a network signal that can be exploited by existing adaptation logics without much efforts.

It is known that in-network caching can positively impact the quality of experience by improving the requested video quality and relieving traffic load on servers, as we show in [121], proactively caching video content at the network edge effectively relieves traffic load from the core network.However, in-network caching can negatively affect the controller stability, by increasing the quality switch ratio, and thus inducing quality oscillations [78]. To reduce this cache-induced oscillation, bandwidth shaping is seldom used to reduce the rate toward the cache [67] (which, in practice, limits the cache usefulness), or the assistance of a centralized resource manager is usually assumed [20] (which requires global instantaneous knowledge and decision of all cached content and is therefore too heavyweight to have practical relevance). More specifically, our goal is to understand the feasibility of a network-controller interaction that (i) uses the most lightweight network signal, (ii) requires only minimal changes to existing DASH adaptation logics to take into account this signal and (iii) maximize cache usefulness and thus increasing the average video quality, while (iv) avoiding cache-induced oscillations.

To summarize the main takeaways from this chapter, we first expose in Section 4.2 the various works that exploit network interactions with DAS clients and then we systematically assess the impact of in-network caching when several ICN

[1] A controller is defined as in Section 3.2.2 : it is an entity that drives the video segment request at the client following a given adaptation logic.

DAS players compete to watch a video, exploring the boundaries of the design space for network versus client interaction using various adaptation logics (Section 4.3) and then we propose and evaluate (Section 4.4) a network-aware evolution of an existing adaptation logic (specifically, AdapTech [15]), based on simple network signals (specifically, per-quality cache hit-rates) exported at low rate timescales (specifically, tens of seconds). We conducted an experimental campaign, worth *several weeks* of video streaming, to confirm the soundness of our approach: by integrating this simple network signal to an adaptation logic, we significantly increase the DAS performance with respect to a network-blind adaptation logic. The results gathered over this campaign also show the robustness of the network signal used, i.e., tuning is not critical.

The results presented here were published in [123], and received the NOSSDAV 2018 best paper award.

4.2 Related Work

HTTP Adaptive Streaming (HAS) systems traditionally delegate bitrate selection and control to the client application, which takes independent decisions based on local estimates of available end-to-end bandwidth and awareness of buffer level dynamics. Recently, many studies have shown the benefits of network assistance in HAS to overcome the limitations of a purely client-driven scheme (see e.g. [15, 36, 44, 23, 30, 72, 127, 20]). The lack of direct knowledge about network status (congestion, bottleneck, cached content) as well as of coordination among concurrent flows may result in frequent quality/bitrate oscillations, sub-optimal bitrate decisions by the clients, unfairness among ongoing flows and inefficient overall utilization of network resources. In addition, service differentiation and management policies cannot be guaranteed in purely client-driven HAS. A dedicated MPEG standard has been developed to formalize the interaction between client and network elements under the name of SAND, Server and Network Assisted DASH([135]). For the sake of simplicity, we classify previous work on network-assisted video delivery in two classes: the first considering the assistance of a single network element (i.e. server, intermediate cache, edge router), the second considering more network elements and involving complex control systems (e.g. CDNs, SDN-based networks). Within each class, we also highlight the work leveraging an Information-Centric approach. Finally, Table 4.1 presents the previous work, explicitly separating the work using a single network element versus using multiple network elements, with a further division of the work done in the TCP/IP model versus in the ICN domain.

4.2.1 Single element network assistance

Cache-assisted. In [78], authors observe that the deployment of cache servers within the access network to cache video segments may cause incorrect estimation of the available bandwidth at the client due to a possible overestimation of the available bandwdith in case of retrieval from the cache and thus lead to bitrate oscillations and lower QoE. They propose ViSIC (VIdeo Shaping Intelligent Cache), an intelligent cache indirectly assisting bitrate adaptation by performing shaping of traffic from the cache to prevent oscillations.

[69] proposes cache-assisted HAS. It estimates the bandwidth between the client and the cache to orchestrate prefetching of segments from the server to match such estimate, as to avoid cache-induced oscillations. To this aim, it requires the cache to be aware of the MPD and of the adaptation logic used by the client(s).

[110] presents a cache-aware version of MPC [141], where two bandwidth estimations are kept at the client, one for the throughput to the cache and one to the server. It also proposes an upper-bound for their cache-aware predictor to MPC, where the cache state (i.e., which segments are cached) is perfectly known at any point in time.

Proxy-assisted. [39] considers multiuser DASH distribution over a mobile network access and proposes to optimize QoE by means of a proxy-assisted in-network adaptation: a target transmission rate per user is determined and user requests are modified at the proxy to match the computed optimal streaming rate. A similar approach is described in [93], where authors introduce QoE-DASH,a proxy-based system performing throughput measurements in the network and providing instructions to the client application about bitrates to select over time.

Server-assisted. [23] shares the same objective of maximizing QoE in a multiuser HAS by means of an ILP optimisation solved in a centralised or decentralised way. The optimization problem takes into account the impact of several factors, such as number of clients, number of bottlenecks, latency, number of servers. In case of centralized optimization, the server recomputes the optimal constant bitrate to assign to each user every time a new client request arrives. As in proxy-assisted HAS proposals, the capability to adapt rate at the client is traded-off for a global optimization of resources' allocation. In our work we take the purely client-driven and network-driven solutions as references and investigate the space of solutions in between combining network awareness and user QoE for informed client-driven rate adaptation.

ICN-based

[48] starts by illustrating the downsides of video delivery in presence of in-network caching: decreased cache hits and oscillations in video quality at the client if the

path client-cache is significantly better than client-server. It then introduces some cache assistance for DASH over ICN in order to avoid oscillations and reduced cache hit ratio. The cache is given the ability to modify the MPD, to advertise its cached qualities. Under the assumption that a distinction can be made between a content served by the server and a content served by the cache, the client can then manage two rate estimations: one for the server and one for the cache. We adopt a similar per path estimation in our work. To reduce bitrate oscillations and to maximise caching effectiveness, the intermediate cache performs live transcoding. The drawback is that transcoding is heavy and not really scalable.

A similar cache-assisted HAS approach is developed in [111] under the assumption of SVC (Scalable Video Coding). An intermediate ICN router caching video segments monitors all the requests and then chooses to forward or drop the requests, sending in the latter case a NACK to the client. The work also shows the effect of video popularity (starvation of unpopular videos watchers that compete with popular videos).

Recently, [67] has suggested shaping at the cache to avoid oscillations. The shaping is done to guide the client to the next decision: if the next segment is present in cache, at a given quality, the shaping will be done to match the bitrate of this quality. The drawback is a diminished adaptiveness to variable network and client conditions which we try to avoid in our work.

4.2.2 Multi-element network assistance

The body of work surveyed in this class focuses on multi-element network assistance by designing a control plane assisting video delivery in multi-user, multi-cache environments (e.g. CDNs) [85, 96, 43, 33]. The video control plane aggregates user and network periodical measurements of parameters impacting video distribution to feed global optimization based on the inferred network view. These works assume a centralised system where the collection of information and the coordination of client decisions take place, under either centralised or distributed optimisation.

The definition of the utility function may differ from one study to the other, as well as the characterization of network parameters impacting user QoE. For instance in [33], authors focus on network link congestion and use it at an aggregate network level to drive local maximization of user QoE. A coordinator node is utilized to aggregate network and client information and send back updates to the clients to perform the next bitrate selection. Fairness is the key metric considered in [51]. Indeed, the paper deals with the competition between HAS and TCP flows. It derives an optimal bandwidth allocation scheme to guarantee fairness and stability to HAS and to TCP flows. Also, it considers service differentiation and applies bandwidth access priority to flows.

SDN control. [98] leverages SDN to periodically collect application measurements from the clients (such as buffer level) and network measurements (such as congestion) and enforce decisions like route recomputation upon detection of congestion with the aim of maximizing clients' QoE.

[44] uses centralised SDN control to achieve QoE fairness between several competing HAS players (a TV, a smartphone and a tablet). Similarly, [109] focuses on avoiding rebuffering events at the player by prioritising streaming flows from clients with a low buffer level. This is achieved by having the clients modifying their HTTP requests to integrate their buffer level. Then a centralised SDN-based controller is able to extract the information from the request and to calculate player likelihood to experience a stall. Based on that, prioritization of flows and rate switches at client side are determined. [127] performs a comparison of these two approaches and more generally characterizes the interaction of Application-Level Optimisation (ALO) and Network-level Optimisation (NLO).

[19] proposes an SDN-based dynamic resource allocation and management architecture for HAS systems where resources are allocated dynamically for each client with the aim to maximize per-client QoE, whilst taking into account different clients needs (e.g., buffer sizes, display resolutions and quality requirements, etc.) and network requirements (e.g., available bandwidth, latency, etc.)

Another SDN-based framework is proposed in [30] to perform bandwidth reservation and provide bitrate guidance to clients. It investigates how these strategies can impact the video delivery. Centralised optimisation to provide fairness in terms of video quality, formulated as a max-min fairness problem. This results in having a bitrate assigned to each client and then one of the three strategies is applied: (i) *bandwidth reservation*, where a slice of bandwidth, matching the computed bitrate, is assigned to the client, still deciding independently about rate adaptation, (ii) *bitrate guidance*, where the result of a global optimisation is communicated to the client to driven bitrate decisions, (iii) a hybrid of the first two strategies, namely both bandwidth reservation and bitrate guidance are performed in a centralized way.

[72] addresses the bursty nature of DASH by ressource allocation and player assistance during playback. Upon the start of a streaming session, the client extracts from the DASH manifest the different video bitrates and selects the one it can use (based on constraints on the client side, e.g., if the bitrate is too high and exceeds the capacities of the device).Then the client sends this selection to a service manager, that will do the ressource allocation. If the capabilities of the device evolve over time (switch to full stream), the client can send an updated selection of bitrate to the service manager. They compare different ressources allocation and show that DASH performs better when there are two queues in the network devices : one for DASH players and one for the rest of the traffic. They

show that enforcing the bitrate selection at the service manager allows to lower significantly the number of quality switches but may induce rebuffering events as the service manager is not aware of the buffer level at the client side (to avoid this, they introduce a safety threshold).

[20] defines an SDN-based framework for ABR control in CDNs. Network measurements are periodically performed to monitor path conditions, the information collected is aggregated by an SDN controller referred to as SABR and communicated to CDN caches and end users, respectively to enforce specific content placement strategies and to lead to informed decisions at the clients based on the communication with SABR. Unlike [20] we leverage ICN reactive caching to optimize client bitrate selection in non-controlled network environments.

ICN-based

[115] investigates the performances of several forwarding strategies along with different caching policies in the network. In [79], authors design DASCache, a framework for optimal video caching in ICN which considers a controlled network of caches. DASCache performs online optimization aiming at optimal video content placement in cache. It works in rounds, by minimising the average access time per bits for each clients. When a round starts, information about popularity of content is collected at the edge routers. At the end of the round, a solver is called to solve a BIP (Binary Integer Programming) problem. Clearly, the approach is suitable for a small scale CDN-like environment where it is feasible to solve an online optimization to derive content placement strategies. To broaden the scope of previous work beyond the CDN case, we try to assess whether (and how) network-assistance can be beneficial also in non-controlled network environments like the mobile access and in presence of purely reactive caching with the objective of preserving dynamic bitrate adaptation at the client.

4.3 To Cache or not to Cache?

While caching can be beneficial to DASH by *increasing the average quality* (e.g., as typically the bandwidth to the edge-cache is larger than that toward the end-server), it may also negatively impact performance *by increasing the quality switch ratio* (e.g., in case of a cache miss, which can further lead to oscillations). To understand which conditions lead to performance impairment, and how to avoid it, it is important to systematically study the design space of DASH interaction with in-network caches.

4.3.1 Design space

We consider a reference baseline scenario (without in-network caches) and contrast it with in-network caching scenarios (with proactive vs reactive policies), considering both network-blind (buffer vs rate based) and network-aware adaptation: the cache advertises to the client which quality is cached, and the client can either have a strict (i.e., always follow the cache feedback and download the segments at the advertised quality) or soft (i.e., follow the cache feedback, unless it can be harmful to the client's QoE, e.g., requesting a high quality when the buffer level is low) interpretation of this feedback, to which we refer to as network feedback.

Table 4.1: State of the art of network assistance in dynamic adaptive streaming, over either TCP/IP or ICN.

		Reference	Main Approach	Shaping Traffic	Client Notification	Altered Requests	Resource Allocation	Client Collaboration
Single element Network assistance	**TCP/IP**	ViSIC [78]	*Cache-assisted*	✓			✓	
		Essaili('15) [39]	*Proxy-assisted*			✓		✓
		QoE-DASH [93]	*Proxy-assisted*		✓			
		Bouten('14) [23]	*Server-assisted*				✓	
		Juluri('15) [69]	*Cache-assisted*				✓	
		Kleinrouweler('17) [73]	*DANE*		✓		✓	
		Poliakov('16) [110]	*Cache-assisted*		✓		✓	
	ICN	DASH-INC [48]	*Cache-assisted*		✓		✓	✓
		Posch('14) [111]	*Cache-assisted*		✓			
		Jmal('17) [67]	*Cache-assisted*	✓				
		Our proposal	***Cache-assisted***		**✓**			**✓**
Multi elements Network assistance	**TCP/IP**	Aronco('17) [33]	*CDN*		✓			
		C3 [43]	*CDN*		✓			
		Liu('12) [85]	*CDN*		✓			
		VDN [96]	*CDN*				✓	
		Gugen('17) [51]	*CDN*		✓		✓	✓
		Nam('14) [98]	*SDN*				✓	
		QoE-FF [44]	*SDN*				✓	
		Petrangeli('15) [109]	*SDN*		✓		✓	✓
		Bentaleb('16) [19]	*SDN*		✓		✓	✓
		Cofano('16) [30]	*SDN*		✓		✓	
		Kleinrouweler('16) [72]	*SDN*		✓		✓	✓
		SABR [20]	*SDN*		✓		✓	
	ICN	Rainer('16) [115]	*Caches Network*				✓	
		DASC [79]	*Caches Network*				✓	

DASH adaptation logic

We consider two representative examples of network-blind DASH controller : BBA [59] and AdapTech [15]which are described in the next paragraphs.

Buffer-Based Algorithm (BBA). Introduced in [59], this simple and robust algorithm is agnostic to the network conditions, and its decisions are only driven by the buffer state. In a nutshell, BBA[2] defines two buffer thresholds, $B_{\min}$ and B_{upper} and uses the buffer level $B(t)$ at the client to take decisions. If $0 \leq B(t) \leq B_{\min}$, the quality selected is the lowest. If $B(t) \geqslant B_{\text{upper}}$, the quality selected is the highest one. When $B_{\min} \leq B(t) \leq B_{\text{upper}}$, a linear mapping is done from the buffer level to the selected quality.

AdapTech. A description of AdapTech can be found in Section 3.2. However, for the ease of the reader, the following paragraphs describe AdapTech again.

[15] introduces AdapTech, an adaptation logic that offers characteristics from both buffer-based and rate-based strategies. The main goal of AdapTech is to stabilize the buffer level around a target value, B_{steady}, while keeping the video quality as smooth as possible, by avoiding to react to short term bandwidth spikes, which would trigger unnecessary quality switches. The algorithm defines two thresholds, θ_1 and θ_2, which divide the buffer in three regions: the *panic* zone ($0 \leq B(t) \leq \theta_1$), the *buffering-state* zone ($\theta_1 \leq B(t) \leq \theta_2$) and the *cushion-state* zone ($\theta_2 \leq B(t) \leq B_{max}$). The algorithm keeps track of two different bandwidth estimates: the throughput of the last segment, A to which we will refer as the instantaneous throughput, and its smoothed version, $\hat{A}$, computed via an EWMA ($\hat{A(n)} = \alpha \times \hat{A(n-1)} + (1-\alpha) \times A(n)$), to which we will refer as the average throughput.

The selection of the quality of the next segment is done following one of the three modes described below. The used mode depends on which zone the buffer level is and the current quality (q_k):

- **Panic mode:** when the buffer level is in the panic zone, the lowest video quality is selected;
- **Buffering-state mode:** when the buffer level is in the buffering-state zone, the video quality is selected using the instantaneous throughput: if the instantaneous throughput is higher than the video bitrate of the next quality ($A > b_{k+1}$), then the next quality, q_{k+1} is selected. If the instantaneous throughput is lower than the current video bitrate ($A \leq b_k$), then the qual-

[2]We use the BBA-0 algorithm, referred to as BBA in the remainder of this chapter

ity is decreased and q_{k-1} is selected (assuming that the current quality is not the lowest one);

- **Cushion-state mode:** when the buffer level is in the cushion-state zone, the video quality is never decreased, and can be increased if over the last T seconds, the average throughput is higher than the video bitrate of the next quality ($\hat{A} > b_{k+1}$) and the instantaneous throughput is also higher than the video bitrate of the next quality.

The panic mode aims at quickly building up the buffer to avoid rebuffering events. The buffering-state mode aims also at increasing the buffer, but as the panic mode partially filled up the buffer, it is not critical to have the video segments as quickly as possible, therefore the quality selection can be done following the instantaneous throughput variations. Therefore, the quality selection rapidly adapts to network conditions by quickly switching to sustainable qualities. Finally, in the cushion-state mode, on the one hand, the video quality can not be decreased to avoid negative short-term fluctuations of the bandwidth, as the already built-up buffer can absorb the downloading time variations induced by a negative spike in the bandwidth. On the other hand, the T parameter is introduced to prevent positive short-term fluctuation of the bandwidth to trigger unwanted quality switch.

Network: Cache policies

We consider three scenarios, and devise two modes of interaction between network and application.

Baseline. No caches are considered in the network. On the client side, network-blind BBA and AdapTech logics are used.

Proactive placement, no cache replacement. A proactive placement is performed at the cache: one quality, arbitrarily selected, is fully cached at the router (i.e., all the data packets of all the segments of the selected quality are cached), and there is no cache replacement. Moreover, the router advertises the quality that is in cache to the clients via annotations in the MPD file. *Network-blind* clients resort to their unmodified BBA and AdapTech adaptation logics to select the quality of the next segment. Conversely, *network-aware* clients can interact with two modes: ① a *strict* one (referred to as the *Strict policy*), that forces the client to download segments at the quality advertised by the network, regardless of its state (buffer level and rate estimation); ② a *steered* one (referred to as the *Steered policy*), that follows the router indication unless its buffer level is below a given

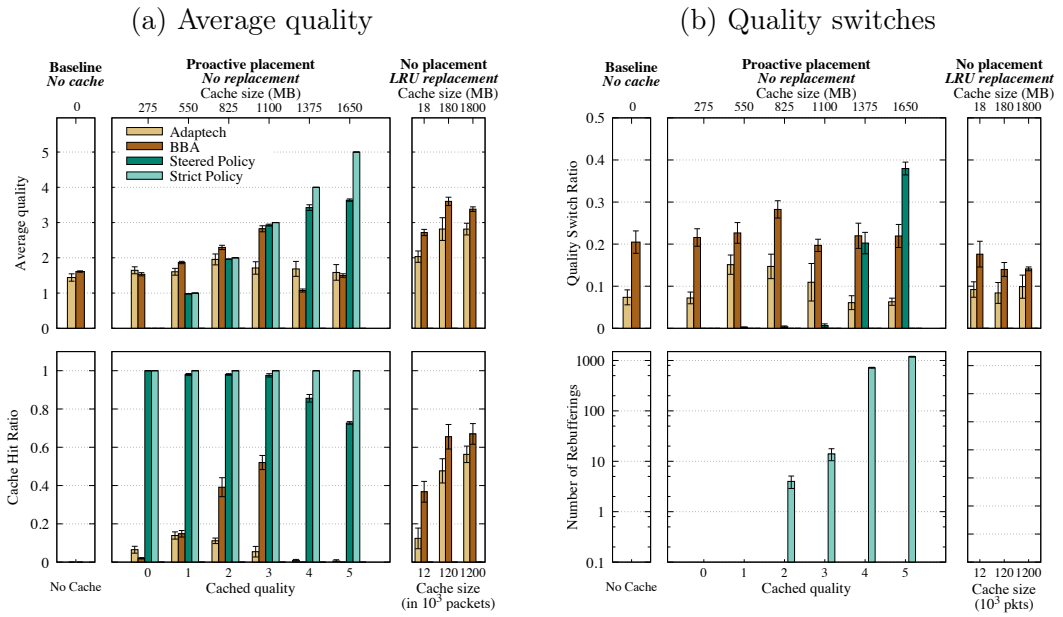


Figure 4.1: Design space at a glance: plots (a)-(d) show a key performance metric, with different subplots for different scenarios, i.e., baseline (left), proactive placement without replacement (middle) and no placement with LRU replacement (right). The bottom x-axis reports the quality cached at the router (in the proactive placement scenario) or the cache size in number of data packets (in the LRU replacement scenario), while the top x-axis reports the equivalent cache size in MB (both scenarios).

threshold ($B_{min} = 20\%$ in our experiments), in which case the client downloads at the lowest quality $q = 0$ (i.e., panic mode as in AdaptTech).

No proactive placement, LRU replacement. No proactive placement is performed and the cache uses a LRU replacement as cache management policy. On the client side, network-blind BBA and AdapTech logics are used.

4.3.2 Results at a glance

Scenarios

For the sake of simplicity, we consider six emulated clients connected (via Wi-Fi 802.11n or Ethernet) to an intermediate router (possibly equipped with a transparent cache of controlled size S) connected to a video server via an Ethernet

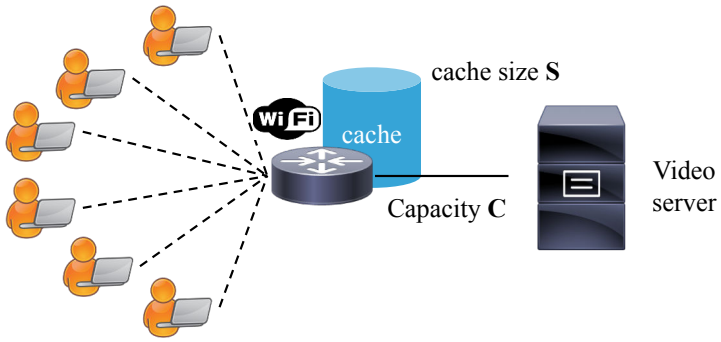


Figure 4.2: The topology used in our experiments: 6 clients are connected to an intermediate router equipped with a cache and connected to a video server.

link (of controlled capacity C). The topology is described in Figure 4.2. All the nodes (i.e., clients, router, servers) are ICN-enabled (using the ICN stack of the Linux Foundation CICN project [134]), and each client runs an instance of Viper (the default dual-stack TCP/IP and ICN video player of CICN, described in more details in appendix B). We set the buffer capacity to 20 video segments, and uses the default parameters of the adaptation algorithms indicated earlier. The server hosts the Tears of Steel video encoded at 6 different qualities, identified by their bitrates (i.e., 3, 6, 9, 12, 15 and 18 Mbps). More information on the tools used to deploy our testbed can be found in Appendix A, and the instructions to reproduce our results can be found in Appendix D.

Without loss of generality, we report here a case where we set the capacity $C = 60$ Mbps, and connect the clients to the intermediate router using a WiFi 802.11n link: clients are close to the access point, so that the WiFi channel capacity is about 100 Mbps. In order to prevent PIT aggregation from occurring at startup in our experiments (which would lead to a multicast tree to naturally form in ICN), we introduce stochastic arrivals (with average inter-arrival of two seconds). For each scenario, each player downloads once the video and we gather 95% confidence interval over 10 runs.

Experimental results

We contrast in Figure 4.1 the wide boundaries of the design space with the usual metrics: the average quality perceived by all clients (Figure 4.1-(a)), the number of rebuffering events (Figure 4.1-(d)), the ratio of quality switches (the number

of quality switches divided by the number of segments downloaded, Figure 4.1-(b)) and the cache hit ratio (whenever relevant, Figure 4.1-(c)). Each subfigure is further divided in three plots, one for each scenario: baseline (left), proactive placement (middle) and reactive LRU cache (right). Whenever relevant, the cache size is reported in bytes (top x-axis) and the placed quality or the equivalent number of data packets are also indicated (bottom x-axis).

Baseline. In the *baseline* scenario, when no cache is available and without PIT aggregation, the bottleneck for each client is the router-server link: the fair-share is about 10 Mbps, so that the highest viable quality is $q = 2$ (bitrate $b_2 = 9$ Mbps). However, due to the segment size fluctuation, the average quality is around 1.5 for both BBA and AdapTech.

Proactive placement. For *Network-blind* clients, placement may be inefficient when there is a mismatch between the available resources and the cached qualities: these will be unlikely to be requested by the clients, either because too high qualities are cached and there is not enough bandwidth to request them (e.g., $b_4 = 15$ and $b_5 = 18$ Mbps exceed the fair-share of the WiFi access to the cache), or because the cached quality is too low in terms of bitrate compared to the bottleneck capacity (e.g., $b_0 = 3$ and $b_1 = 6$ Mbps are lower than the fair-share of each client on the bottleneck link between the router and the original server). As such, placing either the lowest or the highest qualities (qualities 0,1,4 and 5) does not result in measurable gain compared to the baseline scenario, while the cache hit ratio is low (up to 15% for quality 1, close to 0% for the other qualities). Rather, wrong placement can even worsen the user QoE: placing $q = 1$ induces quality oscillations (the quality switch ratio for AdapTech nearly doubles), while the other metrics are not impacted.

Network-aware strategies, that are informed of the cached qualities, do not necessarily benefit from proactive placement either: if the cached quality is too low (qualities 0 and 1), the average quality of all clients is below the baseline scenario. Conversely, when the cached quality is too high (qualities 4 and 5), we see an improvement of the average quality, but at the costs of either rebufferings (strict policy), or quality switches (steered policy). A well dimensioned proactive placement (e.g., which can be the result of an optimization problem) exacerbates the tradeoff between average quality and quality switch for both *network-blind* and *network-aware* clients. For instance, placing quality $q = 2$ or 3 in our scenario induces a cache hit increase for BBA (respectively 40% and 50%), along with a slight increase of both the average quality and of the quality switch ratio[3]. Even

[3]This is due to BBA's linear mapping from the buffer level to the quality requested: when quality $q = 2$ is cached, it will be quickly retrieved from the cache, thus the client will eventually

with well dimensioned proactive placement, AdapTech still suffers from cache-induced quality oscillations: this can be inferred by a low cache hit ratio, a higher quality switch ratio and no significant improvement of the average quality. It is worth noting that network-aware policies with proactive placement yield better results in terms of cache hit ratio and better (than AdapTech) or comparable (to BBA) results for the average quality, but do not eradicate oscillations.

LRU caches. Finally, when LRU replacement is performed at the cache, we observe an improvement compared to the other scenarios: the average quality and cache hit ratio are higher, while the quality switch ratio remains more or less the same. It is worth noting that the size of the cache has some influence on the performance of the clients: when the cache has a small size (18 MB), the average quality is less than the one observed with a bigger cache (180 MB or 1800 MB).

4.4 Network-aware DASH proposal

From Section 4.3, we learn that caching increases average quality but possibly induces quality oscillations. Proactive placement is cumbersome (as it should carefully take into account available and time-varying resources) whereas LRU caches are simpler and thus appealing. At the same time, we observe that advertising the cached quality to guide the client choice can improve the overall QoE: while advertisement is natural in the proactive placement case, benefits should arise also under LRU caches. Indeed, LRU caches are driven by the controller requests, which would thus benefit from informed assistance from the network to increase cache efficiency and reduce oscillations: the simplest possible signal that an LRU cache can track (at low overhead) and export (through SAND, at low rate) is the *average per-quality cache-hit-ratio* (to which we will refer to as the *average per-quality hit-ratio*). We now show how network-aware DASH clients can turn this simple indication to a useful knob to refine their decision process by simply performing throughput estimations on a per-path basis.

4.4.1 NA^2: Network-Aware AdapTech

We enable clients to differentiate the source of each ICN Data packet by using a path label. This allows to dicriminate server vs cache traffic, so that clients can keep track of throughput estimations on a per-path basis: one estimation for the throughput toward the cache and one toward the server. Furthermore, we enable the cache to periodically advertise to the clients a per-quality pair of signals: the

request quality $q = 3$, which will take longer to download, depleting the buffer and causing the player to request quality $q = 2$, and so on.

average hit-ratio and number of samples (ICN Interest packets received). This advertisement can be done using SAND, but it can also be achieved by updating the MPD on the fly at the cache in cases where the MPD is periodically updated (e.g., MPD live). By combining these informations, the client can make an educated choice on the quality of the next segment to download. Algorithm 1 describes Network-Aware AdapTech (NA^2), a modified version of AdapTech taking into account the in-network assistance provided by both path-labelling and cache advertisements.

Like AdapTech, NA^2 divides the buffer in three zones: the ① *panic* zone, the ② *growing* zone and the ③ *steady* zone, delimited by two thresholds: B_{panic} and B_{steady}. In the ① panic zone, the lowest quality is selected in order to quickly fill the buffer as to avoid rebuffering events that are harmful to the user QoE. In the ② growing and ③ steady zones, selection is a two-step process: first we compute the *feasibility* of each considered quality and second, we select the highest feasible quality. A quality q is *feasible* if the downloading rate BW is higher than the associated bitrate b_q, i.e., the segment is downloaded faster than viewed. Specifically, the rate is multiplied by a conservative slack factor δ to account for size variations across segments, and the instantaneous BW or average $\widehat{BW}$ rates are used depending on the buffer state.

Network-awareness kicks in zones ② and ③. If there are not enough samples for this quality ($N_q < T_{samples}$), the informations provided by the cache are not significant and therefore a conservative choice is made, by using the estimated throughput to the server to compute the quality's feasibility. If there are enough samples, the average per-quality cache-hit ratio P_q is segmented in three zones: ④ cold ($P_q \leq P_{Low}$), ⑤ warm ($P_{Low} < P_q \leq P_{High}$) and ⑥ hot ($P_q > P_{High}$) cache. In the ④ cold zone, it is likely that segments of quality q are not cached, and will be downloaded from the server, therefore the estimated throughput to the *server* is used to *conservatively* compute the quality's feasibility. In the ⑥ hot zone, it is likely that the quality is cached and thus the estimated throughput to the *cache* is used. Finally, in the ⑤ warm zone both estimates are used: in the ② growing buffer state, the main objective is to fill the buffer, therefore a conservative choice is made (i.e., the quality has to be feasible for both paths), while in the ③ steady buffer state, the buffer level is high enough to allow for a more optimistic choice (i.e., the quality has to be feasible for at least one of the two paths).

As in AdapTech, we restrict the magnitude of a quality switch to one, and therefore, in the ② growing and ③ zones, we consider only the current quality and the ones directly below and above it (when possible). Note that, just like AdapTech, in the steady zone, we can only increase the quality if for at least T seconds, the network conditions allow us to switch to a higher quality: at that point, the can-switch-up (CSU) flag is set to $True$.

Algorithm 1 NA^2: Network-Assisted AdapTech

Params : $T_{Samples}$ // # of samples threshold
$P_{Low,High}$ // Cache thresholds
δ // slack factor

Variable: $B(t)$ // Buffer level
BW_S, BW_C // Server and cache instant throughput
$\widehat{BW}_S$, $\widehat{BW}_C$ // Server and cache average throughput
$\{P_i\}_{1\leq i\leq M}$, $\{N_i\}_{1\leq i\leq M}$ // Per quality cache hit and samples #
q, b_q // Current quality and associated bitrate
CSU // Can-Switch-Up flag

Function `BitrateSelection()`
  if $B(t) \leq B_{panic}$ **then** ▷ ① Panic
    $q \leftarrow 1$
  else if $B(t) \leq B_{steady}$ **then** ▷ ② Growing
    $q \leftarrow \underset{i\in[\![q-1,q+1]\!]}{\text{ArgMax}}$ `(IsFeasible(`$i,\ BW_S,\ BW_C$`))`
  else ▷ ③ Steady
    if `IsFeasible(`$q{+}1, \widehat{BW}_S, \widehat{BW}_C$`)` && CSU **then**
      $q \leftarrow q+1$
  return q

Function `IsFeasible(`$q,\ BW_S,\ BW_C$`)`
  if $N_q \leq T_{Samples}$ || $(P_q \leq P_{Low})$ **then** ▷ ④ Cold
    return $(BW_S \times \delta > b_q)$
  else
    if $P_q \leq P_{High}$ **then** ▷ ⑤ Warm
      return $(BW_S \times \delta > b_q)$ && $(BW_C \times \delta > b_q)$
    else ▷ ⑥ Hot
      return $BW_C \times \delta > b_q$

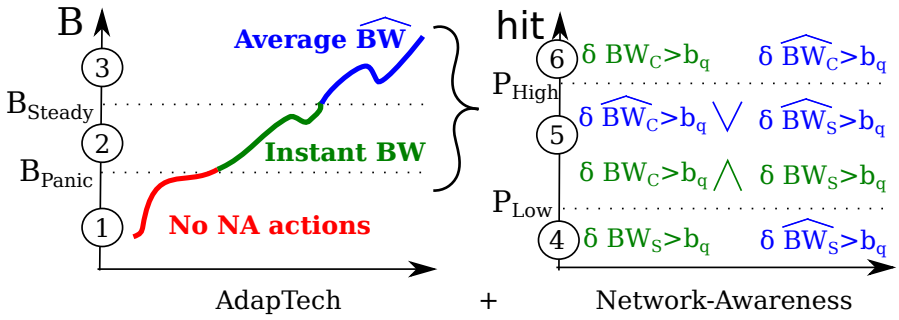


Figure 4.3: Synoptic of AdapTech (left) + Network-Aware (NA, right) decisions

Finally, Figure 4.3 presents a graphic summary of our NA^2 algorithm. The right plot presents the buffer level divided in the three zones and the left plot presents the Network-aware decisions.

4.5 Evaluation

4.5.1 Sensitivity analysis

To assess the impact of the different parameters of our algorithm on the user QoE, we use the early described topology, in which all six clients are connected to the router using an Ethernet link with a capacity of 30 Mbps. The router is connected to the server via an Ethernet link of capacity C = 30 Mbps. To avoid PIT aggregation, we introduce stochastic arrivals (with average inter-arrival of 6 seconds). We set the cache capacity of the router to 1.8 GB (corresponding to 1.2 M Data packets). We use default AdapTech values for $B_{panic} = 10s$, $B_{steady} = 20s$, $\delta = 0.8$. For Network-awareness, we advertise the cache hit-ratio every 30 seconds, and require to have collected at least $T_{Samples} = 10^4$ packets. Assuming that the available bandwidth to the server is lower than the one to the cache, the higher P_{Low} is, the more conservative our algorithm is. We thus set $P_{Low} = 0.1$ to avoid being too conservative and we vary P_{High} from 0.1 to 0.9. For each value, we repeat experiments 40 times, for a total of over 100 hours worth of experiments.

Figure 4.4 presents the ratio of the usual metrics of our algorithm vs baseline AdapTech for the 10-th worst percentile of clients (left) and the median client (right). We see that our algorithm significantly reduces the number of *quality switches* (by about a factor of 2 in both cases) and drastically cut the *rebuffering rate* (by about one order of magnitude for both cases). In the case of the median client, this is done as expected at the expense of the average quality, which reduces by about 15%. Interestingly, for the worst 10-percentile of clients, the av-

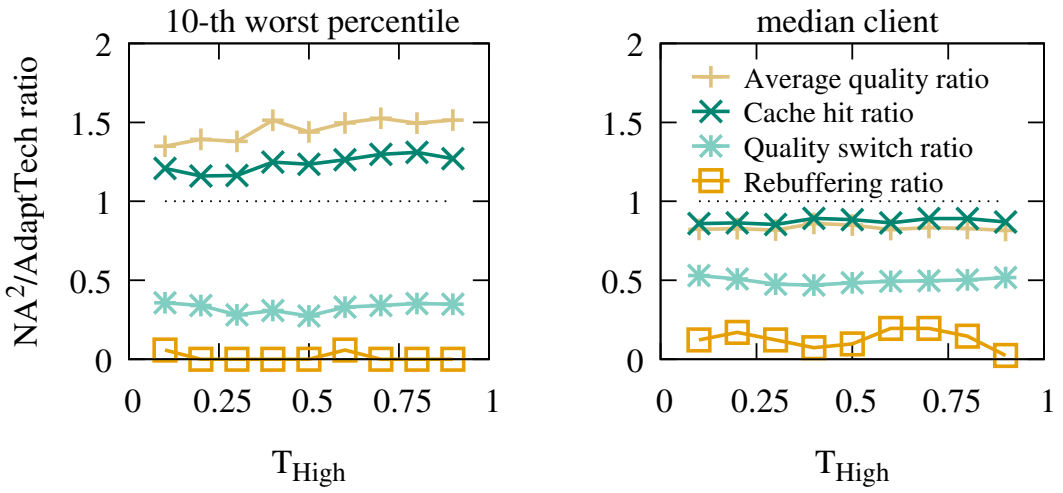


Figure 4.4: Sensitivity analysis: ratio of Network Aware vs Network blind performance for the 10-th worst percentile of clients (left) and for the median client (right).

erage quality actually increases by 50%, which is again a sizeable improvement. Finally, note that results are very stable irrespectively of the exact (P_{Low}, P_{High}) parameterization: we can observe the upper bound of average quality is obtained at (0.1, 0.35) and the lower bound of quality switches at (0.1, 0.5).

4.5.2 Comparison with Network-blind baseline

In this section, we vary the cache capacity of the router between 90MB and 1.8 GB (respectively corresponding to 60k and 1.2M packets). For each cache size, we run 20 experiments for each adaptation logic: network-blind AdapTech, NA^2 (upper bound), and NA^2 (lower bound), for yielding a total of 120 experiments. The results are presented in Figure 4.5, in terms of average quality, quality switch ratio, cache hit ratio and rebuffering probability (the ratio of the number of rebufferings over all experiments over the number of segments downloaded). The gold bars present the results for network-blind AdapTech, the brown ones present the results for NA^2 (upper bound) and the green ones presents the results for NA^2 (lower bound).

For both cache sizes, we confirm that NA^2 sizeably (drastically) reduces the quality-switch (rebuffering) ratio. Particularly, when the cache is small (60k packets), NA^2 outperforms a network blind AdapTech, both in average quality and in quality switches. This is due to a better utilisation of the cache (notice the hit rate increase): with network-blind AdapTech, quality oscillations happen which pollute the cache. When the cache is small, this pollution is critical because it replaces

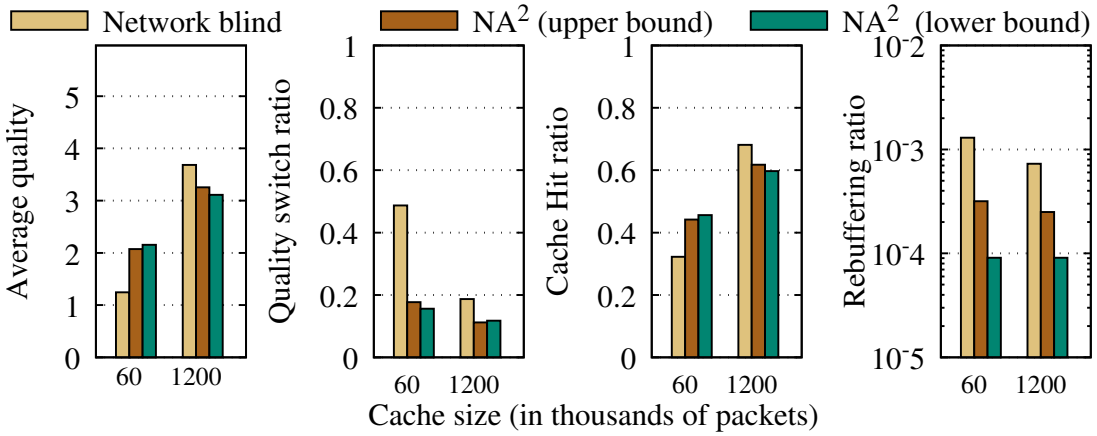


Figure 4.5: Comparison of Network blind vs Network Aware AdapTech, for two NA^2 settings: upper bound of average quality $(P_{Low}, P_{High}) = (0.1, 0.35)$ and lower bound of quality switch ratio $(P_{Low}, P_{High}) = (0.1, 0.5)$.

segments that can be useful for the other clients. Our network-aware approach circumvents this pollution by giving more informations to the client, which can make educated choices for the quality of the next segment and thus preventing quality oscillations.

With a bigger cache (1.2M packets), the pollution is still present, but is less critical because it does not replace useful segments. As a result, the average quality is higher, for both network-aware and network-blind algorithms. The average quality observed for our network-aware approach is slightly lower than compared to network blind AdapTech, which is necessary to prevent cache-induced quality oscillations. Shortly, NA^2 is simple and robust, providing sizeable benefits for DASH QoE.

Finally, we compare a network blind approach and NA^2 (both upper and lower bound) under different access networks. To do so, we use the same topology, along with 3G[4] and 4G[5] traces for the access network between the clients and the router. For both traces, we modified them to get an average bitrate of 18 Mbps, and each client starts at a random point in the trace. Furthermore, we set the capacity of the link between the access router and the server to $C = 25$ Mbps and we reproduce the same experiments as described earlier. The results are gathered in Figure 4.6: Figure 4.6-(a) reports results for the 3G traces and Figure 4.6-(b) the 4G traces.

We observe here that the size of the cache has no impact on the average quality of the clients. This can be explained by the fluctuations in the traces along with the

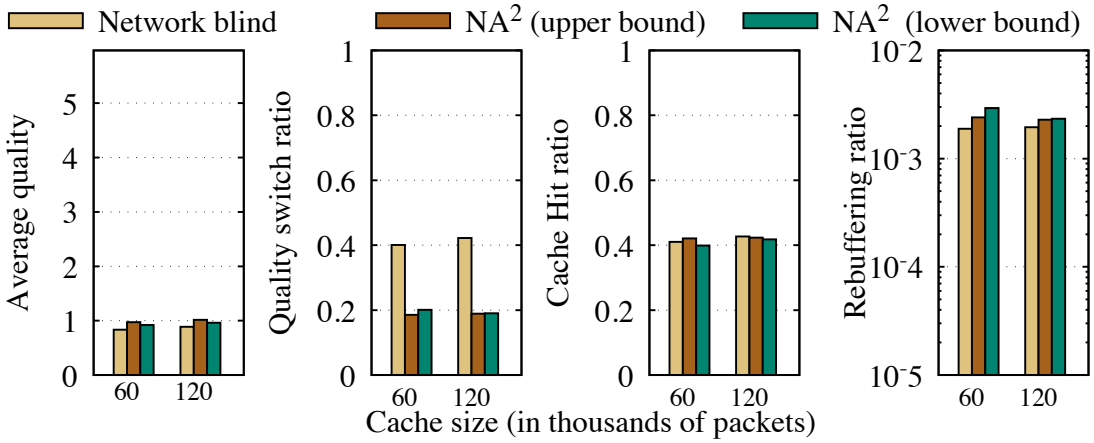
[4] `http://home.ifi.uio.no/paalh/dataset/hsdpa-tcp-logs/tram.jernbanetorget-ljabru/report.2010-12-16_1100_CEET.log`

[5] `http://users.ugent.be/~jvdrhoof/dataset-4g/logs/report_foot_0001.log`

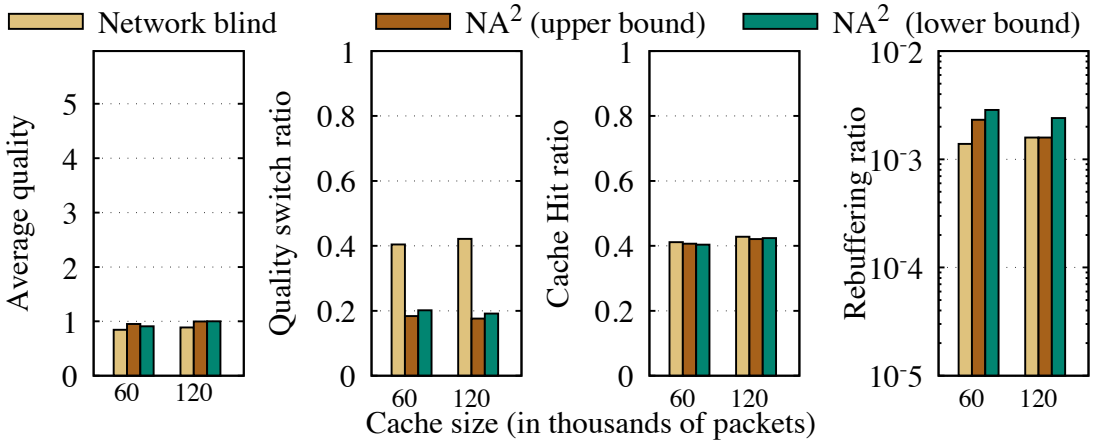
fact that users start at a random point in the traces making the cache less useful: note how the cache-hit ratio remains the same (around 40%), independent of the cache size. However, we also observe that NA^2 sizeably reduces the quality-switch ratio, successfully preventing the cache-induced quality oscillations.

4.6 Conclusion

In this chapter, we explored how in-network caching, coupled with network assistance, impacts dynamic adaptive streaming. Specifically, we explored the design



(a) 3G traces.



(b) 4G traces.

Figure 4.6: Comparison of Network blind vs Network Aware AdapTech , for the upper and lower bound of average quality, under various access networks: 3G and 4G traces.

space (Section 4.3) of in-network caching and showed that while in-network caching can improve the average video quality, it can induce quality oscillations. We then proposed a simple signal such as the per-quality cache hit-rates from cache, coupled with a per-path throughput estimation, which is effective for (i) avoiding the cache-induced oscillations (reduction of the quality switch ratio), while (ii) maintaining a comparable average quality (increasing worst case quality but necessarily reducing quality for more aggressive clients), and (iii) increasing the cache hit ratio.

While the quantitative results showed in this chapter are gathered with a specific network-aware evolution of AdapTech, we argue that network-assistance such as the one we propose is beneficial to all rated-based adaptation logics: part of our future work aims at systematically leveraging such signals in multiple controllers [71].

Additionally, we argue that the class of signals exported by the network could be extended. Particularly, another appealing signal is binary feedback piggybacked from the cache to assert whether the *next* segment in the same quality is cached. On one hand, such feedback would require additional overhead at the cache (extra lookup for content that has not been requested yet) and would also need a refined timing: if the feedback is too early, the segment could be evicted from the cache, and if the feedback is too late, it will reach the client after his decision. On the other hand, such feedback would help further to increase the user QoE.

Finally, such network signal can be helpful in the case of live video, especially in the case of dynamic adaptive streaming over ICN, where PIT aggregation and in-network caching can significantly reduce the traffic load upstream. The next chapter will cover how ICN can be leveraged in real-time communication to improve existing architectures.

Chapter 5

Assessing ICN benefits in Real-Time Streaming

Contents

5.1 Introduction

5.1.1 DAS real-time communication

As stated earlier in this thesis, DASH presents some advantages compared to live streaming protocols (such as RTP, RTSP, RTMP, etc): for example, it does not suffer from UDP filtering and it can do NAT traversal. However, DAS is not well suited for ultra low-latency media streaming: some delays are induced with DAS, as explained in [87]. It identifies the sources of delay as:

1. *Content acquisition.* It represents the delay induced by recording the media (video and/or audio). This delay is not specific to DAS.

2. *Server-side packetization of media segments.* It represents the delay to have a new media segment available. Most of the time, this is equal to the segment duration as the server waits until it has received the whole media content for this segment before making it available.

3. *Asynchronous fetch of media segment.* As the server does not signal to the client that a new media segment is available, the client must poll for data, resulting in some additional delay.

4. *Time to download the segment.* The delay induced by the network, it depends on the size of the media segment and the network characteristics.

5. *Buffering at the client-side.* To ensure a smooth playback, the client buffers one or more media segments to mitigate transport jitter, such as varying download time.

6. *Decoding at the client.* The delay induced by the decoding of the media segment at the client. This delay is not specific to DAS.

Therefore, the most straightforward way to reduce the latency for DAS is to lower the segment duration. However, as shown in [87] and [139], this creates an overhead: shorter video segments go hand in hand with a larger number of segments to represent the same video and thus the HTTP encapsulation overhead increases [87]. Moreover, shorter video segments lead to an explosion in the number of HTTP requests [139]. To circumvent this increase of HTTP requests, [139] proposes to leverage HTTP 2.0 server push feature: the client can request in advance video segments, which will be pushed to it as soon as one requested segment becomes available at the server, hence reducing the delay by decreasing the segment duration without increasing the number of HTTP requests from the client.

[24] exploits low latency video coding techniques (Gradual Decoding Refresh) and HTTP 1.1 chunked-transfer encoding to reduce the segmentation delay. More

precisely, by fragmenting media segments into chunks down to the video frame level, the authors achieve a end-to-end latency of 240 ms. In [57], the authors rely on application-layer multi-path along with frame byte-range signalling to the client, in order to retrieve faster the video frames composing a segment and thus reducing (i) the packetization delay since the segment is downloaded while it is being constructed at the server side and (ii) the network induced delay by using multiple paths to spread the downloading load.

However, while some issues are mitigated by these works, some issues are still open. For instance, if short segments are used, each segment should start with an I-frame, to ensure a smooth transition from one video quality to another, and as I-frames are bigger (in bytes) than P or B-frames, reducing the duration of media segments could lead to an increase of the size of the media segments. Furthermore, DAS adaptation logics relies on having the same segment encoded at different qualities. For low latency purposes, this would require to have all qualities encoded at the same time, and thus having multiple encoders running, which is CPU intensive for the server. Furthermore, if we assume an intra-segment downloading mechanism (as described in [57, 24]) with very low latency, if the network conditions change rapidly, the client may want to switch the video quality, which is only possible at the beginning of the downloading of a segment, limiting the responsiveness of the adaptation logic, which, in very low latency environments, can deeply impact the user's quality of experience.

5.1.2 WebRTC

Traditionally used in collaboration systems, WebRTC is increasingly adopted by media companies for live eventing, gaming, betting or auctioning services where real-time latency is required for both media distribution and live interaction and/or feedback from participants. One of the reasons for its recent success is the capability to support low-latency targets better than HTTP live streaming technologies such as HLS or MPEG/DASH, as shown in the previous section. By using WebRTC the latency can be reduced from tens of seconds to few hundred milliseconds as required by realtime communication applications.

However, unlike modern CDNs, WebRTC distribution model is not designed for large scale and thus, scalability appears to be a key metric to validate WebRTC as a credible candidate to support low latency streaming services. Taking a single video stream and mirroring/multiplying it to thousands of users with the necessary bandwidth today is complex and costly to manage via WebRTC, due to the point-to-point nature of peer-to-peer or peer-to-central node communication model, as well as to the centralised signalling infrastructure. To improve scalability, the architecture of traditional multiparty conferencing tools has moved from peer-to-peer to centralized: the Multi-point Control Unit (MCU)-based architecture was first

introduced and, more recently, there was a trend to move toward the lighter and possibly decentralised Selective Forwarding Unit (SFU)-based architecture. We review all the WebRTC architectures in Section 5.4.

Using an SFU, most of the complexity of the central node is offloaded to the conference participants or distributed to multiple control nodes. To avoid costly encoding/decoding operations at the SFU (also referred to as media bridge), the use of Simulcast [50] is becoming increasingly popular.

In the quest of a solution that combines scalability and efficiency benefits, we explore in this chapter the potential benefits coming from the use of a different underlying transport based on ICN principles. Native mobility and multicast are among the major benefits that ICN would bring, where low-latency would be achieved by dynamic hop-by hop forwarding of packets based on network conditions and content awareness. ICN appears suitable for the support of RTC applications, as partially confirmed by initial work within the ICN community [29, 65, 144].

In this chapter, we build on the SFU-based WebRTC architecture and investigate the additional scalability benefits that may result from removing the point-to-point transport limitations by using an underlying content-based network transport. To do so, we leverage the Hybrid Information-Centric Networking (*hICN*) architecture, an ICN-in-IP solution incrementally deployable that requires minimal modifications to the current network and applications.

The contribution here is twofold: (i) we design and implement *hICN-RTC*, an integrated WebRTC over hICN system, and (ii) we propose a new Realtime Information Centric Transport Protocol, RICTP, a content-aware transport that minimizes the communication latency. We assess their performance against standard WebRTC. The results are encouraging: hICN-RTC shows lighter load on the SFU due to its integrated content-aware transport. More interestingly, hICN-RTC scales with the number of active speakers rather than the total number of connected users, overcoming existing known limitations of RTC platforms in terms of maximum number of participants.

The rest of the chapter is organized as follows: Section 5.2 presents related work in the ICN domain, Section 5.3 describes hICN; Section 5.4 reviews the existing WebRTC architectures; we introduce hICN-RTC in Section 5.5 and the Realtime Information Centric Transport Protocol (RICTP) in Section 5.6; our proposals are evaluated in Section 5.7.

The work presented in this chapter was submitted here [103].

5.2 Related work

There have been some efforts in the ICN community to address real-time communication over ICN. VoCCN [61] and ACT [146] show the feasibility of real-time

communications using ICN considering respectively two-party and multi-party audio conferencing, while pointing out the benefits in terms of scalability, robustness and security over similar IP based solutions. ACT also highlights specific methods to achieve conference discovery and speaker discovery in order to generalize a distributed conferencing framework.

While they are both limited to audio conferencing, NDN-RTC [52] addresses directly WebRTC by making the underlying networking stack NDN-based. NDN-RTC addresses the additional complexities for generating, publishing, and consuming video content, and provides novel approaches to minimize latency using a pull based communication framework. However, the application-support features offered by NDN-RTC continue to rely on WebRTC host-based approaches, such as measuring response time for Interests during bootstrap phase or during connection disruptions for the wireless consumer, without exploiting the content awareness in the network to improve efficiency and reduce latency of RTC.

Other work replaced the WebRTC-based approach with a full ICN architecture trying to leverage monitoring at client-level [29, 65] and at network-level [144] to decide upon rate adaptation. In [29] and [65], the authors design a Serverless Scalable Audio-Video Conferencing, leveraging a push primitive rather than the pull-only transport model of CCN/NDN to minimize latency without sacrificing multicast, flow balance, caching and multipath routing features. They show good scalability properties of the ICN approach to any number of conferences and video flows, along with a good flexibility in adapting to random join/leave of participants. They also highlight the benefits of a distributed approach to avoid single point of bottlenecks in client-to-server and P2P exchanges. Finally, security considerations are taken into account: communications are authenticated and private, using the ICN content-based security model as opposed to the TLS/DTLS channel-based approach. However, the proposed approach still remains host-only driven. In [65], the proposed architecture, SRMCA (Scalable Real-time Multiparty Communication Architecture) attempts to reduce the end-user complexity by offloading some of the conference framework functionality to the network. Critical operations like namespace synchronization are provided by the network as a service, resulting in a more deterministic response to join/leave events and network disruptions, as well as opening up the case of a newer business model. The resulting simplified end-user component allows higher scalability in terms of the number of participants as corroborated by their experimental analysis. However, this comes hand-to-hand with the loss of the pure distributed nature of the application as certain functionalities are concentrated at network service points, and the incompatibility of such an approach with WebRTC primitives.

5.3 Hybrid ICN Background

Hybrid Information-Centric Networking (hICN) [97] extends the Internet Protocol (IP) to allow named-data communications in IP, both v4 and v6, by overloading the semantics of a few header fields in the IP packets as described in Section 5.3.1. In particular, hICN (i) does not loose any of the ICN features, (ii) transparently interconnects hICN routers with IP ones, which are able to process hICN packets as standard packets, and (iii) can be implemented by reusing most of the existing software, minimizing the modifications required to the existing network and applications.

As an implementation of the ICN architecture, hICN inherits the ICN pull-based request/reply protocol semantics: an *Interest* packet is sent to retrieve a *Data* packet. The forwarding pipeline implemented by the hICN forwarders is introduced in Section 5.3.2.

The hICN architecture comprises a network layer (L3) and a transport layer (L4) and provides service access points to content producers and consumers to produce and consume named resources, respectively. This is done using the hICN socket API discussed in Section 5.3.3.

5.3.1 Naming

Resource names are encoded as 128 (or 32) bits and can be represented using the common hexadecimal or dotted decimal notation for IPv6 or IPv4 addresses respectively, e.g. FE80:1111:2020::2020 or 192.168.1.1. hICN envisages the creation of a new address family `AF_HICN` to encode resource names. We will refer to these names as network names or `name_prefixes`. The `name_prefixes` are stored in the IP packet header. In particular, in a data packet the `name_prefix` is stored in the IP source address, while in an Interest packet it is stored in the IP destination field. The modifications to the IP header (IPv6 in this example) packet are highlighted in red in Figure 5.1.

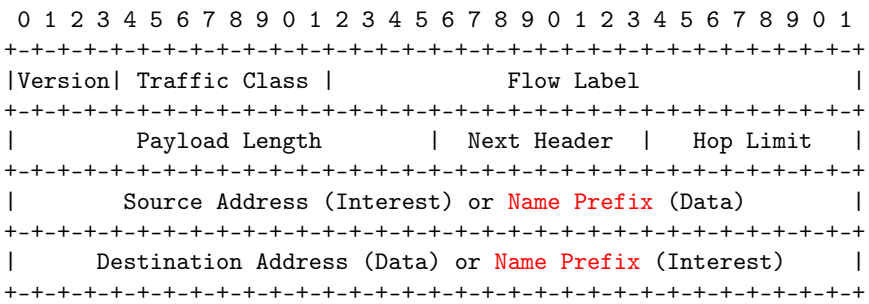


Figure 5.1: IPv6 interest and data packet description

The transport layer, by definition, takes care of segmentation and reassembly of upper layers protocol data units, with optional signing operations. The full data packet name is obtained by appending the segmentation information, namely the `name_suffix`, to the `name_prefix`. The `name_suffix`, along with other transport layer information, is carried in a TCP-like header, represented in Figure 5.2 (the modifications to the TCP header packet are also highlighted in red).

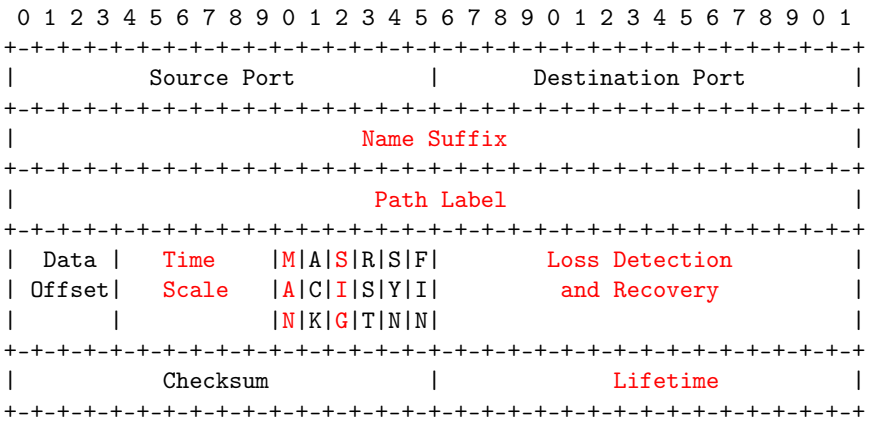


Figure 5.2: hICN transport header inside TCP-like header.

For the sake of clarity, in the remainder of this paper we use human readable names instead of IP addresses, where the `name_prefix` is made of a set of name components. An example of human readable `name_prefix` is "/video/participant/" where "/" is the separator between name components. The `name_suffix` is indicated as "seg=x".

5.3.2 Forwarding

Interest and Data packets follow a different forwarding path inside an hICN forwarder. From a high level standpoint, Interests are forwarded using routing by name while data packets are label switched, similarly to IP packets. The details of the forwarding mechanisms are reported in the Internet Draft [97] and we briefly discuss them here for clarity, distinguishing between Interest packet and Data packet forwarding paths. In each case we explain how the forwarding function is realized in an hICN-enabled router and how coexistence and transparent interconnection with regular IP routers is provided.

A typical ICN router manage three structures to forward packets: the Forwarding Information Base (FIB), the Pending Interest Table (PIT) and the Content Store (CS), the goal of the last two is to temporary store either pending Interest packets or Data packets. In hICN, these two structures are collapsed into a single

one, called *packet cache*. It is indexed by the full packet name (concatenation of prefix and suffix) and uses different insertion/eviction policies for Interest and Data packets, as discussed later in this section.

Packet classification and punting. Before explaining the forwarding procedures we need to explain how an hICN router can classify a packet in order to process it in the correct way. In particular the router needs to distinguish between Data packets, Interest packets and standard IP packets. hICN routers use the Access Control List functionality generally available on standard IP routers to classify the packets by using source (src) and destination (dst) address fields: (i) if only src belongs to `AF_HICN` the packet is a Data packet, (ii) if only dst belongs to `AF_HICN`, the packet is an Interest packet, (iii) if none of the two fields belong to `AF_HICN`, the packet is processed as a regular IP packet, (iv) the packet is dropped since it is an invalid packet. Once the packet is classified it is punted in into the right forwarding pipeline.

Interest forwarding path. When an hICN router receives an Interest packet, an exact match lookup on the full name is performed in the *packet cache*. If there is a `DataHit` (a Data packet corresponding to that name is in the *packet cache*), the router directly satisfies the Interest packet, without forwarding it upstream: the destination address of the matching Data packet is rewritten with the source address carried in the Interest packet and the Data packet is then forwarded on the incoming interface of the Interest packet. Note here that the source address of the Interest packet is the IP address of the previous hICN-hop traversed by the Interest packet. This address translation guarantees path symmetry at the hICN level, but the Data packets will not necessarily follow the reverse path of the Interest when traversing IP networks.

If there is an `InterestHit` (an Interest packet corresponding to that name is in the *packet cache*), if the source address of the received Interest packet is the same as the cached Interest packet, the incoming Interest packet is a duplicate, if the source addresses differ, it is a request coming from a new source. There already exist several solutions to handle these two cases in ICN and hICN adopts the one described in [95]. The latter case, corresponding to a *PIT aggregation* is of particular interest for this chapter. Specifically in that case, the incoming Interest packet is stored in the *packet cache* and dropped. This ensures that only one request for the same content is forwarded upstream, thus minimizing the traffic.

Finally, if there is `NoHit` (no packet corresponding for that name is stored in the *packet cache*), the Interest packet is passed to the IP FIB lookup stage to determine the set of available next hop options. This stage makes use of the existing IP FIB lookup engine without modifications, with the sole exception that hICN lookups return all the available output faces. The result is then passed to the forwarding strategy that decides on which face(s) the Interest packet will be

forwarded, according to some metrics that can be programmed. The Interest packet is also stored in the *packet cache* to allow for aggregation of future Interest packets.

Data packet forwarding. When a Data packet arrives at the hICN forwarder, an exact match lookup is done on the *packet cache* in order to find all the matching Interest packets (Interest packets with the same name that can be satisfied with the received Data packet). If there is no match, the Data packet is discarded. Otherwise, if there is an `InterestHit`, the Data packet is cloned to satisfy all the matching Interest packets: for each Interest packet that is cached, a copy of the Data packet with its destination address rewritten with the Interest packet source address is forwarded on the incoming interface of the Interest packet. The Interest packet is then evicted from the *packet cache*. In this way, hICN is able to multicast the content to many users. It is this feature that allows our proposed architecture to scale with the number of content streams rather than the number of users, since only one request per content is forwarded in the network. hICN-RTC is designed to maximize the advantages coming from interest aggregation and hICN multicast.

5.3.3 Consumer/Producer socket API

The consumer and the producer sockets are the core of the API that applications can leverage to implement location independent communications. hICN builds upon [124] that provides an API and implementation of two name-based socket types: *consumer* and *producer* socket. These sockets are uni-directional and are used to send data at the producer side, and to receive data at the consumer side.

The consumer socket pulls data sending interests, with no knowledge of the location of the one or multiple producers that can reply to the requests. For example, the network itself, meaning an hICN router, can reply to an interest sent by a consumer in case of a data hit in the packet cache, as described previously. The consumer socket also implements the transport protocol that is responsible to decide which and how many interests to send at any time. In this paper we present the Realtime Information Centric Transport Protocol (RICTP) in Section 5.6. Other kinds of protocols can be implemented, such as [25].

The producer socket responds to data requests coming from one or multiple consumers with no knowledge of their location. In Section 5.6 we describe also the role of the producer socket in RICTP. The producer socket also offers per-packet integrity and data origin authentication which are exploited by the consumer socket to verify the validity of the data, i.e., the data has been originated by a trusted producer and never modified. Data confidentiality is instead left to the application or higher layer protocols, e.g., SRTP.

5.4 Review of WebRTC architectures

In the last six years, WebRTC architecture has significantly evolved from the original peer-to-peer mesh. We quickly review here the main architectural options with pros and cons.

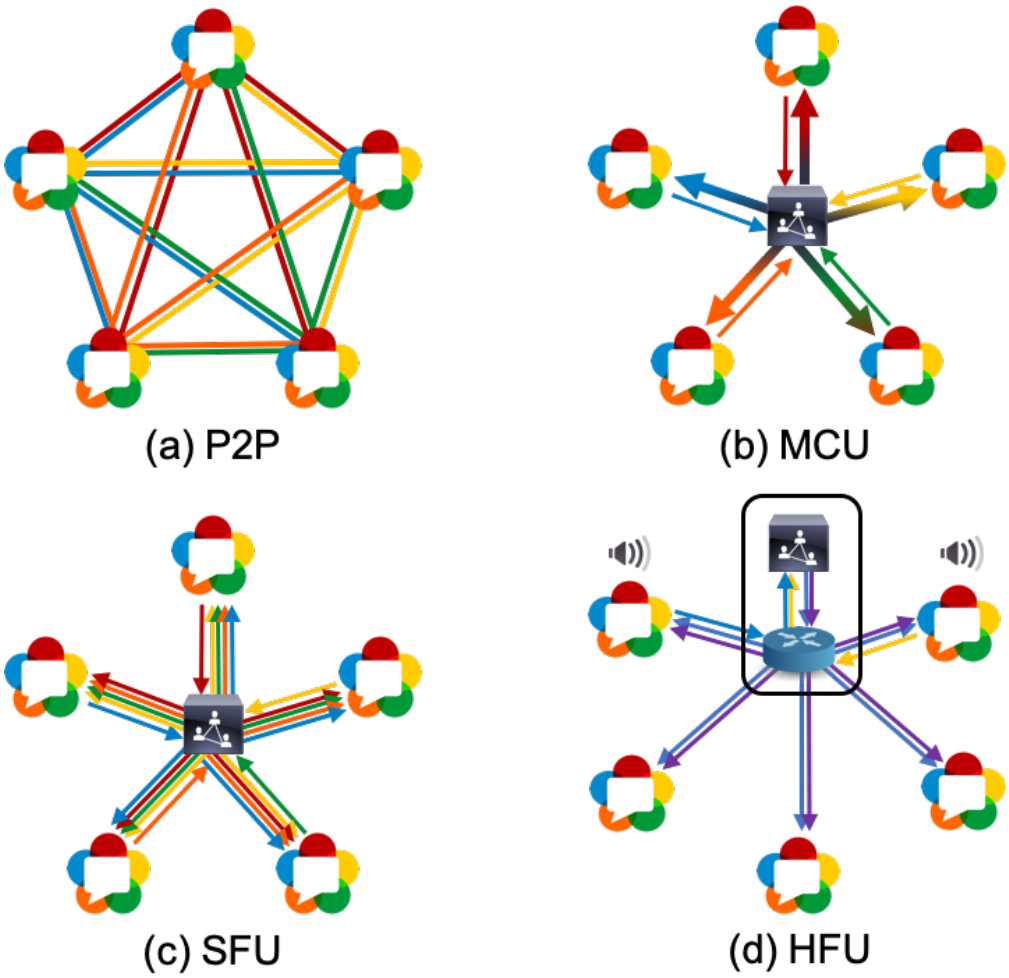


Figure 5.3: WebRTC (a,b,c) and hICN-RTC (d) architectures

Full mesh or P2P-based: in this architecture, depicted in Figure 5.3-(a), each participant sends directly its audio/video streams to all the other participants in a peer-to-peer fashion. On the one hand, this architecture has the advantage of minimizing latency by relying on direct point-to-point communications and, as such, does not require any intermediate node between the participants since they are all interconnected. On the other hand, it clearly does not scale well with the number of participants: assuming N participants, each one has to encode $N-1$

times its audio/video stream and to decode each one of the received $N-1$ streams. These are CPU-intensive tasks. In addition, this architecture requires significant uplink bandwidth at the participant's side, who needs to send its streams $N-1$ times, and this is not always realistic.

Multipoint Control Unit (MCU)-based: in this architecture, depicted in Figure 5.3-(b), each participant sends its audio/video streams to an intermediate node, called Multipoint Control Unit (MCU). This node gathers all media streams from the participants and, for each one of them, it encodes a composite stream that contains all the media. These new streams are personalized for each client, both in terms of content and bitrate. Thanks to this, the MCU is able to provide a really good user experience to the participants. Such architecture also allows to partially relieve some of the load on the participants, as each one encodes and sends only one stream to the MCU and receives and decodes only one stream. However, it creates new issues: the end-to-end delay increases since the MCU has to (i) decode the incoming video streams, (ii) re-scale the different videos for the composite flow, (iii) encode the composite video and (iv) send it to the participants. Furthermore, the decoding and re-encoding of the video streams can be very intensive in terms of CPU consumption, reducing the scalability of this approach.

Selective Forwarding Unit (SFU)-based: in this architecture, depicted in Figure 5.3-(c), each participant sends its audio/video streams to an intermediate node, called a Selective Forwarding Unit (SFU). This node then decides which streams to forward to the participants. This adds some flexibility as the SFU can decide to drop streams that are not important to the conference and forward only the important ones, such as the streams from active speakers. The selection of which streams to forward is performed using some selection algorithms, such as [49]. This is achieved without any media processing at the SFU node, thus not inducing extra delay nor extra CPU cycles. However, new challenges arise from such architecture: while in the previously presented architectures (P2P and MCU) a media stream is designed to be received by only one entity (participant or MCU), thus allowing a one-to-one video bitrate adaptation, with a SFU, a video stream can be received by more than one participant. Using the SFU, the video bitrate adaptation must be handled by the sender of the stream that needs to take into account the network conditions of all the receivers. To deal with this problem Simulcast has been introduced [50]: the producer generates and sends the same stream encoded with multiple bitrates to the SFU. The SFU decides which bitrate to send to which participant.

Recent evolution of MCU/SFU-based architectures has triggered work in many directions, notably distribution of control nodes (e.g. cascaded SFU [137], distribution of MCUs in the cloud [118], rate selection and adaptation at participant's side (e.g. Simulcast and related work on adaptation and congestion con-

trol [50], [106], [108]).

These architectures have different properties and may be differently selected, depending on the number of participants in a conference and of the resources available at participant/server side. However, the SFU-based architecture seems the most suitable for a scalability study. In the following section, we build upon such an architecture to discuss the integration of Hybrid ICN in the underlying network transport stack.

5.5 hICN-RTC Architecture

In this section we describe the architecture of our proposal. The main idea behind hICN-RTC is to build upon the standard SFU-based architecture and adapt it to hICN, by (i) integrating hICN both at clients and at central node, (ii) limiting modifications to application-layer semantics, (iii) designing a RTC-tailored hICN transport protocol.

In terms of proposed architecture, like in the SFU case, the central node forwards streams without any transcoding operation, with the difference that in hICN-RTC we exploit the multicast features of the underlying hICN network to replicate and distribute each flow, offloading the media bridge application. We call our central node Hybrid Forwarding Unit or HFU. This node is composed of two parts: the application, which remains similar to that of a classic SFU, and a hICN forwarder. The HFU is represented in Figure 5.3-(d). We describe hICN-RTC assuming a star topology, i.e., a set of participants connected to the HFU through a standard IP network and no hICN intermediate routers. We discuss more complex typologies, in which participants connect to the HFU through an hICN network at the end of the section. We also distinguish between uplink and downlink: the uplink, also known as contribution, is the connection between the participants and the HFU (or the central node in general), while the downlink, or distribution, is the connection from the HFU to the participants.

The way hICN-RTC distributes audio and video in the uplink is depicted in Figure 5.4. The HFU receives audio streams from all the participants by means of standard UDP connections, and uses the audio level information [60] to rank each participant according to their speaking activity [49]. The top-N participants are elected as active speakers and their videos will be displayed during the conference. The number N is decided by the application. Once the HFU has its rank, it starts to ask the video from the active speakers. Every active speaker publishes its video stream under a unique `name_prefix`, e.g., */video/participant-2/*. The video stream is segmented into a sequence of RTP packets, each of them carried into a single hICN data packet. The hICN name of each data packet is thus composed of the `name_prefix` of the participant followed by the sequence number of the RTP

packet, e.g, */video/participant-2/seg=x*. To retrieve a video stream, the HFU associates the SSRCs, available in the audio streams, to the `name_prefix` associated to the active speaker. The transport layer then constructs the entire hICN data name and retrieves the video. In the example in Figure 5.4, the application uses $N = 1$, and the HFU retrieves the video stream from participant 2 who is the only active speaker, i.e., the only one that will be displayed in the conference. The transport protocol used to retrieve the video is detailed in the next section.

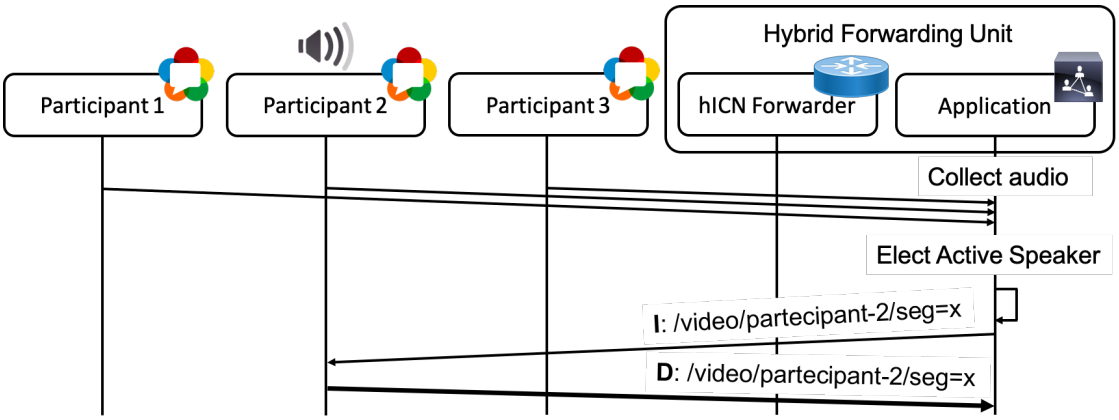


Figure 5.4: hICN-RTC architecture: uplink message exchange

We stress that in hICN-RTC the HFU pulls only the video from the N active speakers, while a standard SFU usually retrieves the video streams from all participants and drops those belonging to the non-active speakers. This is the first major difference between hICN-RTC and standard RTC architectures: since the content is pulled, only the streams that are really needed are requested, thus reducing the amount of traffic in the uplink.

In the downlink, every participant retrieves the video streams of the active speakers from the HFU, as shown in Figure 5.5. As soon as the HFU gets some valid data packets from an active speaker, it re-publishes them using some pre-defined hICN names, e.g., */video/active-speaker-1/seg=x*, */video/active-speaker-2/seg=x*. These names, which are distributed to the participants at connection time, are used by all participants to request the video of the active speakers. By doing so, all requests are aggregated by the hICN forwarder on the HFU, or they are directly served by the hICN forwarder *packet cache*. For example, in Figure 5.5, the three requests are aggregated at the forwarder and only one reaches the application. When the application produces the data (only once), the hICN forwarder serves all requests. In this way the application always receives one single request per packet generated by each active speaker, serving only N flows, instead of $N \times P$ flows, where P is the number of participants in the conference. Thus, the application can scale much more as assessed in the evaluation in Section 5.7.

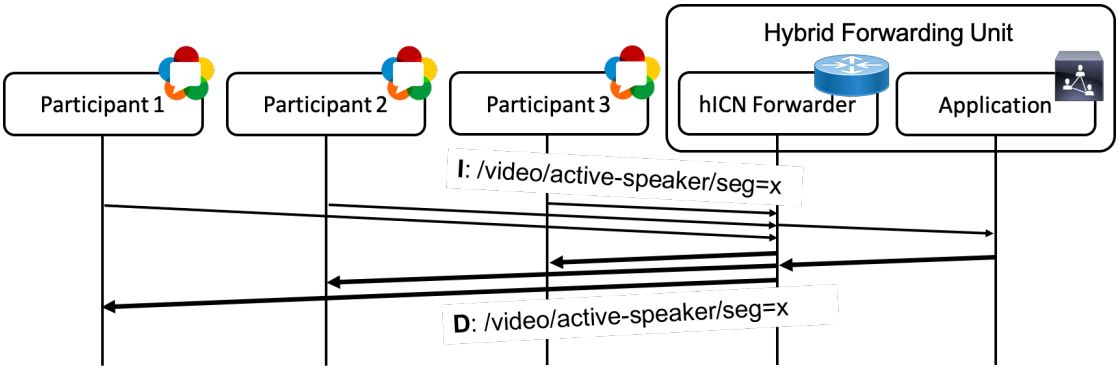


Figure 5.5: hICN-RTC architecture: downlink message exchange

We described how we can reduce the load from the application by exploiting the hICN forwarder at the HFU. However, the traffic sent by the hICN forwarder in the network is the same generated by a standard SFU, because it needs to replicate all the streams for all the conference participants. This load can be significantly reduced by moving from a star topology to one with some hICN forwarders between the clients and the HFU. These nodes will work as aggregation points, reducing the load on the network. We show the benefits of additional in-network hICN nodes in Section 5.7.

A drawback of the re-publishing approach used by hICN-RTC is that it invalidates the signature carried by each data packet. A simple mechanism that solves this problem is to allow the HFU to re-sign each packet, at the cost of: (i) increasing the per-packet computational cost at the HFU, (ii) breaking the end-to-end integrity and data provenance between consumers and producers which is one of the pillars of hICN. A different approach involves the adoption of a modified version of a hICN manifest [124], called *mapping manifest*. A mapping manifest contains the changes made by the HFU in one or more hICN data packet. By receiving a mapping manifest, a participant can reconstruct the original name in the data packet and verify the signature calculated by the producer of the data, i.e., the active speaker from which the HFU retrieved the data packet.

5.6 Realtime Information Centric Transport Protocol

As described in Section 5.3, hICN is a pull-based architecture: in order to retrieve a Data packet, a consumer needs to request it using an Interest packet. This communication pattern introduces two potential issues in the hICN-RTC architecture: (i) consumers may suffer additional latency since a full round-trip time (RTT)

elapses between the sending of an Interest packet and the reception of the corresponding Data packet; (ii) consumers need to know what data to request, i.e., the segment number of the newly generated data by the producer. In order to avoid these two problems, we designed a transport protocol called Realtime Information Centric Transport Protocol or RICTP. RICTP keeps at the consumer a window of pending Interests packets (Interest packets sent but not yet satisfied), for data that will be generated by the producer in the near future. In this section, we describe how RICTP decides what data to ask and how many pending Interest packets are needed for a consumer in order to get always fresh data in a timely fashion. The protocol that we describe here is implemented inside the hICN sockets. A detailed description of the hICN socket is available in [124].

At the current time RICTP is still a preliminary prototype that lacks many features. We are actively working to extend RICTP in order to support Simulcast and receiver-based adaptive bitrate selection as well as congestion-control. As discussed at the end of this section, we are also looking at packet loss recovery mechanisms, even if we think that the support given by the hICN network may reduce the need of complex recovery protocols such as FEC. This will be part of a future study. However, the preliminary results shown in Section 5.7.2 are encouraging.

5.6.1 RICTP Producer socket

Besides making available the RTP packets received from the application through Data packets, the RICTP producer socket also provides auxiliary information that consumers use to generate interests for new data. In particular, when the producer socket receives an interest, it evaluates if it can be satisfied. If not it notifies the consumer. Algorithm 2 describes in detail the actions executed by the producer.

When the producer socket receives an interest from the network the `OnInterest` function is executed. First of all, the producer extracts the segment number from the interest name, as well as the interest lifetime. Using these values, it computes the maxSeg, which is the largest segment number that will be produced by the socket before the expiration of the received interest. The producer socket also keeps track of the segment number that will be used for the next data packet. This value is stored in currentSeg. If the interest segment number is smaller than currentSeg or it is larger than maxSeg, then the interest refers to data that was produced in the past or to a data that will be produced too far ahead in the future. In these cases the producer replies with a negative acknowledgment (nack). A nack is a normal Data packet that contains the currentSeg and the current production rate of the producer, called prodRate. The cache lifetime of a nack is set to 0 to prevent them from being cached in the network and thus to ensure that the consumers always receive an updated nack. The handling of a nack by the consumer is described in the following section. In the other cases (segment $>$ currentSeg and

Algorithm 2 RICTP Producer Socket

```
Function OnInterest(interest)
    segment = getSegment(interest);
    lifetime = getLifetime(interest);
    maxSeg = currentSeg + lifetime * prodRate;
    if segment < currentSeg or segment > maxSeg then
        sendNack(currentSeg, prodRate);
    // else: do nothing, drop packet

Function OnDataRTP(rtpPacket)
    data = createData(rtpPacket, prefix, currentSeg);
    sendData(data);
    currentSeg ++;
```

segment < maxSeg), the producer does not generate a nack, it simply drops the received interest packet.

When the producer socket receives an RTP packet from the application the OnDataRTP function is executed: the producer generates an hICN data packet with the received RTP packet as payload, using as a name the prefix specified by the application and, as a segment number, currentSeg. This packet is then passed to the forwarder that runs underneath the producer socket. When the forwarder receives the data packet it satisfies all the pending interests for such data.

This ensures that the Data packet will be distributed to all the consumers, offloading the cost of packet replication from the application. Unlike nack packets, the Data packets are produced with a larger caching lifetime (1000ms in our implementation) in order to allow for loss recovery from in-network caches.

5.6.2 RICTP Consumer Socket

The RICTP consumer socket needs to fulfil two objectives: (i) sizing the window of pending Interest packets in order to avoid additional latency when retrieving the video, and (ii) learning the segment number used by the producer in order to ask for new data. To this end, RICTP uses a two phases approach: CATCH_UP and IN_SYNC. In the CATCH_UP phase the consumer does not know how many in flight Interest packets are needed in order to get fresh data from the producer. Therefore, during this phase the consumer tries to quickly estimate this value by increasing the pending Interest packets window exponentially. Once the consumer starts to get new data, it switches to the IN_SYNC phase, where it simply tries to remain in this state, taking into account possible network variations.

The consumer uses three values to keep track of the pending Interest packets: currentWin that indicates the current window size (i.e., the maximum number of Interest packets that can be in flight at a given point in time), maxWin (an upper bound for the window), and inFlight (the actual number of Interest packets in flight). At rounds of fixed duration Δ, the consumer estimates the minimum RTT to the producer and the producer's production rate, which are then used to update the size of the window.

Protocol Description

When the consumer joins the conference, it enters the CATCH_UP phase and starts requesting the packet with segment number 0. currentWin and inFlight are initialized to 1, and maxWin is initialized to MIN_WINDOW, a constant value that we set, in our implementation, to 5. The behaviour of the RICTP consumer is described in Algorithm 3.

Upon reception of a nack, the `OnDataNack` function is executed: first, inFlight is decremented, as a nack satisfies a valid Interest packet. Second, the nack's segment number, the inProduction segment and the estimatedProdRate (the last two are transported by the nack in its payload, and correspond to currentSeg and prodRate defined earlier) are extracted. As nacks indicate that the consumer is out of sync with the producer, we make use of the information given by the nack to try to quickly re-synchronize: the consumer sets nextSegment (the segment number of the next Interest packet) to inProduction + 1, and updates the estimation of the production rate using the value in the nack's payload. This value is used to size the window of pending Interest packets, as described later. If inProduction > segment, the consumer is asking for old data, therefore, it switches to the CATCH_UP phase and increases the window. In the opposite case (inProduction < segment, as the two values cannot be equal in a nack packet), the consumer is asking for data that are not yet produced and therefore the consumer is in sync with the producer, but it window of pending Interest packets is too large and has to be decreased. Finally, the consumer tries to send a new Interest packet.

Upon reception of a Data packet, the `OnDataRTP` function is executed: first, inFlight is decremented and the packet RTT (`updateRtt`) is computed, to be used to determine the minimum RTT at each round. Second, the hICN header is removed from the data packet and the resulting RTP packet is sent to the application. If the consumer is in CATCH_UP phase, the window size is increased. Finally, the consumer schedules new Interest packets. It is worth noting here that the socket sends to the application RTP packets as soon as valid Data packets are received, with no regard to the phase the consumer is in. In fact, valid Data packets can be received during both (CATCH_UP and IN_SYNC) phases, forwarding directly the resulting RTP packets to the application minimizes the time to video at the

Algorithm 3 RICTP Consumer Socket: Packet handling

```
Function OnDataNack(nack)
    inFlight --;
    segment = getSegment(nack);
    inProduction = getSegmentInProduction(nack);
    estimatedProdRate = getProductionRate(nack);
    nextSegment = inProduction +1;
    if inProduction > segment then
        phase = CATCH_UP;
        increaseWindow();
    else
        decreaseWindow();
        phase = IN_SYNC;
    scheduleInterest();

Function OnDataRTP(data)
    inFlight --;
    updateRtt(getRtt(data));
    sendContentToApp(getRTP(data));
    if phase == CATCH_UP then
        increaseWindow();
    scheduleInterest();

Function scheduleInterest()
    while inFlight < currentWin do
        sendInterest(prefix, nextSegment);
        inFlight ++;
        nextSegment ++;

Function OnNewRound()
    prodRateOnRound = receivedBytes/ roundDuration × τ;
    estimatedProdRate = α × estimatedProdRate + (1 − α) × prodRateOnRound;
    minRtt = getMinRtt();
    if no Nacks in the last 4 rounds then
        phase = IN_SYNC;
    updateWindow();
```

application.

Finally, the `OnNewRound` function is called upon the beginning of a new round. It estimates the production rate of the producer and the minRtt to the producer: estimatedProdRate is computed by dividing the bytes received in the last round by the round duration (roundDuration). To take into account estimation error, a factor τ is used. Furthermore, an exponential weighted moving average of the estimated production rate is used. minRtt is computed as the minimum RTT measured over the last N rounds of the protocol (in our implementation, $N = 30$), so that the estimation can take into account possible changes in the network RTT (this is achieved via the `getMinRtt()` function in Algorithm 3). If the consumer did not received any nack packet over the last four rounds, the consumer is considered in sync with the producer and therefore the phase is set to IN_SYNC. At last, the window of pending Interest packets is updated.

Window Adjustment

Algorithm 4 describes all the functions used by the RICTP consumer to handle the window of pending Interest packets.

The function `computeMaxWindow` is used to compute the upper bound of the window size, maxWin. It is computed as a sort of bandwidth delay product, where the bandwidth is the estimated production rate of the producer and the delay depends on the consumer phase: in the CATCH_UP phase, maxWin is set as the product of estimatedProdRate and interestLifetime, while in the IN_SYNC phase, maxWin is the product of estimatedProdRate and minRtt. In the CATCH_UP phase, interestLifetime is used as delay because (i) during the CATCH_UP phase, the consumer may have a wrong estimation of the minimum RTT and (ii) it allows to generate a larger window (since an Interest packet lifetime is higher than an RTT), increasing the chances to quickly match the producer production rate, even in case of large network delay. Using the minRtt in the IN_SYNC phase allows the consumer to maintains a window size close to the minimum value required to remain synchronized with the producer.

During the CATCH_UP phase, the consumer tries to quickly synchronize with the producer, therefore it increases the window size exponentially: every time a packet (Data or nack) is received (except when the nack concerns a segment that is not yet produced: inProduction $<$ segment), the `increaseWindow` function is called. This function first calls the `computeMaxWindow` function, and then increases currentWin if possible (currentWin must be less or equal to maxWin).

When a nack concerning a segment that is not yet produced is received, the window size is decreased through the function `decreaseWindow`. When the function is called for the first time in a round, the window size is multiplied by 2/3, while for subsequent nacks, the window size is reduced by 1. As most of the time nacks

Algorithm 4 RICTP Consumer Socket: Window adjustment

```
Function computeMaxWindow()
    if phase == IN_SYNC then
        delay = minRtt;
    else
        delay = interestLifetime;
    maxWin = delay * estimatedProdRate;

Function increaseWindow()
    computeMaxWindow();
    currentWin = min(maxWin, currentWin +1);

Function decreaseWindow()
    if is the first time in this round then
        currentWin = currentWin *2/3;
    else
        currentWin --;
    currentWin = max(currentWin, MIN_WINDOW);

Function updateWindow()
    if phase == IN_SYNC then
        computeMaxWindow();
        if currentWin < (maxWin *2/3) then
            currentWin = min(maxWin, currentWin * α);
        else if currentWin > maxWin then
            currentWin = max(currentWin * β, MIN_WINDOW);
```

are received in batches, this prevents decreasing too much the window size, which could lead to a desynchronization of the consumer and the producer. In any case, the window size is bounded by MIN_WINDOW, which is the lowest possible value.

During the IN_SYNC phase, the goal is to keep the window size as stable as possible, and therefore the function `updateWindow` is run once per round. If currentWin $< 2/3 \times$ maxWin, the window size is increased by an increase factor α. If currentWin $>$ maxWin (this may happen when switch from CATCH_UP phase to IN_SYNC), the window size is decreased by a decrease factor β. In the other case, the window size remains the same.

Packet Loss Recovery

In the case of a lossy network the consumer is responsible to recover the lost packets. In hICN, like in standard ICN, a loss is detected when an Interest timeout occurs. This happens if an Interest packet is not satisfied within its lifetime. Unfortunately this is not enough in real time applications, since the Interest lifetime can be quite large, and when the loss is detected it is too late to recover the packet. To overcome this problem we take advantage of the RTCP packets generated by the application. In the standard WebRTC, the application generates an RTCP Generic NACK [100] packet as soon as a loss is detected. In our implementation we intercept these packets in our socket and we transform them into Interest packets. The generated Interest packets can then retrieve the corresponding Data packets from in-network caches (if available) without having to be forwarded up to the producer. Using this technique we are able to promptly detect losses, overcoming the limitation of the timeout approach of ICN, and to reduce the time required to retrieve lost packets with respect to the standard implementation, which requires end-to-end retransmissions. This, combined with other in-network loss recovery such as WLDR, described in [28], may reduce the need for more complex recovery mechanisms such as FEC. We leave the discussion of this topic open for future work.

5.7 Evaluation

In the following sections, we gather the results of a performance evaluation of hICN-RTC: in Section 5.7.2, we assess the efficiency of RICTP, the pull-based transport proposed in Section 5.6, in terms of response and quick retrieval of media streams from the participants; and in Section 5.7.3, we report the scalability analysis of hICN-RTC and compare it to state-of-the-art SFU-based WebRTC architecture. The motivation for the former evaluation is to show that RICTP does not introduce any additional latency in real-time video distribution, while the latter shows that

hICN-RTC scales with the number of active speakers rather than the number of participants in the conference.

5.7.1 Implementation and Experimental Settings

To implement hICN-RTC we have modified the code provided by the open source project Jitsi [5]. We have implemented our hICN-RTC clients as web applications, customizing WebRTC library inside Chromium [46]. The main modification inside WebRTC library is the replacement of standard sockets by hICN ones. The hICN sockets are implemented inside the hICN transport library, which provides multiple transport protocols, but in our experiments, we used RICTP, described in Section 5.6. Each client runs its own hICN forwarder, which implements the hICN network stack. Both hICN transport library and forwarder are extensions of the open source code published in CICN as part of the FD.io project [134].

The application running on the HFU is a slightly modified version of the Jitsi video-bridge. While we reuse most of the application logic, in particular the speaker ranking that has a central part in our architecture, we have implemented, at the application level, the video distribution protocol described in Section 5.5 (i.e., the data renaming and the data publication) and hICN sockets with RICTP are used in the application as well. The hICN forwarder running in the HFU is based on VPP [84], which allows the forwarder to handle traffic in a more efficient way. When in-network hICN nodes are used, their forwarder is also VPP-based.

All the participants in our conference produce a single video stream with an average bitrate of 500 kbps at 25 frames per seconds (fps). We have also forced the WebRTC library to produce new key frames at constant intervals, one every 100 frames, as recommended by WebRTC specifications. Since our stream produces 25fps, each client produces a new key frame each 4 seconds.

5.7.2 RICTP benchmarking

In this section we run some benchmark experiments to evaluate RICTP, the hICN-RTC transport protocol proposed in Section 5.6. First, we start by measuring the impact of the network delay on RICTP. For this purpose, we designed a simple scenario: three users are connected to a video conference and a fourth one joins it after a while. We then measure the time it takes for the new participant to receive the first valid Data packet, i.e., the time elapsed between the connection of the participant and the reception of the first video packet. We vary the RTT between the new user and the HFU using the netem qdisc [42]. For each RTT value, we run the experiment 30 times and compute the average and the standard deviation of the collected measurements. We report the results in Figure 5.6.

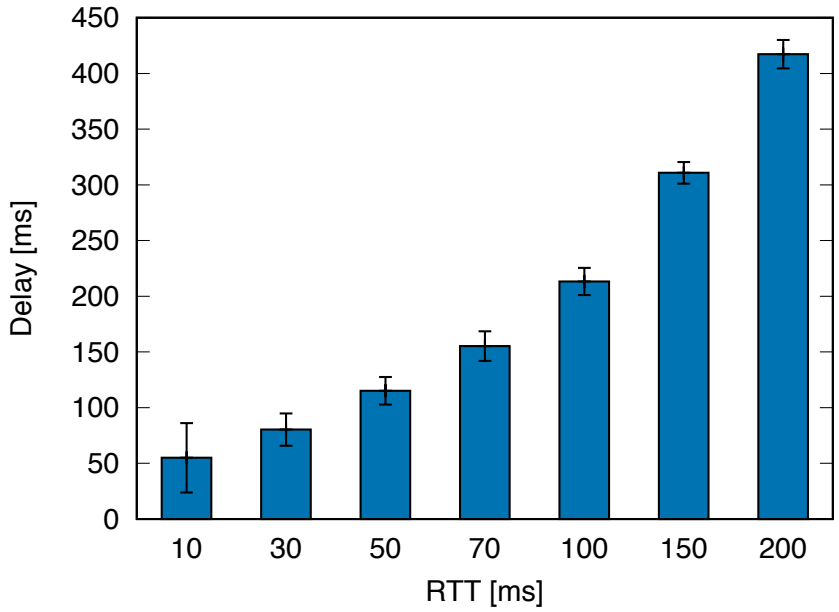


Figure 5.6: Time elapsed between the first Interest packet sent by a user and the reception of the first valid Data packet varying the RTT

The plot shows that the new participant is able to get the first data packet in about 2 RTTs, most of the time, meaning that RICTP is robust to network delay. The only case where this observation does not hold is when we set the round trip time equal to 10ms. This is due to the video production rate used in our tests. In fact, when the new client joins the conference, it asks for video segment 0 and receives a nack with the segment number of the packet that will be produced next, say segment s. At this point, the client asks for segment $s + 1$ (according to Algorithm 3). The producer generates a stream at 500 kbps, meaning one packet every 17ms (considering 1100 bytes the average size of an RTP packet). This means that the reply to the second Interest packet may require up to 34ms, that summed to the 2 RTTs, roughly gives the 50ms of the plot. This also explains the higher variation on this result, since the delivery time depends on when the new participant joins the conference. For higher RTT values, the production delay is masked both by the RTT itself and by the fact that the new participant can fetch the required data from the HFU cache.

However, in order to be able to render the video, a single data packet is not enough and the client needs to receive an entire key frame. As described before, a participant generates a key frame every 4 seconds. When a new client joins the conference, it will receive a key frame in 2 seconds (on average) after reception of the first valid data packet. Therefore, reducing the time needed by a participant

to receive the first valid data packet helps reduce the time elapsed between a participant joining a conference and rendering the video. We acknowledge that the key frame reception can be sped up by explicitly asking for a new key frame at connection time, like WebRTC does using RTCP PLI packets [100]. We leave the explicit request of a key frame for future work.

In a second experiment we show that RICTP does not add extra delay in addition to the one introduced by the network during the conference, even if it is a pull-based protocol. We run again our conference with 4 participants connected directly to the HFU and we measure the time that elapses between the creation of a Data packet at producer side and its reception at consumer side. We run the experiment in two settings: in the first setting, each link has 10ms of delay, resulting in a total RTT between two participants, passing through the HFU, of 40ms; in the second setting the link delay is set to 30ms, with a total RTT of 120ms. The results in Figure 5.7 are obtained from one minute of conference.

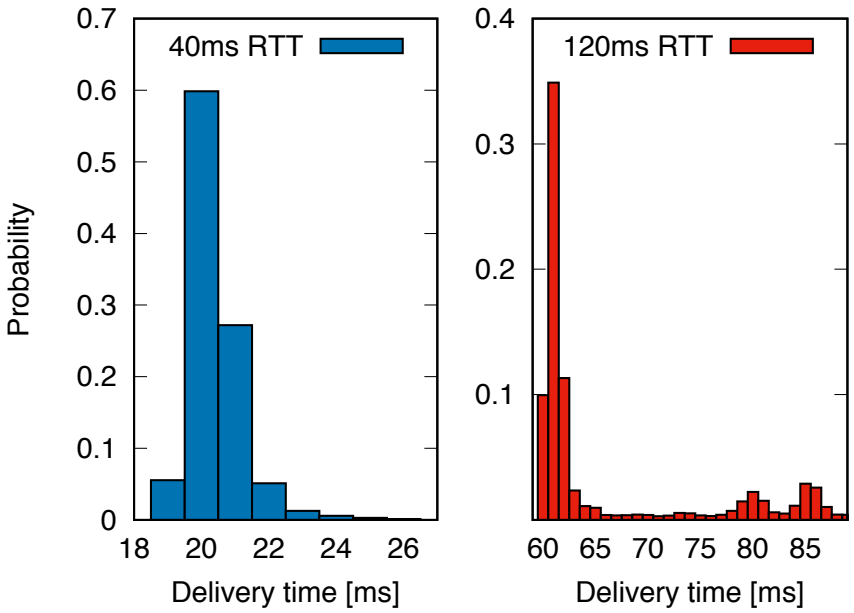


Figure 5.7: Distribution of the time elapsed between the production of an RTP packet and its reception at the consumer

As expected, the plot shows that the majority of the packets are received within 1/2RTT, meaning 20ms in the case of 40ms RTT and 60ms in the case of 120ms RTT. Is it worth noticing that with 120ms RTT some packets are received with a small additional delay. These are the packets taken from the HFU cache. A participant may retrieve content from the cache if it goes temporally out of synch with the producer. However the cache helps to mask this border effect and

helps the client to keep downloading and watch the video without switching to the `CATCH_UP` phase of RICTP transport protocol.

5.7.3 hICN-RTC Scalability

In this section we show the better scalability of hICN-RTC with respect to baseline WebRTC. More precisely, we set up a conference with many participants and measure the traffic generated in the network, as well as the CPU load on the central node. To test WebRTC we use a star topology where we use the Jitsi SFU to connect all the clients. In the hICN-RTC case, we use the topology depicted in Figure 5.8. It offers the same star topology as WebRTC for the comparison (upper cloud), and we add a group of participants connected to the HFU through an additional hICN node (lower cloud) to distinguish incremental benefits resulting from hICN network enablement. In both cases the hICN traffic needs to traverse an IP network to go from the clients to the HFU and vice versa. In all experiments, we progressively add participants to the conference, setting the number of active speakers equal to 3 (each client will display over time up to 3 videos in parallel).

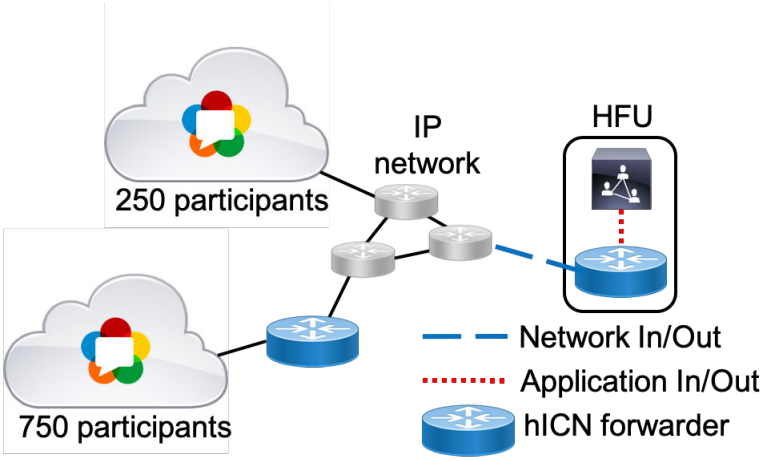


Figure 5.8: Topology used for scalability tests.

The workload for the WebRTC experiments is generated using Selenium Grid [9]. We run multiple LXC [7] containers, each one containing an instance of Chrome and a Selenium node. such nodes are controlled by a centralized orchestrator that runs Jitsi Torture [6]. This tool sets all the parameters of the experiment, such as number of participants, duration of the experiment and many others. The generation of the workload in this setting is quite heavy and we were not able to scale it to more than 100 users due to the amount of resources required. In fact we used 5 servers with 24 Xeon E5-2690 v3 CPUs each, for a total of 240 cores using

hyper-threading, and a sixth server to run the Jisti video-bridge in isolation, in order to reduce context switching that may have altered our measurements. For the hICN-RTC workload, instead, we used a simple test application running over RICTP transport. This application does not perform encode/decode operations of the video and is therefore much more lightweight than a Chrome instance. This allowed us to run many more clients in our testbed, up to 1000 clients with 3 servers only.

Figures 5.9 and 5.10 show the overall traffic generated in the conference. For hICN-RTC we distinguish between network and application traffic. The network traffic is the traffic received and sent by the hICN forwarder on the HFU. This is measured on the link represented by a dashed blue line in Figure 5.8. The application traffic is the traffic received and sent between the application and the HFU, measured on the link represented by a dotted red line in Figure 5.8.

The uplink traffic (the traffic from the clients to the HFU) is reported in Figure 5.9. In the chart, we show how the traffic varies increasing the number of participants in the same conference. In particular, for hICN-RTC we connect up to 250 participants directly to the HFU, and we further increase the number of participant up to 1000, connecting the new clients behind an hICN node. The results in presence of such added clients are depicted with a grey background in the plot. As already discussed above, we were only able to go up to 100 clients for WebRTC, given the much higher amount of computing resources required for the experiment. The log scale on the y axis highlights the different scaling behaviour between WebRTC and hICN-RTC.

Specifically, we observe from the plot that the uplink traffic generated by WebRTC increases linearly with the number of participants, as each one of them sends its contribution (its media streams) to the SFU. With hICN-RTC, the uplink traffic also grows when the number of participants increases, but only to account for the increase of Interest packets sent by the users, which are clearly much smaller in size. However, the traffic that reaches the application is always constant in hICN-RTC, regardless of the number of participants (the application traffic is depicted with red solid bars on the graph), and corresponds to the three videos produced by the active speakers.

Moreover, the plot shows that adding an hICN router in the network helps to improve even further the scalability of the system: not only the application traffic remains unchanged, but also the incoming network traffic remains constant when the number of users increases (gray zone from 250 to 1000 participants in the graph). This is due to the request aggregation performed by the intermediate hICN node in the network between clients and HFU.

The downlink traffic (the traffic from the HFU to the clients) is reported in Figure 5.10. As expected, the downlink traffic in hICN-RTC is comparable to

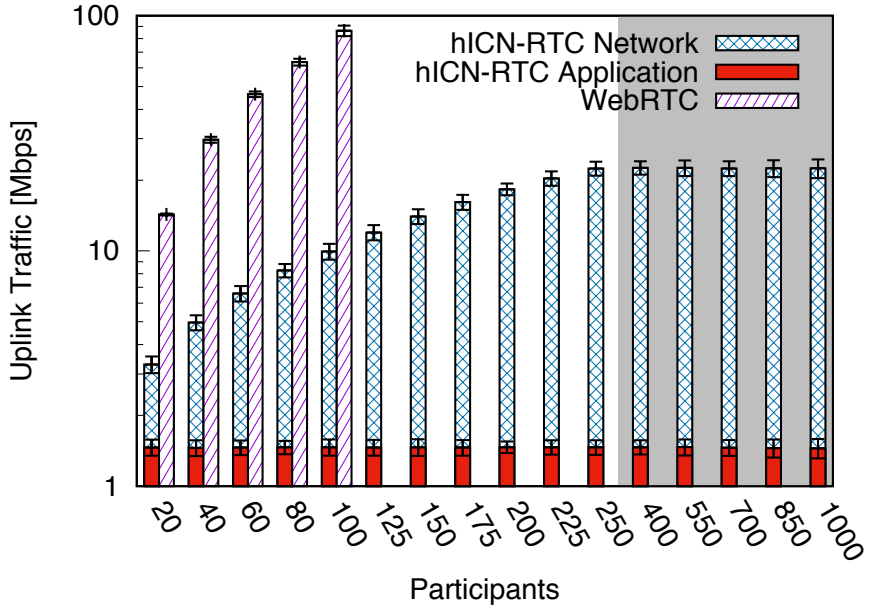


Figure 5.9: Uplink traffic.

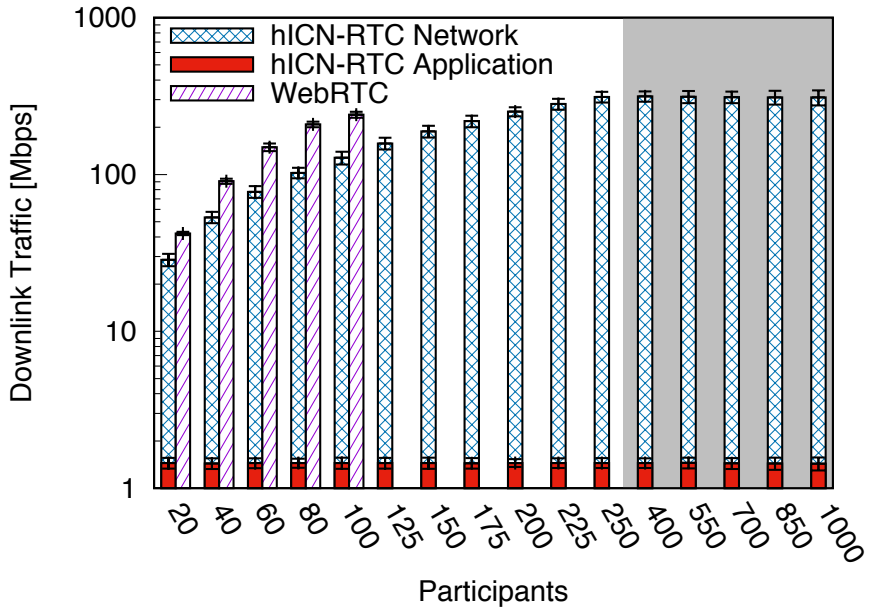


Figure 5.10: Downlink traffic.

the downlink traffic generated by WebRTC. However, the traffic coming from the application is again constant, showing that hICN-RTC scales with the number of active speakers, not with the number of participants. We also observe that the

introduction of an hICN router in the network keeps the downlink traffic constant, again as a result of the request aggregation.

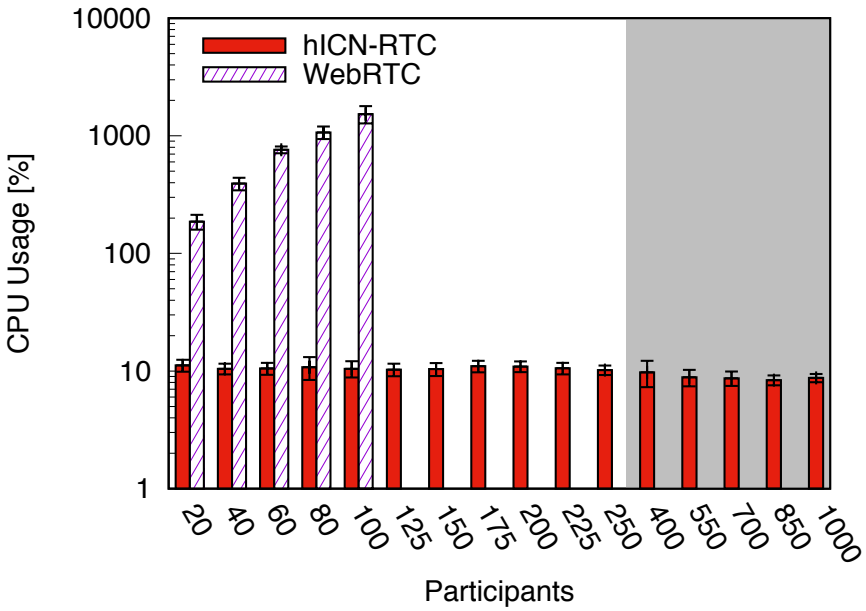


Figure 5.11: CPU usage.

The CPU usage in the central node is reported in Figure 5.11. We observe that the CPU usage grows linearly in the case of WebRTC, while it remains constant when using hICN-RTC. In particular, the plot shows that hICN-RTC supports a conference with 10 times more participants with respect to WebRTC, using 150 times less CPU on the central node: 100 participants in a WebRTC conference correspond to a 1500% (or 15 CPUs) usage, while 1000 participants in hICN-RTC correspond to only 10% of one CPU.

5.8 Conclusion

The capability to scale with a large number of participants is key for WebRTC to sustain increasing adoption in multiparty collaboration systems and in emerging low-latency media streaming applications.

ICN communication model (embodied here by hICN) appears as an appealing option for providing an efficient (multi)-point to (multi)-point communication model that scales with content, i.e. number of active media streams, rather than end hosts, i.e. number of participants. The use of ICN connectionless and pull-based transport could allow to overcome the limitations of existing point-to-point

model, especially in presence of mobile or multi-homed devices, and potentially provide interesting features in terms of low-latency congestion control (e.g. reliability), caching and security. Previous attempts have either partially integrated ICN characteristics with limited advantages or proposed a clean slate architecture not compatible with WebRTC application layer.

In this chapter, we investigate the potential scalability benefits of leveraging ICN in WebRTC stack, while minimizing modifications overall and specifically in the application layer. To this aim, we select a recently proposed incremental design of ICN implemented inside IP, Hybrid ICN (hICN), and propose hICN-RTC, a new WebRTC architecture that fully exploits hICN features.

A light SFU-based topology is assumed where the central control node (renamed HFU) offloads operation to the underlying network stack. It results in a drastic reduction of network traffic as well as of CPU usage at the HFU.

We carry out a first evaluation assessment by means of hICN-RTC implementation in Jitsi. Results show that (i) application traffic in the HFU (i.e., the traffic handled by Jitsi Videobridge) is minimized in hICN-RTC as a result of the RICTP pull-based transport model: it becomes proportional to active speakers' media streams only, rather than to the aggregate of the media streams produced by all participants. (ii) The request aggregation performed by the hICN forwarder in the HFU shields the application layer from handling the majority of traffic, minimizing the CPU utilization which remains low and constant regardless the number of participants in the conference. (iii) The introduction of an additional hICN-enabled node in the network between a group of participants and the HFU may help to further reduce traffic, as a consequence of in-network request aggregation and caching.

As predicted, such features of hICN-RTC allow the collaboration system to scale with the number of active participants, i.e., the number of displayed media streams at client side, which is a parameter decided by the application, rather than with the number of total participants, e.g., *supporting with hICN-RTC a conference with 10 times more participants than in WebRTC using 150 times less resources on the media bridge*. A series of benchmarking tests also show that RICTP, the pull-based transport presented in the paper, does not introduce additional latency to the video distribution, validating the feasibility of our proposal.

Besides the encouraging results, the design, implementation and evaluation of hICN-RTC are still at an initial stage. As future work, we plan to enrich hICN-RTC with multiple features and to leverage more of the properties that the underlying hICN communication paradigm offers. More specifically on transport protocol features, we plan to extend basic loss and congestion control in RICTP, as well as to add the capability to explicitly request a new key frame as a way to reduce the time required to render the video when a new client joins the conference. In all

such cases, in-network caching property of hICN would be leveraged for retrieval of data directly from the cache, with no involvement of the application. We also plan to implement Simulcast with bitrate adaptation decided at the client side in a way that minimizes signaling in the network and leverages the statistics collected on path in virtue of the hICN interest/data symmetric routing principle. Finally, we believe that hICN-RTC holds promises for improved mobility and object-based security in WebRTC as naturally inherited by hICN, still yet to be evaluated in WebRTC context.

Chapter 6

Conclusion

Information-Centric Networking (ICN) appears as a promising candidate to solve the *mismatch between Internet usage and its architecture*: nowadays, Internet usage is mostly centered on information dissemination and retrieval, while its architecture still relies on an end-to-end model IP-based model that presents some limitations when it comes to multimedia delivery. Several OTT approaches were deployed to overcome this mismatch, such as CDNs and peer-to-peer networks, and carry today a large fraction of the Internet traffic. However, such approaches are far from perfect as technical inefficiencies such as mobility management, dynamic content-to-location bindings, multicast or multi-homing, require complex solutions to be handled. ICN offers a novel networking paradigm, where the content itself is the focus point, rather than its location: rather than addressing the hosting server of the content, the content itself is addressed in ICN. To put it simple, the *Where* is replaced by the *What*. In ICN, mobility management, multicast or multi-homing issues are gracefully handled, thus simplifying networks. In this thesis, we look carefully at the interaction of ICN and multimedia delivery, with a particular attention to the resulting user's quality of experience. To do so, we considered several ICN architectures, namely NDN, CICN and hICN, but the results we gathered are independent from the architecture and we argue that our results still hold under other ICN architectures. We also considered several multimedia delivery cases: Video on Demand, using broadly deployed ISO standard MPEG-DASH, and Real-Time Communication, relying on WebRTC.

For the DASH use-case, we contrasted the performance achievable by DAS systems (represented by MPEG-DASH) using rate-based and buffer-based adaptation logics, on top of an ICN or a TCP/IP network stack. Our experimental campaign resorted on different channels (DASH profiles, Wi-Fi and LTE access emulated via NS-3, or real 3G/4G traces), multiple clients (in homogeneous or heterogeneous population mixture (with respect to the network stack they used), with synchronous or asynchronous arrival patterns) and various adaptation logics.

We showed that when combining simple building ICN blocks, such as refined bandwidth estimation, in-network loss detection and recovery or multi-path forwarding, the performance of DAS over ICN matches and possibly significantly outperforms (in terms of user's quality of experience) the performance of DAS over TCP/IP. However, we also show that to be efficient, these building blocks need to be used *together*, otherwise, the DAS performance over ICN can worsen with respect to TCP/IP.

However, this comparison only considered the benefits of ICN at the client-side, without considering in-network caching that can be leveraged by ICN. Therefore, we explored how in-network caching, along with network assistance, would impact the user's quality of experience. Specifically, we explored the design space of in-network caching and pointed out (although we were not the first ones to do so), that while in-network caching can improve the average video quality for the user, it may also induce quality oscillations, degrading the user quality of experience. We then proposed a simple signal from a cache that periodically exports to the clients the per-quality cache hit-rate. This signal, coupled with a per-path bandwidth estimation, successfully prevent the cache-induced quality oscillations, while maintaining a comparable average quality and increasing the cache hit ratio. We presented the integration of such signal to one adaptation logic, namely AdapTech, but we argue that such network assistance can be leveraged by all rate-based adaptation logics.

Due to the limited storage and the temporality of caching, in-network caching is particularly appealing for popular contents, with a lot of watchers over a short time span, such as live events (e.g., sports events such as the soccer world cup 2018, or Olympics). While DASH presents advantages over live streaming protocols, such as RTP, RTSP or RTMP, it cannot sustain ultra-low latency constraints, as required in Real-Time Communication that is used for video conferencing. In such context, WebRTC is gaining popularity, but as its distribution model is not designed for large scale, scalability is one of the key metrics to validate WebRTC as a credible candidate to support large scale low latency streaming services. To improve scalability, the architecture of traditional multiparty conferencing tools moved from peer-to-peer to centralized with MCU-based and SFU-based architectures. However, we argue that the scalability of such architecture can be improved by leveraging ICN features and therefore we designed and implemented hICN-RTC, a new WebRTC architecture that fully exploits hICN features. This architecture builds on a SFU-based architecture, where the central node, called HFU, offloads operations to the underlying network stack. The results of our first evaluation of hICN-RTC are promising: the traffic handled by the HFU is proportional to active speakers' media streams only, rather than to the aggregate of the media streams produced by all participants, thanks to RICTP, a pull-based transport protocol

that was designed for hICN-RTC. As a consequence, the CPU utilization remains low on the HFU. Furthermore, in-network caching and PIT aggregation can help to further reduce traffic.

Overall, in this thesis we show the benefits that ICN can bring to multimedia delivery, both in terms of user's quality of experience and in terms of scalability for WebRTC architecture.

Appendices

Appendix A

Tools used in our experiments

A.1 vICN

vICN (virtualized ICN) [125], is a data model-driven management system that offers orchestration services and emulation of radio channels (LTE and Wi-Fi). It relies on Linux containers to emulate large-scale networks. In a nutshell, vICN relies on vICN topology files to describe the topologies to deploy. A vICN topology file consists of a set of *resources* defined in a JSON file. These resources are virtual representations of elements to be deployed by vICN. Such resources include physical nodes, containers, applications, or network links and each resources is further described by *attributes* . An example of such resource is given in Figure A.1: here, the resource describe a Linux container ("type" : "LxcContainer"), named *cons* ("name" : "cons"), using the *cicn-image* image ("image" : "cicn-image") and deployed on the physical node named *server* ("node" : "server").

When a topology file is given to vICN, it will first parse the JSON file and break down the resources dependencies to issue a set of instructions to execute in order to deploy the desired topology. If we take again our example in Figure A.1, to deploy this container, vICN needs first to resolve where the node on which the container will be deployed is located (basically, a physical server is described as a physical resource with a hostname). Then vICN needs to find where the container image specified is stored. Once all of this is done, then vICN can instantiate the *cons* container.

vICN presents a certain amount of resources, including *CentralIP*, a virtual resource that assigns IP addresses and sets up the IP routing over the generated topology. vICN also offers as resources applications from the CICN suite, described in the next section, such as CICN forwarder. Finally, as vICN is open source and modular, it is easy to add new resources to deploy new elements (e.g., new applications) using vICN. Instructions to get the code and install vICN can be

```
{
        "type" : "LxcContainer",
        "image": "cicn-image",
        "name" : "cons",
        "groups": [ "virtual" ],
        "node" : "server"
},
```

Figure A.1: An example of resource description in a vICN topology file.

found here [10].

A.2 The CICN suite

The open-source CICN project [134], as part of the FD.io project, offers a set of open-source CICN applications.These applications include a CICN forwarder (called Metis), an HTTP server and a DASH video player, called Viper, which is described in more details in Appendix B. The HTTP server is an HTTP proxy server, able to serve client requests using both TCP and ICN as underlying protocols. The ICN flavour, so far, only supports the HTTP GET method.

A.3 Videos used

In our experiments, we used several open-source videos. More specifically, we used Big Buck Bunny from the Blender Institute, Tears of Steel from the Blender Foundation and Sintel from the Blender Foundation. When available, we used DASH dataset of these videos, found here [77], otherwise, we did the DASH encoding ourselves.

Appendix B

Viper

Viper (VIdeo PlayER) is an open-source video player based on the official reference software of the ISO/IEC MPEG-DASH standard, *libdash*[1], which provides an object-oriented interface to the MPEG-DASH standard. Viper is part of the CICN suite [134]. It is a dual stack DASH player, that supports video retrieval using either TCP or CICN from an HTTP video server. As it was mainly designed as a prototype application for demonstrations, there is currently no support for audio streams, but this feature is planned to be added later on. Moreover, a headless version of Viper is available, which only sequentially downloads the video segments (i.e., it does not decode the video segments) as to limit the CPU usage of one instance of Viper in headless mode. Limiting the CPU usage to the downloading of segments is particularly useful to run several instances of Viper on the same machine, for experimental campaigns purposes.

Furthermore, Viper implements a variety of adaptation logics from the literature, such as BBA [59], PANDA [83], AdapTech [15], BOLA [131]. Viper code was designed to be modular and therefore, adding new adaptation logics to it is fairly easy to do, hence enabling Viper to be also used to test new adaptation strategies. Besides, the network stack can be easily modified as well, e.g., to have Viper run on top of NDN [143] or on top of QUIC [54].

Viper offers a GUI to set the various parameters of the player, such as which network stack to use, the size of its buffer (in video segments), the adaptation logic and its associated parameters. It also presents a field to enter the URL of the MPD of the video to be fetched. For the headless version, all of these parameters can be passed to the player through the command line. Furthermore, plots are available to display, in real-time, user centric metrics, such as the buffer level, the quality at which current segments are being downloaded and the quality of the video segment currently displayed, Figure B.1 presents a screenshot of Viper, displaying

[1] https://github.com/bitmovin/libdash

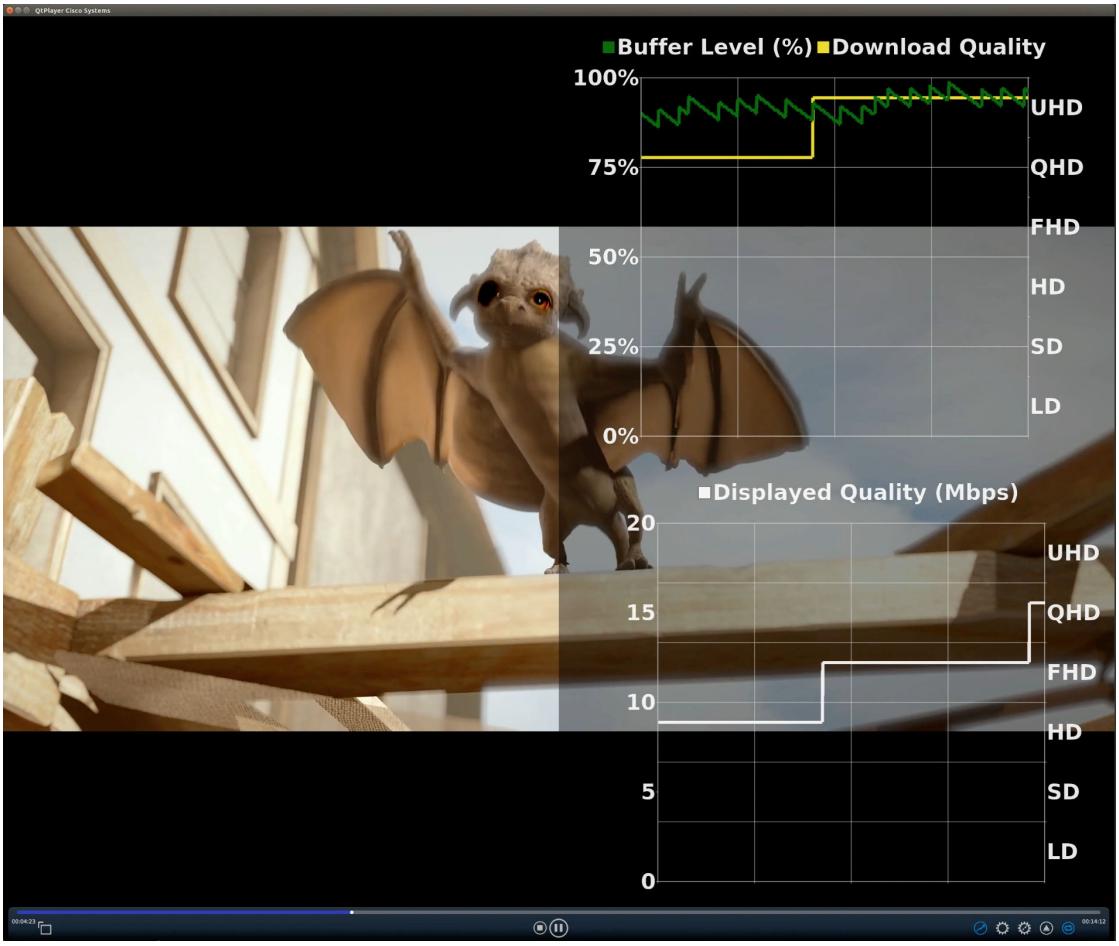


Figure B.1: Screenshot of Viper playing Sintel and displaying in real-time the buffer level, the quality at which current segments are downloaded and the video quality displayed.

such plots. A websocket [40] inside Viper can be used to export these user centric information to external entities to collect stats or to monitor the behaviour of a headless instance.

Finally, Viper is available and has been tested on several platforms, such as Ubuntu 16.04, MacOSX 10.13, Android 7 and iOS 10. Code and instructions to build are available here: [11].

Appendix C

Small guide for DASH experiments

"Dynamic Adaptive Video Streaming: Towards a systematic comparison of ICN and TCP/IP"

Reproducing 'Calibration' results (A Quick Guide)

This short guide will help you reproducing all the results reported in the Calibration section of Chapter 3. In particular, we investigate the performance of a DAS system over a TCP/IP vs ICP/NDN stack, using three different adaptation logics, namely PANDA, BOLA, and AdapTech (please refer to Chapter 3 for further details and bibliographic references).
In order to reproduce the experiments you will need:

- The `icn_das_lxc.tar.gz` tarball, available at `http://jaqu.eu/thesis/DATA/icn_das_lxc.tar.gz`, used to create the Linux Container (LXC) image that will be instantiated as both the Client and the Server of the simple scenario under investigation. An Ubuntu environment, along with the encoded video and the Apache web server are provided with the tarball.

- The `icn_das_code.tar.gz` tarball, available at `http://jaqu.eu/thesis/DATA/icn_das_code.tar.gz`, which contains scripts to setup the topology and launch experiments, as well as scripts to post-elaborate produced log files (inside the *results* folder). Source code is also available: *player.tar.gz* contains code for libdash, dash.js, and qtsampleplayer (inside which the three adaptation strategies are coded), while *client.tar.gz* contains the code for the ICP client.

Prerequisites

- **LXC**

`sudo apt install lxc`

- **LXD**

  LXD is the lightweight container hypervisor. For prerequisites, installation and configuration please follow instructions at:

  `https://github.com/lxc/lxd`

  We configuring LXD with `sudo lxd init`, you can decide the type of storage backend to be used: `dir` (default) or `ZFS`. In case you want to use ZFS, you may need to install it on your machine (e.g., for Ubuntu < 15.10):

  `sudo apt-add-repository ppa:zfs-native/stable`

  `sudo apt update`

  `sudo apt install ubuntu-zfs`

- **Open vSwitch (OvS)**

  For the installation of OvS on 14.04 LTS Trusty (3.19.0), i.e., the only one tested so far, please refer to the following instructions (taken from `https://gist.github.com/olegslavkin/af989e7a850eb67fa779`):

  - Install the git package
    `sudo apt-get update`
    `sudo apt-get -y install git`
  - Clone OvS git Repo
    `git clone https://github.com/openvswitch/ovs.git`
  - Install tools
    `sudo apt-get -y install autoconf libtool make`
  - Build OvS
    `cd ovs/`
    `./boot.sh`
    `./configure --with-linux=/lib/modules/`uname -r`/build`
    `make`
  - Install OvS
    `sudo make install`
    `sudo make modules_install`

- Create OvS Database

  `sudo ovsdb-tool create /usr/local/etc/openvswitch/conf.db vswitchd/vswitch.ovsschema`

- Run OvS

```
sudo ovsdb-server --remote=punix:/usr/local/var/run/openvswitch/db.sock \
    --remote=db:Open_vSwitch,Open_vSwitch,manager_options \
    --private-key=db:Open_vSwitch,SSL,private_key \
    --certificate=db:Open_vSwitch,SSL,certificate \
    --bootstrap-ca-cert=db:Open_vSwitch,SSL,ca_cert \
    --pidfile --detach
sudo ovs-vsctl --no-wait init
sudo ovs-vswitchd --pidfile --detach
```

Calibration Experiments

Creation of a LXC Image

In order to create a LXC image, tarball `icn_das_lxc.tar.gz` (downloaded from `http://jaqu.eu/thesis/DATA/icn_das_lxc.tar.gz`) must be imported using the following command:

```
lxc image import icn_das_lxc.tar.gz --alias icn_das
```

where `icn_das` will be the alias associated to the LXC image. Please note that if using a different alias, the change must be reflected inside the provided script `launch.sh`, where also the name of the host interface must be changed if different from `eth0`.
Type `lxc image list` in order to check the presence of the newly created image.

Instantiation of Client and Server Containers

The idea (so far) is that of (i) launching the two containers (i.e., Client and Server), (ii) uploading scripts and source code (i.e., libdash, dash.js, qtsampleplayer, etc.), and (iii) compiling them directly inside the Client container. This means that possible modifications of the source code can be done either inside or outside the container environment (in the latter case, a new tarball must be created after having modified the code).

In order to have the two containers ready to use, place your terminal inside the *icn_das_code* directory, and execute the provided script:

```
./launch.sh
```

At the end of the experiments, containers can be removed by executing:

```
./cleanup.sh
```

Experiments

The provided container and scripts can be used to experiment with dynamic adaptive streaming by changing different parts of the whole system, like network stack, link conditions, adaptation logics and their parameters. As a consequence, a minimum level of automation in the experiment campaign would be beneficial; to reach that, it is required to:

- Guarantee a *password-less ssh access* between the two containers:
  - Attach to the Client bash: `lxc exec client bash` (default way);
  - Type: `su ubuntu`; `cd ~`;
  - Generate a pair of authentication keys: `ssh-keygen -t rsa` (press Enter at all steps);
  - Append the new public key in Server *authorized_keys*:
    1. Copy the Client public key: `cat .ssh/id_rsa.pub`;
    2. Attach to the Server container (opening a new tab): `lxc exec server bash`;
    3. Copy the Client public key inside `.ssh/authorized_keys`;
    4. After the first login (*ssh root@10.2.0.1*), you will not be asked for the password anymore.
- Attach to the Client container with the possibility to execute the *screen* terminal:
  `lxc exec client -- sh -c "exec >/dev/tty 2>/dev/tty </dev/tty && /bin/bash"`

Once in `ubuntu@client:~$` (i.e., after `su ubuntu`; `cd ~`), Calibration experiments for the three adaptation strategies can be launched by executing the respective script:

```
./Exec_PANDA_Experiments.sh
./Exec_BOLA_Experiments.sh
./Exec_AdapTech_Experiments.sh
```

Since experiments cannot be executed in parallel, it might be useful to launch a *screen* terminal, execute one of the script, and detach from the *screen* terminal. Scripts will take care of executing the *bandwidth shaping* at the Server side, and of producing a log file named according to the adaptation strategy and its parameters. By default, the three provided scripts reproduce all the parameter settings reported in the Calibration section, which could take time (10 minutes for each experiment, on average). In order to reduce the combinatory, respective variables can be directly modified from inside .sh scripts.

Results Processing

Post-processing can be done directly on the host machine by transferring the generated log files from the Client container into the *results* folder:

```
lxc file pull client/home/ubuntu/logFile .
```

where *logFile* must be changed with the actual name of the generated log file.
Scripts are also provided into the *results* folder; indeed, the processing can be launched by executing:

```
./Extract_KPIs.sh . logFile
```

where the two parameters are the *directory of the logFile* (i.e., '.') and the *logFile* name. This script will generate a folder correspondent to the emulated scenario, in which multiple files reporting different statistics will be created.
Notice that if emulating a scenario with the TCP/IP stack, it is required to execute an additional script:

```
./Extract_QualitySwitch_TCP.sh . logFile
```

which will elaborate only the quality switch aspects (e.g., number, frequency, etc.), and update the correspondent file in the folder created previously.
In the end, in order to have a synthetic representation of all the KPIs, the following script:

`./CollectSummary.sh . logFile`

will create a *summary.csv* file with all the metrics. It can be executed for all the emulated scenarios in order to have a single file with all the aggregated statistics.

Appendix D

Small guide to network-assisted DASH experiments

This small guide aims at giving you the instructions to reproduce the results of Chapter 4. The following will give you instructions to reproduce the extreme cases results (Section 1), the network assisted results (Section 2) and finally Section 3 gives more information on the video player we used and the cache.

Preambule

To get the scripts and the container image, you must download the tarball: `https://jaqu.eu/thesis/DATA/in-network-repro.tar.gz`. Once expanded, this tarball gives the container image along with the scripts and a copy of these instructions.

A working installation of vICN is needed, please follow instructions here: https://wiki.fd.io/view/Vicn

Once you have installed vICN, you need to import the container image to lxc. To do so, run the following command in your terminal:

$ lxc image import *container-image/ubuntu_1604_NA.tar.gz --alias ubuntu_1604_NA*

Note that *ubuntu_1604_NA* is the alias given to the LXC image. If you want to change it, your change should be reflected in the vICN topologies provided under the ./vicn folder. Once the import is done, you can check the presence of the image with the command lxc image list.

For the processing of the results, we use gnuplot to compute means and standard deviations, it should therefore be installed on your system:

```
$ sudo apt-get install gnuplot
```

D.1 Extreme cases

In this section, you will find the instructions to reproduce the results from Section 3 in our paper.

vICN deployment

First, spawn a screen for vICN:

```
$ screen -S vicn
```

Launch vICN:

```
$ sudo vicn -s vicn/wifi/topology.json
```

Once the deployment is done (the output is a bunch of INFO and/or WARNING logs from websocket), leave the screen (do not kill the vicn process). Modify the setup script, `vicn/wifi/setup.sh`, to set the right path to your video repository by setting the variable `PATH_TO_VIDEO_FILES`. Then, run the script:

```
$ ./vicn/wifi/setup.sh
```

Note that, in order to keep the size manageable, we did not include the video files in this folder. Once the setup script is run, it will make the folder indicated by the PATH_TO_VIDEO_FILES variable the root folder of our web server.

Experiments

Once the testbed is deployed and setup (see previous section), you can start experimenting. In the folder `scripts/extremes-cases`, you will find three scripts, corresponding to the three scenarios described in our paper, as to reproduce the results for the baseline scenario (`no-cache.sh`), the proactive placement scenario (`cache-no-replacement.sh`) and the reactive LRU cache scenario (`cache_LRU.sh`).

Results processing

Once the experiments are done, the results are uploaded in the `results/extreme-cases` folder, with one folder per scenario. The following assumes that every scenarios were run in the previous section. If this is not the case, please update the `process_data.sh` script to take into account only the scenarios that were performed. To process the resulting log files, go into this folder:

```
$ cd results/extreme-cases
```

Run the post-processing script:

```
$ ../post-processing/process_data.sh
```

Upon completion, it will have created a new folder `results` inside the `extreme-cases` folder. In this folder, you will find the aggregated metrics for the different scenarios. The per-client metrics are also available, in the scenario folder, you will find one folder per client with the associated metrics.

Please note that the post-processing scripts expect a certain file organisation, so if you change the way results are stored, keep in mind that your changes should reflected in the post-processing scripts.

Teardown

To tear down the topology, kill the vicn process:

```
$ sudo killall -9 vicn
```

And then, run the clean-up script:

```
$ ./vicn/wifi/clean.sh
```

D.2 Network assistance

In this section, you will find the instructions to reproduce the results from Section 4.2 in our paper.

vICN deployment

First, spawn a screen for vICN:

```
$ screen -S vicn
```

Launch vICN:

```
$ sudo vicn -s vicn/ethernet/topology.json
```

Once the deployment is done (the output is a bunch of INFO and/or WARNING logs from websocket), leave the screen (do not kill the vicn process). Modify the setup script, `vicn/ethernet/setup.sh`, to set the right path to your video repository by setting the variable `PATH_TO_VIDEO_FILES`. Then, run the script:

```
$ ./vicn/ethernet/setup.sh
```

Note that, in order to keep the size manageable, we did not include the video files in this folder. Once the setup script is run, it will make the folder indicated by the PATH_TO_VIDEO_FILES variable the root folder of our web server.

Experiments

Once the testbed is deployed and setup (see previous section), you can start experimenting. In the folder `scripts/network-assistance`, the script `full-script.sh` enables you to re-run the scenarios described in our paper.

Results processing

Once the experiments are done, the results are uploaded in the `results/network-assistance` folder, with one folder per scenario. The following assumes that every scenarios were run in the previous section. If this is not the case, please update the `process_data.sh` script to take into account only the scenarios that were performed. To process the resulting log files, go into this folder:

```
$ cd results/network-assistance
```

Run the post-processing script:

```
$ ../post-processing/process_data.sh
```

Upon completion, it will have created a new folder `results` inside the `network-assistance` folder. In this folder, you will find the aggregated metrics for the different scenarios. The per-client metrics are also available, in the scenario folder, you will find one folder per client with the associated metrics.

Please note that the post-processing scripts expect a certain file organisation, so if you change the way results are stored, keep in mind that your changes should reflected in the post-processing scripts.

Teardown

To tear down the topology, kill the vicn process:

```
$ sudo killall -9 vicn
```

And then, run the clean-up script:

```
$ ./vicn/ethernet/clean.sh
```

D.3 Extra information

Emergency button

The script `./scripts/tools/stop-all.sh` gives the possibility to abort an experiment by killing all the instances of the video player.

Changing the parameters of the video player

The video player we used is a modified version of Viper, available here: `https://wiki.fd.io/view/Viper`. The usage is as follow:

```
$ viper [-nohead] [-n] [adaptationLogic [adaptationParameters]] [-u url]
```

-nohead
: If present, the player will be in headless mode (it will download the MPD from the url specified by the **-u** option and returns once it has finished downloading the whole video)

-n
: If present with **-nohead**, the player will use the ICN stack. Otherwise, the TCP stack is used.

-u

If present, it must be followed by the URL of the MPD.

adaptationLogic

BBA

-b B_{Min} B_{Upper}

ADAPTECH

-br Alpha B_{Min} B_{Steady} $T_{Switch\text{-}up}$ *SlackParameter*; The *Alpha* parameter is the parameter for the EWMA calculating the average bandwidth.

STRICT

-strict enforcedQuality; The quality downloaded will be the one at index *enforcedQuality* (it should be an integer) in the MPD.

STEERED

-steered enforcedQuality B_{Panic}

ADAPTECH_NA

-brna Alpha B_{Min} B_{Steady} $T_{Switch\text{-}up}$ *SlackParameter* T_{Low} T_{High} $T_{Samples}$

Please note that all the buffer thresholds are expecting values as a percentage of the total buffer.

Using the metis forwarder

Please refer to https://wiki.fd.io/view/Sb-forwarder for a full overview of the metis forwarder.

In our scripts, the configuration of the cache is done using config files located in the folder scripts/*chosen-scenario*/conf. In these config files, the two parameters that can be modified are **CS_SIZE**, the size of the cache, and **TIME_INTERVAL**, the time elapsed (in seconds) between two cache advertisements.

Appendix E

Résumé de la thèse en français

Cette thèse étudie l'impact des réseaux centrés sur l'information (ICN – Information Centric Networking) sur la qualité d'expérience des utilisateurs dans le cadre de streaming multimedia, et plus particulièrement les cas de vidéos à la demande (VoD) et vidéos en temps réel. Cette thèse est divisée en 6 chapitres, structurés comme suit.

Le chapitre 1 introduit le contexte de cette thèse et les enjeux associés. En effet, le streaming vidéo, et plus généralement multimédia, a pris une place de choix dans l'utilisation d'internet au cours de ces dernières années, et tout porte à croire que son impact sur le réseau internet sera prépondérant dans les années à venir. Tout ceci met en lumière le décalage croissant entre l'architecture d'internet et son usage. D'une part, l'essentielle utilisation d'internet aujourd'hui (sans pour autant en faire la satire) réside dans la diffusion d'informations[1]: l'avènement de plateformes sociales, telles que Facebook, YouTube ou encore Netflix, pousse leur usagers à regarder et partager toujours plus de contenus, métamorphosant le réseau en instrument pour connecter les gens à du contenu. De l'autre coté, l'architecture d'internet repose toujours sur son principe fondateur de communication hôte à hôte, inadapté à la distribution de contenu multimédia.

Afin de palier ce décalage et d'anticiper la charge colossale qui sera imposée sur le réseau par le streaming multimédia, un nouveau champ de recherche est apparu, appelé *Future Internet*. Parmi les architectures proposées, les réseaux centrés sur l'information (ICN, pour *Information-Centric Networking*) offrent un nouveau paradigme de réseaux qui permet une prise en compte du contenu directement au niveau de la couche réseau, c'est à dire à l'échelle du paquet. De plus ICN présente une connivence certaine avec les techniques d'adaptation dynamique de streaming[2]

[1]Information désigne ici tout type de contenu, comme par exemple des vidéos, des fichiers audios, etc.

[2]Ce terme regroupe toute une variété de techniques apparues au cours des dernières années pour optimiser la distribution multimédia. Parmi elles se trouvent MPEG-DASH, Microsoft HSS,

(DAS, pour *Dynamic Adaptive Streaming*), ce qui en fait un bon candidat pour accompagner l'évolution de la distribution multimédia, en exploitant ses diverses caractéristiques.

Le chapitre 1 se termine par un énoncé des contributions de cette thèse, ainsi que la liste des publications associées.

Le chapitre 2 parcourt l'état de l'art de la distribution vidéo, en faisant la distinction entre la vidéo à la demande et la vidéo en temps réel. Notamment, ce chapitre s'attarde sur MPEG-DASH (Dynamic Adaptive Streaming over HTTP, ou, en français, streaming adaptatif dynamique via HTTP), un récent standard de la communauté MPEG. Très succinctement, MPEG-DASH fonctionne comme suit: chaque vidéo est divisée en segments de durée égale. Chaque segment est encodé à des qualités différentes (différents débits vidéos, différentes résolutions d'écran). Ainsi, télécharger séquentiellement les segments constituant une vidéo permet de la reconstituer. Chaque segment peut être téléchargé à une qualité indépendante les unes des autres. Un algorithme, appelé la logique d'adaptation, est responsable du choix de la qualité à laquelle un segment va être téléchargé, basé sur des paramètres mesurés au niveau du client (comme par exemple son débit disponible, la quantité de vidéo présente en mémoire tampon, etc...). Ces logiques d'adaptation ont fait l'objet de nombreux travaux ces dernières années, et ce chapitre conduit un état de l'art des logiques existantes.

En outre, le chapitre 2 présente ICN et ces différentes caractéristiques en détails. Très brièvement, avec ICN, chaque contenu (vidéo, fichier, etc...) est divisé en fragments, appelés paquets *Data*, et chaque paquet *Data* est identifié par un *data name*. Pour récupérer un paquet *Data*, un utilisateur envoie un paquet *Interest*, avec le *data name* du paquet *Data* demandé. Pour ce faire, un routeur ICN se repose sur trois structures de données: la **FIB** (*Forwarding Information Base*, ou Base d'Informations de Retransmission), la **PIT** (*Pending Interest Table*, ou Table des *Interests* en Attente) et le **CS** (*Content Store*, ou magasin de contenus). La FIB contient les informations nécessaires pour transmettre un paquet *Interest*; le CS agit comme un cache et permet à un routeur ICN de stocker des paquets *Data* pour pouvoir servir plus vite certains paquets *Interest*; enfin, la PIT permet au routeur ICN de transmettre un paquet *Data*.

Le chapitre 3 expose les bénéfices qui sont apportés par ICN dans la distribution vidéo, et en particulier dans la VoD. Pour se faire, nous comparons ainsi les deux piles réseau (ICN et IP) dans le cas de la VoD. Afin de couvrir tout le spectre des logiques d'adaptation, nous en sélectionnons trois parmi les logiques existantes dans l'état de l'art, que nous estimons représentatives des différentes fa-

Apple HLS et Adobe HDS.

milles d'algorithmes. Une calibration de chaque algorithme sélectionné a été faite, afin d'optimiser la meilleure qualité d'expérience (QoE) possible pour chaque algorithme. Cette calibration a été effectuée sur une pile réseau TCP/IP et une pile réseau NDN simple (c'est-à-dire sans aucune des améliorations proposées ultérieurement), et parmi les différentes métriques existantes pour estimer la QoE, nous en avons choisi six: la qualité moyenne de la vidéo, le nombre de changements de qualité, la fréquence moyenne de changement de qualité, la moyenne de l'amplitude des variations de qualité, le nombre d'évènements *rebuffering*[3] et le temps total passé en *rebuffering*. L'appendice C présente des instructions pour reproduire nos expérience de calibration.

Une fois les algorithmes calibrés, nous avons pu comparer la qualité d'expérience que les clients pouvaient espérer lors d'expériences de streaming multimédia via les deux piles réseau dans différents scénarios, faisant varier le nombre de clients (un seul client ou plusieurs), le type de réseau d'accès (Wi-Fi, LTE, traces 3G/4G), le nombre de réseaux d'accès (un seul accès ou un scénario multi-homed, où un client possède au moins deux accès – Wi-Fi et LTE), ou encore certaines fonctionnalités de la pile réseau ICN. Nous montrons ainsi qu'une pile réseau ICN apporte des bénéfices quantitatifs en termes de qualité d'expérience aux clients dans le cas de streaming multimedia VoD, grâce à une estimation plus précise de la bande passante disponible pour le client, ainsi qu'à un mécanisme de détection et de recouvrement des pertes réseaux (appelé WLDR [28]), ou encore à un balancement de la charge au niveau du client dans le cas d'un scénario multi-homed. Nous montrons aussi que ces fonctionnalités permettent d'obtenir la meilleure qualité d'expérience possible lorsqu'elles sont utilisés conjointement. Il est à noter cependant que cette comparaison des deux piles réseau se concentre sur le client, et ne prend pas en compte les interactions possibles avec le réseau.

Le chapitre 4 propose une approche du streaming multimédia VoD assisté par le réseau, en se focalisant sur une fonctionnalité précise d'ICN: le caching réseau. Tout d'abord, nous explorons le champ des possibles dans l'espace des interactions client-réseau, en présentant les avantages qu'un client peut tirer d'une interaction active avec le réseau (par exemple, une augmentation de la qualité moyenne de la vidéo), ainsi que les désavantages qui en découlent (par exemple, le phénomène des oscillations de qualité vidéo demandée induites par le cache). Nous montrons notamment que l'interaction client-réseau ne doit pas être faite de manière naïve, afin d'améliorer la qualité d'expérience du client.

Nous proposons ensuite dans une deuxième partie du chapitre 4 une évolution consciente du réseau d'une logique d'adaptation existante: AdapTech [15]. Notre

[3]Un évènement de rebuffering est un instant où le décodeur de vidéo n'a plus de segment vidéo à décoder et donc force un arrêt temporaire de la vidéo au client

proposition repose, d'une part, sur un signal du cache vers le client, lui donnant le nombre de *cache-hit* par qualité vidéo; et d'autre part, sur un algorithme permettant à la logique d'adaptation de prendre en compte ce signal pour faire un choix de qualité vidéo. Sans entrer dans les détails, notre algorithme garde deux estimations de débit: une vers le cache, et une vers le serveur source de la vidéo. En fonction du signal émis par le cache, le client choisit d'utiliser l'une ou l'autre des estimations: si le *cache-hit* est élevé pour une certaine qualité, il y a des chances que cette dernière se trouve en cache, ainsi le client préférera l'estimation de débit vers le cache, et si, au contraire, le *cache-hit* est bas pour une qualité, le client préférera l'estimation de débit vers le serveur source. Une campagne expérimentale vient confirmer notre approche et montre une performance accrue en comparaison à une approche agnostique au réseau, où les décisions sont prises au niveau du client uniquement.

Le chapitre 5 s'intéresse en particulier au cas du streaming vidéo en temps réel. En particulier, alors que les chapitres 3 et 4 sont centrés sur le streaming vidéo VoD et sur MPEG-DASH, ce standard, de part la granularité des segments vidéo (de l'orde de quelques secondes), introduit des délais tels qu'il ne peut pas être utilisé dans le cas du streaming vidéo en temps réel, où des temps de latence très faibles sont attendus pour la distribution multimédia et pour l'interaction des clients, comme par exemple avec les jeux en ligne, les enchères en ligne ou plus simplement les visioconférences. Dans de tels cas, *WebRTC* (Web Real-Time Communication) semble être une solution prometteuse. Ainsi, le chapitre 5 propose et explore une intégration d'ICN dans WebRTC, et en particulier les bénéfices sur le passage à l'échelle qui peut en résulter.

Nous concevons et implémentons un système WebRTC sur *hICN* (pour hybrid ICN, une solution ICN-dans-IP incrémentalement deployable), ainsi qu'un protocole de transport ICN qui minimise la latence, appelé RICTP (pour *Realtime Information Centric Transport Protocol*). Une première campagne expérimentale montre qu'ICN et WebRTC ensemble donne une architecture qui passe l'échelle au niveau du contenu (c'est-à-dire le nombre de flux media actifs dans une visioconférence) plutôt qu'au niveau du nombre de participants, au contraire de WebRTC sur IP.

Finalement, le chapitre 6 vient conclure cette thèse.

Bibliography

[1] CCN/NDN Convergence Effort at IRTF ICN Research Group. https://trac.ietf.org/trac/irtf/wiki/icnrg/convergence.

[2] CICN project, wiki page. https://wiki.fd.io/view/Cicn.

[3] DASH Industry Forum: DASH-AVC/264 Test Cases and Vectors. http://dashif.org/wp-content/uploads/2015/04/DASH-AVC-264-Test-Vectors-v09-CommunityReview.pdf.

[4] DASH Industry Forum, GitHub page. https://github.com/Dash-Industry-Forum/dash.js.

[5] Jitsi. https://jitsi.org/.

[6] Jitsi meet torture. https://github.com/jitsi/jitsi-meet-torture/.

[7] Linux Containers. https://linuxcontainers.org.

[8] NDN Repo, GitHub page. https://github.com/named-data/repo-ng.

[9] SeleniumHQ. https://docs.seleniumhq.org.

[10] vICN, wiki page. https://wiki.fd.io/view/Vicn.

[11] Viper, wiki page. https://wiki.fd.io/view/Viper.

[12] Adobe. HTTP Dynamic Streaming. https://www.adobe.com/products/hds-dynamic-streaming.html.

[13] Saamer Akhshabi, Lakshmi Anantakrishnan, Ali C Begen, and Constantine Dovrolis. What Happens when HTTP Adaptive Streaming Players Compete for Bandwidth? In *Proceedings of the 22nd international workshop on Network and Operating System Support for Digital Audio and Video*, pages 9–14. ACM, 2012.

[14] Saamer Akhshabi, Lakshmi Anantakrishnan, Constantine Dovrolis, and Ali C Begen. Server-based Traffic Shaping for Stabilizing Oscillating Adaptive Streaming Players. In *Proceeding of the 23rd ACM workshop on network and operating systems support for digital audio and video*, pages 19–24. ACM, 2013.

[15] Saamer Akhshabi, Sethumadhavan Narayanaswamy, Ali C Begen, and Constantine Dovrolis. An Experimental Evaluation of Rate-Adaptive Video Players over HTTP. *Signal Processing: Image Communication*, 27(4):271–287, 2012.

[16] Claudio Alberti, Daniele Renzi, Christian Timmerer, Christopher Mueller, Stefan Lederer, Stefano Battista, and Marco Mattavelli. Automated QoE Evaluation of Dynamic Adaptive Streaming over HTTP. In *Quality of Multimedia Experience (QoMEX), 2013 Fifth International Workshop on*, pages 58–63. Ieee, 2013.

[17] ATIS 5G Americas. Understanding Information Centric Networking and Mobile Edge Computing. `http://www.5gamericas.org/files/3414/8173/2353/Understanding_Information_Centric_Networking_and_Mobile_Edge_Computing.pdf`, Dec 2016.

[18] Jordan Augé, Giovanna Carofiglio, Giulio Grassi, Luca Muscariello, Giovanni Pau, and Xuan Zeng. MAP-Me: Managing Anchor-less Producer Mobility in Content-Centric Networks. *IEEE Transactions on Network and Service Management*, 2018.

[19] A. Bentaleb, A.C. Begen, and R. Zimmermann. SDNDASH: Improving QoE of HTTP Adaptive Streaming Using Software Defined Networking. In *ACM Multimedia (MM'16)*, MM '16, pages 1296–1305, New York, NY, USA, 2016. ACM.

[20] D. Bhat, A. Rizk, M. Zink, and R. Steinmetz. Network Assisted Content Distribution for Adaptive Bitrate Video Streaming. In *ACM on Multimedia Systems (MMSys'17)*, 2017.

[21] Divyashri Bhat, Cong Wang, Amr Rizk, and Michael Zink. A Load Balancing Approach for Adaptive Bitrate Streaming in Information Centric Networks. In *Multimedia & Expo Workshops (ICMEW), 2015 IEEE International Conference on*, pages 1–6. IEEE, 2015.

[22] Christos Bouras, Georgios Kioumourtzis, and Apostolos Gkamas. Simulcast Transmission for Video Applications: Performance Evaluation with an Integrated Simulation Environment. In *Performance Evaluation of Computer &*

Telecommunication Systems, 2009. SPECTS 2009. International Symposium on, volume 41, pages 339–346. IEEE, 2009.

[23] Niels Bouten, Steven Latré, Jeroen Famaey, Werner Van Leekwijck, and Filip De Turck. In-Network Quality Optimization for Adaptive Video Streaming Services. *IEEE Transactions on Multimedia*, 16(8):2281–2293, 2014.

[24] Nassima Bouzakaria, Cyril Concolato, and Jean Le Feuvre. Overhead and Performance of Low Latency Live Streaming using MPEG-DASH. In *Information, Intelligence, Systems and Applications, IISA 2014, The 5th International Conference on*, pages 92–97. IEEE, 2014.

[25] G. Carofiglio, M. Gallo, L. Muscariello, M. Papalini, and Sen Wang. Optimal Multipath Congestion Control and Request Forwarding in Information-Centric Networks. In *2013 21st IEEE International Conference on Network Protocols (ICNP)*, pages 1–10, Oct 2013.

[26] Giovanna Carofiglio, Jordan Augé, Luca Muscariello, Michele Papalini, and Jacques Samain. Mobile Video Delivery with Hybrid ICN. Cisco, White Paper, 2016.

[27] Giovanna Carofiglio, Jordan Augé, Luca Muscariello, Michele Papalini, Jacques Samain, and Mauro Sardara. Using ICN to Simplify Data Delivery, Mobility Management and Secure Transmission over an Heterogeneous Network Access. In *FG IMT-2020 Workshop and Demo Day: Wireline Technology Enablers for 5G*. ITU, December 2016.

[28] Giovanna Carofiglio, Luca Muscariello, Michele Papalini, Natalya Rozhnova, and Xuan Zeng. Leveraging ICN In-network Control for Loss Detection and Recovery in Wireless Mobile Networks. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking*, pages 50–59. ACM, 2016.

[29] Asit Chakraborti, Syed Obaid Amin, Bin Zhao, Aytac Azgin, Ravishankar Ravindran, and Guoqiang Wang. ICN Based Scalable Audio-Video Conferencing on Virtualized Service Edge Router (VSER) Platform. In *Proceedings of the 2Nd ACM Conference on Information-Centric Networking*, ACM-ICN '15. ACM, 2015.

[30] Giuseppe Cofano, Luca De Cicco, Thomas Zinner, Anh Nguyen-Ngoc, Phuoc Tran-Gia, and Saverio Mascolo. Design and Experimental Evaluation of Network-Assisted Strategies for HTTP Adaptive Streaming. In *Proceedings of the 7th International Conference on Multimedia Systems*, page 3. ACM, 2016.

[31] Bram Cohen. The BitTorrent Protocol Specification, 2008.

[32] Xavier Corbillon, Ramon Aparicio-Pardo, Nicolas Kuhn, Géraldine Texier, and Gwendal Simon. Cross-layer Scheduler for Video Streaming over MPTCP. In *Proceedings of the 7th International Conference on Multimedia Systems*, page 7. ACM, 2016.

[33] S. D'Aronco, L. Toni, and P. Frossard. Price-Based Controller for Utility-Aware HTTP Adaptive Streaming. *IEEE MultiMedia*, 24(2):20–29, Apr 2017.

[34] Luca De Cicco, Vito Caldaralo, Vittorio Palmisano, and Saverio Mascolo. Elastic: a Client-side Controller for Dynamic Adaptive Streaming over HTTP (DASH). In *Packet Video Workshop (PV), 2013 20th International*, pages 1–8. IEEE, 2013.

[35] Vladimir Dimitrov and Ventzislav Koptchev. PSIRP Project–Publish-Subscribe Internet Routing Paradigm: New Ideas for Future Internet. In *Proceedings of the 11th International Conference on Computer Systems and Technologies and Workshop for PhD Students in Computing on International Conference on Computer Systems and Technologies*, pages 167–171. ACM, 2010.

[36] F. Dobrian, A. Awan, D. Joseph, A. Ganjam, J. Zhan, V. Sekar, I. Stoica, and H. Zhang. Understanding the Impact of Video Quality on User Engagement. *Commun. ACM*, 56(3):91–99, Mar 2013.

[37] Florin Dobrian, Vyas Sekar, Asad Awan, Ion Stoica, Dilip Joseph, Aditya Ganjam, Jibin Zhan, and Hui Zhang. Understanding the Impact of Video Quality on User Engagement. In *ACM SIGCOMM Computer Communication Review*, volume 41, pages 362–373. ACM, 2011.

[38] Ali El Essaili, Damien Schroeder, Dirk Staehle, Mohammed Shehada, Wolfgang Kellerer, and Eckehard Steinbach. Quality-of-Experience driven Adaptive HTTP Media Delivery. In *Communications (ICC), 2013 IEEE International Conference on*, pages 2480–2485. IEEE, 2013.

[39] Ali El Essaili, Damien Schroeder, Eckehard Steinbach, Dirk Staehle, and Mohammed Shehada. QoE-based Traffic and Resource Management for Adaptive HTTP Video Delivery in LTE. *IEEE Transactions on Circuits and Systems for Video Technology*, 25(6):988–1001, 2015.

[40] Ian Fette and Alexey Melnikov. The Websocket Protocol. RFC 6455, December 2011.

[41] Roy Fielding, Jim Gettys, Jeffrey Mogul, Henrik Frystyk, Larry Masinter, Paul Leach, and Tim Berners-Lee. Hypertext Transfer Protocol–HTTP/1.1. RFC 2616, 1999.

[42] The Linux Fundation. Linux Fundation Wiki - Netem. `https://wiki.linuxfoundation.org/networking/netem`.

[43] A. Ganjam, F. Siddiqui, J. Zhan, X. Liu, I. Stoica, J. Jiang, V. Sekar, and H. Zhang. C3: Internet-Scale Control Plane for Video Quality Optimization. In *USENIX NSDI*, 2015.

[44] Panagiotis Georgopoulos, Yehia Elkhatib, Matthew Broadbent, Mu Mu, and Nicholas Race. Towards Network-wide QoE Fairness using Openflow-assisted Adaptive Video Streaming. In *Proceedings of the 2013 ACM SIGCOMM workshop on Future human-centric multimedia networking*, pages 15–20. ACM, 2013.

[45] Cesar Ghali, Gene Tsudik, and Ersin Uzun. Network-Layer Trust in Named-Data Networking. *ACM SIGCOMM Computer Communication Review*, 44(5):12–19, 2014.

[46] Google Git. WebRTC library. `https://webrtc.googlesource.com/`.

[47] E. Goldoni, G. Rossi, and A. Torelli. Assolo, a New Method for Available Bandwidth Estimation. In *ICIMP*, 2009.

[48] Reinhard Grandl, Kai Su, and Cedric Westphal. On the Interaction of Adaptive Video Streaming with Content-Centric Networking. In *Packet Video Workshop (PV), 2013 20th International*, pages 1–8. IEEE, 2013.

[49] Boris Grozev, Lyubomir Marinov, Varun Singh, and Emil Ivov. Last N: Relevance-based Selectivity for Forwarding Video in Multimedia Conferences. In *Proceedings of the 25th ACM Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '15. ACM, 2015.

[50] Boris Grozev, George Politis, Emil Ivov, Thomas Noël, and Varun Singh. Experimental Evaluation of Simulcast for WebRTC. *IEEE Communications Standards Magazine*, 2017.

[51] C.T. Guguen, F. Le Bolzer, and R. Houdaille. Improving User Experience when HTTP Adaptive Streaming Clients Compete for Bandwidth. *SMPTE Motion Imaging Journal*, 126(1):28–34, 2017.

[52] Peter Gusev and Jeff Burke. NDN-RTC: Real-Time Videoconferencing over Named Data Networking. In *Proceedings of the 2Nd ACM Conference on Information-Centric Networking*, ACM-ICN '15. ACM, 2015.

[53] K.H. Kim H. Nam and H. Schulzrinne. QoE Matters More Than QoS: Why People Stop Watching Cat Videos. In *Proc. of IEEE INFOCOM*, Apr. 2016.

[54] Ryan Hamilton, Janardhan Iyengar, Ian Swett, Alyssa Wilk, et al. QUIC: A UDP-based secure and reliable transport for HTTP/2. *IETF, draft-tsvwg-quic-protocol-02*, 2016.

[55] S Holmer, H Lundin, G Carlucci, LD Cicco, and S Mascolo. A google Congestion Control Algorithm for Real-Time Communication. *draft-ietf-rmcat-gcc-01 (work in progress)*, 2015.

[56] Tobias Hoßfeld, Michael Seufert, Matthias Hirth, Thomas Zinner, Phuoc Tran-Gia, and Raimund Schatz. Quantification of YouTube QoE via Crowdsourcing. In *Multimedia (ISM), 2011 IEEE International Symposium on*, pages 494–499. IEEE, 2011.

[57] Patrice Houzé, Emmanuel Mory, Géraldine Texier, and Gwendal Simon. Applicative-layer Multipath for Low-Latency Adaptive Live Streaming. In *Communications (ICC), 2016 IEEE International Conference on*, pages 1–7. IEEE, 2016.

[58] Te-Yuan Huang, Nikhil Handigol, Brandon Heller, Nick McKeown, and Ramesh Johari. Confused, Timid, and Unstable: Picking a Video Streaming Rate is Hard. In *Proceedings of the 2012 Internet Measurement Conference*, pages 225–238. ACM, 2012.

[59] Te-Yuan Huang, Ramesh Johari, Nick McKeown, Matthew Trunnell, and Mark Watson. A Buffer-based Approach to Rate Adaptation: Evidence from a Large Video Streaming Service. *ACM SIGCOMM Computer Communication Review*, 44(4):187–198, 2015.

[60] E. Ivov, E. Marocco, and J. Lennox. A Real-time Transport Protocol (RTP) Header Extension for Mixer-to-Client Audio Level Indication. RFC 6465.

[61] Van Jacobson, Diana K. Smetters, Nicholas H. Briggs, Michael F. Plass, Paul Stewart, James D. Thornton, and Rebecca L. Braynard. VoCCN: Voice-over Content-centric Networks. In *Proceedings of the 2009 Workshop on Re-architecting the Internet*, ReArch '09. ACM, 2009.

[62] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking Named Content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, 2009.

[63] Rajendra K Jain, Dah-Ming W Chiu, and William R Hawe. A Quantitative Measure of Fairness and Discrimination. *Eastern Research Laboratory, Digital Equipment Corporation: Hudson, MA, USA*, pages 2–7, 1984.

[64] Cyriac James, Emir Halepovic, Mea Wang, Rittwik Jana, and NK Shankaranarayanan. Is Multipath TCP (MPTCP) beneficial for Video Streaming over DASH? In *2016 IEEE 24th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems (MASCOTS)*, pages 331–336. IEEE, 2016.

[65] A. Jangam, R. Ravindran, A. Chakraborti, X. Wan, and G. Wang. Realtime Multi-Party Video Conferencing Service over Information Centric Network. In *2015 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)*, June 2015.

[66] J. Jiang, V. Sekar, and H. Zhang. Improving Fairness, Efficiency, and Stability in HTTP-based Adaptive Video Streaming with FESTIVE. In *Proc. of ACM CoNEXT*, pages 97–108, Dec. 2012.

[67] Rihab Jmal, Gwendal Simon, and Lamia Chaari. Network-Assisted Strategy for DASH over CCN. In *Multimedia and Expo (ICME), 2017 IEEE International Conference on*, pages 13–18. IEEE, 2017.

[68] Ingemar Johansson. Self-Clocked Rate Adaptation for Conversational Video in LTE. In *Proceedings of the 2014 ACM SIGCOMM workshop on Capacity sharing workshop*, pages 51–56. ACM, 2014.

[69] Parikshit Juluri and Deep Medhi. Cache'n DASH: Efficient Caching for DASH. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 599–600. ACM, 2015.

[70] Rohit Kapoor, Ling-Jyh Chen, Li Lao, Mario Gerla, and M Young Sanadidi. CapProbe: A simple and Accurate Capacity Estimation Technique. In *ACM SIGCOMM Computer Communication Review*, volume 34, pages 67–78. ACM, 2004.

[71] Theodoros Karagkioules, Cyril Concolato, Dimitrios Tsilimantos, and Stefan Valentin. A Comparative Case Study of HTTP Adaptive Streaming Algorithms in Mobile Networks. In *Proceedings of the 27th Workshop on Net-*

work and Operating Systems Support for Digital Audio and Video, pages 1–6. ACM, 2017.

[72] J.W. Kleinrouweler, S. Cabrero, and P. Cesar. Delivering Stable High-Quality Video: an SDN Architecture with DASH Assisting Network Elements. In *Proc. of ACM MMSys*, May 2016.

[73] J.W. Kleinrouweler, B. Meixner, and P. Cesar. Improving Video Quality in Crowded Networks Using a DANE. In *Proceedings of the 27th Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV'17, pages 73–78, New York, NY, USA, 2017. ACM.

[74] Nikolaos Laoutaris, Hao Che, and Ioannis Stavrakakis. The LCD Interconnection of LRU Caches and its Analysis. *Performance Evaluation*, 63(7):609–634, 2006.

[75] Stefan Lederer, Christopher Mueller, Benjamin Rainer, Christian Timmerer, and Hermann Hellwagner. Adaptive Streaming over Content Centric Networks in Mobile Networks using Multiple Links. In *Communications Workshops (ICC), 2013 IEEE International Conference on*, pages 677–681. IEEE, 2013.

[76] Stefan Lederer, Christopher Mueller, Christian Timmerer, and Hermann Hellwagner. Adaptive Multimedia Streaming in Information-Centric Networks. *IEEE Network*, 28(6):91–96, 2014.

[77] Stefan Lederer, Christopher Müller, and Christian Timmerer. Dynamic Adaptive Streaming over HTTP Dataset. In *Proceedings of the 3rd Multimedia Systems Conference*, pages 89–94. ACM, 2012.

[78] Danny H Lee, Constantine Dovrolis, and Ali C Begen. Caching in HTTP Adaptive Streaming: Friend or Foe? In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, page 31. ACM, 2014.

[79] W. Li, S. M. A. Oteafy, and H. S. Hassanein. Dynamic adaptive streaming over popularity-driven caching in Information-Centric Networks. In *IEEE ICC*, 2015.

[80] Wenjie Li, Sharief MA Oteafy, and Hossam S Hassanein. Dynamic Adaptive Streaming over Popularity-driven Caching in Information-Centric Networks. In *Communications (ICC), 2015 IEEE International Conference on*, pages 5747–5752. IEEE, 2015.

[81] Wenjie Li, Sharief MA Oteafy, and Hossam S Hassanein. StreamCache: Popularity-based Caching for Adaptive Streaming over Information-Centric Networks. In *Communications (ICC), 2016 IEEE International Conference on*, pages 1–6. IEEE, 2016.

[82] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C Begen, and David Oran. Probe and Adapt: Rate Adaptation for HTTP Video Streaming At Scale. Jul. 2013. arxiv:1305.0510v2.

[83] Zhi Li, Xiaoqing Zhu, Joshua Gahm, Rong Pan, Hao Hu, Ali C Begen, and David Oran. Probe and Adapt: Rate Adaptation for HTTP Video Streaming at Scale. *IEEE Journal on Selected Areas in Communications*, 32(4):719–733, 2014.

[84] Linux Foundation FD.io. White Paper - Vector Packet Processing - One Terabit Software Router on Intel Xeon Scalable Processor Family Server. `https://fd.io`.

[85] X. Liu, F. Dobrian, H. Milner, J. Jiang, V. Sekar, I. Stoica, and H. Zhang. A Case for a Coordinated Internet Video Control Plane. In *Proceedings of the ACM SIGCOMM 2012 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, SIGCOMM '12, pages 359–370, New York, NY, USA, 2012. ACM.

[86] Yaning Liu, Joost Geurts, Jean-Charles Point, Stefan Lederer, Benjamin Rainer, Christopher Muller, Christian Timmerer, and Hermann Hellwagner. Dynamic Adaptive Streaming over CCN: A Caching and Overhead Analysis. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3629–3633. IEEE, 2013.

[87] Thorsten Lohmar, Torbjorn Einarsson, Per Frojdh, Frédéric Gabin, and Markus Kampmann. Dynamic Adaptive HTTP Streaming of Live Content. 2011.

[88] Rohan Mahy, Philip Matthews, and Jonathan Rosenberg. Traversal using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN). Technical report, 2010.

[89] Muhammad Faran Majeed, Syed Hassan Ahmed, Siraj Muhammad, Houbing Song, and Danda B Rawat. Multimedia Streaming in Information-Centric Networking: A Survey and Future Perspectives. *Computer Networks*, 125:103–121, 2017.

[90] Hongzi Mao, Ravi Netravali, and Mohammad Alizadeh. Neural Adaptive Video Streaming with Pensieve. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication*, pages 197–210. ACM, 2017.

[91] Konstantin Miller, Emanuele Quacchio, Gianluca Gennari, and Adam Wolisz. Adaptation Algorithm for Adaptive Streaming over HTTP. In *Packet Video Workshop (PV), 2012 19th International*, pages 173–178. IEEE, 2012.

[92] Zhongxing Ming, Mingwei Xu, and Dan Wang. Age-based Cooperative Caching in Information-Centric Networking. In *Computer Communication and Networks (ICCCN), 2014 23rd International Conference on*, pages 1–8. IEEE, 2014.

[93] Ricky KP Mok, Xiapu Luo, Edmond WW Chan, and Rocky KC Chang. QDASH: a QoE-Aware DASH System. In *Proceedings of the 3rd Multimedia Systems Conference*, pages 11–22. ACM, 2012.

[94] M Mosko, I Solis, and C Wood. CCNx Messages in TLV Format. `https://datatracker.ietf.org/doc/draft-irtf-icnrg-ccnxsemantics/`.

[95] Marc Mosko, Ignacio Solis, and Christopher A. Wood. CCNx Semantics. Internet-Draft draft-irtf-icnrg-ccnxsemantics-09, Internet Engineering Task Force, June 2018.

[96] M. K. Mukerjee, D. Naylor, J. Jiang, D. Han, S. Seshan, and H. Zhang. Practical, Real-time Centralized Control for CDN-based Live Video Delivery. In *ACM SIGCOMM*, 2015.

[97] Luca Muscariello, Giovanna Carofiglio, Jordan Auge, and Michele Papalini. Hybrid Information-Centric Networking. Internet-Draft draft-muscariello-intarea-hicn-00, Internet Engineering Task Force, June 2018.

[98] H. Nam, K. H. Kim, J. Y. Kim, and H. Schulzrinne. Towards QoE-aware video streaming using SDN. In *IEEE GLOBECOM*, 2014.

[99] Jiri Navratil and R. Les Cottrell. ABwE: A Practical Approach to Available Bandwidth Estimation. 2003.

[100] J. Ott, S. Wenger, N. Sato, C. Burmeister, and J. Rey. Extended RTP Profile for Real-time Transport Control Protocol (RTCP)-Based Feedback (RTP/AVPF). RFC 4585, 2006.

[101] Roger Pantos and William May. HTTP Live Streaming. RFC 8216, 2017.

[102] P. Papageorge, J. McCann, and M. Hicks. Passive Aggressive Measurement with MGRP. *SIGCOMM Comput. Commun. Rev.*, 39(4):279–290, Aug. 2009.

[103] Michele Papalini, Giovanna Carofiglio, Alberto Compagno, Angelo Mantellini, Luca Muscariello, Jacques Samain, and Mauro Sardara. Scaling WebRTC using Hybrid Information-Centric Networking (under submission). In *ACM ICN*, 2019.

[104] Cisco White Paper. Cisco Visual Networking Index: Forecast and Trends, 2017-2022. `https://www.cisco.com/c/en/us/solutions/service-provider/visual-networking-index-vni/index.html`, November 2018.

[105] H. Parmar and M. Thornburgh. Adobe's Real Time Messaging Protocol. *Copyright Adobe Systems Incorporated*, pages 1–52, 2012.

[106] S. Petrangeli, D. Pauwels, J. van der Hooft, J. Slowack, T. Wauters, and F. De Turck. Dynamic video bitrate adaptation for WebRTC-based remote teaching applications. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium*, April 2018.

[107] Stefano Petrangeli, Niels Bouten, Maxim Claeys, and Filip De Turck. Towards SVC-based Adaptive Streaming in Information Centric Networks. In *Multimedia & Expo Workshops (ICMEW), 2015 IEEE International Conference on*, pages 1–6. IEEE, 2015.

[108] Stefano Petrangeli, Dries Pauwels, Jeroen van der Hooft, Tim Wauters, Filip De Turck, and Jürgen Slowack. Improving Quality and Scalability of webRTC Video Collaboration Applications. In *Proceedings of the 9th ACM Multimedia Systems Conference*, MMSys '18. ACM, 2018.

[109] Stefano Petrangeli, Tim Wauters, Rafael Huysegems, Tom Bostoen, and Filip De Turck. Network-based Dynamic Prioritization of HTTP Adaptive Streams to Avoid Video Freezes. In *Integrated Network Management (IM), 2015 IFIP/IEEE International Symposium on*, pages 1242–1248. IEEE, 2015.

[110] Vitalii Poliakov, Lucile Sassatelli, and Damien Saucez. Case for caching and Model Predictive Control quality decision algorithm for HTTP Adaptive Streaming: is cache-awareness actually needed? In *Globecom Workshops (GC Wkshps), 2016 IEEE*, pages 1–6. IEEE, 2016.

[111] Daniel Posch, Christian Kreuzberger, Benjamin Rainer, and Hermann Hellwagner. Using In-Network Adaptation to Tackle Inefficiencies Caused by DASH in Information-Centric Networks. In *Proceedings of the 2014 Workshop on Design, Quality and Deployment of Adaptive Video Streaming*, pages 25–30. ACM, 2014.

[112] Jon Postel and Joyce Reynolds. File Transfer Protocol. RFC 959, 1985.

[113] Konstantinos Poularakis, George Iosifidis, Antonios Argyriou, Iordanis Koutsopoulos, and Leandros Tassiulas. Caching and Operator Cooperation Policies for Layered Video Content Delivery. In *INFOCOM 2016-The 35th Annual IEEE International Conference on Computer Communications, IEEE*, pages 1–9. IEEE, 2016.

[114] Ioannis Psaras, Wei Koong Chai, and George Pavlou. Probabilistic In-Network Caching for Information-Centric Networks. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pages 55–60. ACM, 2012.

[115] B. Rainer, D. Posch, and H. Hellwagner. Investigating the Performance of Pull-based Dynamic Adaptive Streaming in NDN. *Journal on Selected Areas in Communications*, 34(8):2130–2140, May 2016.

[116] Vinay Joseph Ribeiro, Rudolf H Riedi, Richard G Baraniuk, Jiri Navratil, and Les Cottrell. Pathchirp: Efficient Available Bandwidth Estimation for Network Paths. In *Passive and active measurement workshop*, 2003.

[117] Haakon Riiser, Tore Endestad, Paul Vigmostad, Carsten Griwodz, and Pâl Halvorsen. Video Streaming using a Location-based Bandwidth-Lookup Service for Bitrate Planning. *ACM Transactions on Multimedia Computing, Communications, and Applications (TOMM)*, 8(3):24, 2012.

[118] Pedro Rodríguez, Álvaro Alonso, Joaquín Salvachúa, and Javier Cerviño. Materialising a New Architecture for a Distributed MCU in the Cloud. *Comput. Stand. Interfaces*, 2016.

[119] Jonathan Rosenberg. Interactive connectivity establishment (ICE): A protocol for network address translator (NAT) traversal for offer/answer protocols. RFC 5245, 2010.

[120] Jonathan Rosenberg, Rohan Mahy, Philip Matthews, and Dan Wing. Session traversal utilities for NAT (STUN). Technical report, 2008.

[121] Jacques Samain, Jordan Augé, Giovanna Carofiglio, Luca Muscariello, Michele Papalini, and Mauro Sardara. Enhancing Mobile Video Delivery over an Heterogeneous Network Access with Information-Centric Networking. In *Proceedings of the SIGCOMM Posters and Demos*, pages 22–24. ACM, 2017.

[122] Jacques Samain, Giovanna Carofiglio, Luca Muscariello, Michele Papalini, Mauro Sardara, Michele Tortelli, and Dario Rossi. Dynamic Adaptive Video Streaming: Towards a Systematic Comparison of ICN and TCP/IP. *IEEE Transactions on Multimedia*, 19(10):2166–2181, 2017.

[123] Jacques Samain, Giovanna Carofiglio, Michele Tortelli, and Dario Rossi. A Simple yet Effective Network-Assisted Signal for Enhanced DASH Quality of Experience. In *Proceedings of the 28th ACM SIGMM Workshop on Network and Operating Systems Support for Digital Audio and Video*, pages 55–60. ACM, 2018.

[124] M. Sardara, L. Muscariello, and A. Compagno. A Transport Layer and Socket API for (h)ICN: Design, Implementation and Performance Analysis. In *Proc. of ACM SIGCOMM ICN '18*, 2018.

[125] Mauro Sardara, Luca Muscariello, Jordan Augé, Marcel Enguehard, Alberto Compagno, and Giovanna Carofiglio. Virtualized ICN (vICN): Towards a Unified Network Virtualization Framework for ICN Experimentation. *Proc. ACM ICN*, 2017.

[126] Henning Schulzrinne, Stephen Casner, Ron Frederick, and Van Jacobson. RTP: A transport protocol for real-time applications. RFC 3550, 2003.

[127] S. Schwarzmann, T. Zinner, and O. Dobrijevic. Towards a Framework for Comparing Application-Network Interaction Mechanisms. In *Proc. of IEEE ITC 28*, Sep. 2016.

[128] Michael Seufert, Sebastian Egger, Martin Slanina, Thomas Zinner, Tobias Hossfeld, and Phuoc Tran-Gia. A Survey on Quality of Experience of HTTP Adaptive Streaming. *IEEE Communications Surveys & Tutorials*, 17(1):469–492, 2015.

[129] Christian Sieber, Tobias Hoßfeld, Thomas Zinner, Phuoc Tran-Gia, and Christian Timmerer. Implementation and User-centric Comparison of a Novel Adaptation Logic for DASH with SVC. In *Integrated Network Management (IM 2013), 2013 IFIP/IEEE International Symposium on*, pages 1318–1323. IEEE, 2013.

[130] I. Sodagar. The MPEG-DASH Standard for Multimedia Streaming Over the Internet. *IEEE MultiMedia*, 18(4):62–67, Oct 2011.

[131] K. Spiteri, R. Urgaonkar, and R. K. Sitaraman. BOLA: Near-Optimal Bitrate Adaptation for Online Videos. In *IEEE INFOCOM*, Apr. 2016.

[132] Truong Cong Thang, Quang-Dung Ho, Jung Won Kang, and Anh T Pham. Adaptive Streaming of Audiovisual Content using MPEG DASH. *IEEE Transactions on Consumer Electronics*, 58(1), 2012.

[133] Truong Cong Thang, Hung T Le, Anh T Pham, and Yong Man Ro. An Evaluation of Bitrate Adaptation Methods for HTTP Live Streaming. *IEEE Journal on Selected Areas in Communications*, 32(4):693–705, 2014.

[134] The Linux Foundation. Fast Data project (fd.io) Community ICN (CICN). https://wiki.fd.io/view/Cicn, 2017.

[135] E.D.R. Thomas, M.O. Deventer, T. van Stockhammer, A.C. Begen, and J. Famaey. Enhancing MPEG DASH Performance via Server and Network Assistance. In *Proceedings of IET and IBC*, pages 48–53. IET, 2015.

[136] Jeroen van der Hooft, Stefano Petrangeli, Tim Wauters, Rafael Huysegems, Patrice Rondao Alface, Tom Bostoen, and Filip De Turck. HTTP/2-based Adaptive Streaming of HEVC Video over 4G/LTE Networks. *IEEE Communications Letters*, 20(11):2177–2180, 2016.

[137] Vidyo.io. How vidyo.io Delivers Massive Scalability while Maintaining Reliability and Quality through Cascading SFUs. `https://vidyo.io/blog/features/vidyo-io-delivers-massive-scalability-maintaining-reliability-quality-cascading-sfus/`.

[138] Cong Wang, Amr Rizk, and Michael Zink. SQUAD: A Spectrum-based Quality Adaptation for Dynamic Adaptive Streaming over HTTP. In *Proc. of ACM MMSys*, 2016.

[139] Sheng Wei and Viswanathan Swaminathan. Low Latency Live Video Streaming over HTTP 2.0. In *Proceedings of Network and Operating System Support on Digital Audio and Video Workshop*, page 37. ACM, 2014.

[140] Cedric Westphal, Stefan Lederer, Daniel Posch, Christian Timmerer, Aytac Azgin, Will Liu, Christopher Mueller, Andrea Detti, Daniel Corujo, Jianping Wang, et al. Adaptive Video Streaming over Information-Centric Networking (ICN). RFC 7933, 2016.

[141] Xiaoqi Yin, Abhishek Jindal, Vyas Sekar, and Bruno Sinopoli. A Control-Theoretic Approach for Dynamic Adaptive Video Streaming over HTTP. In *ACM SIGCOMM Computer Communication Review*, volume 45, pages 325–338. ACM, 2015.

[142] Alex Zambelli. IIS smooth streaming technical overview. *Microsoft Corporation*, 3:40, 2009.

[143] Lixia Zhang, Alexander Afanasyev, Jeffrey Burke, Van Jacobson, Patrick Crowley, Christos Papadopoulos, Lan Wang, Beichuan Zhang, et al. Named Data Networking. *ACM SIGCOMM Computer Communication Review*, 44(3):66–73, 2014.

[144] Liyang Zhang, Syed Obaid Amin, and Cedric Westphal. Demo: VR Video Conferencing over Named Data Networks. In *Proceedings of the 4th ACM Conference on Information-Centric Networking*, ICN '17. ACM, 2017.

[145] Xiaoqing Zhu, Rong Pan, M Ramalho, S de la Cruz, Charles Ganzhorn, P Jones, and Stefano D'Aronco. NADA: A Unified Congestion Control Scheme for Real-Time Media. *Draft IETF, Mar*, 2013.

[146] Zhenkai Zhu, Sen Wang, Xu Yang, Van Jacobson, and Lixia Zhang. ACT: Audio Conference Tool over Named Data Networking. In *Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking*, ICN '11. ACM, 2011.

université PARIS-SACLAY
ÉCOLE DOCTORALE
Sciences et technologies de l'information et de la communication (STIC)

Titre : Améliorer la qualité d'expérience en streaming multimedia en tirant parti des réseaux centrés sur l'information

Mots clés : ICN, MPEG-DASH, QoE, WebRTC

Résumé : les réseaux centrés sur l'information (ICN) sont une architecture prometteuse pour faire face à l'explosion du trafic multimedia sur internet et á la mobilité croissante des utilisateurs: non seulement ICN peut améliorer la qualité d'expérience de l'utilisateur, mais ICN peut également étendre naturelle et de façon transparente la prise en charge du trafic vidéo dans les fonctions réseau. Cependant, à notre connaissance, une évaluation approfondie des avantages apportés par ICN à la diffusion multimédia n'a pas encore été réalisée. Dans cette thèse, nous voulons réduire l'écart qui nous sépare d'une telle évaluation en prenant en compte ICN dans divers scénarios de diffusion multimédia.
Tout d'abord, nous évaluons les avantages apportés par du DAS (Dynamic Adaptive Streaming) basé sur ICN par rapport au streaming basé sur TCP/IP, au moyen d'une campagne expérimentale comprenant plusieurs canaux (des émulations Wi-Fi et LTE, des traçes 3G/4G), plusieurs clients (mélange homogène et hétérogène, arrivées synchrones et asynchrones) et des logiques d'adaptation DAS soigneusement sélectionnées pour couvrir les deux grandes familles d'algorithmes disponibles. Nous mettons aussi en exergue les pièges potentiels qui sont néanmoins facilement évitables.
Ensuite, nous montrons comment l'assistance du réseau contribue à améliorer la qualité d'expérience des ultilisateurs. Pour ce faire, nous tirons parti de la fonctionnalité de mise en cache réseau d'ICN et proposons un signal réseau simple envoyé périodiquement par le cache à exploiter par l'algorithme d'adaptation DAS pour optimiser la qualité d'expérience de l'utilisateur en évitant le phénomène bien connu des oscillations induites par le cache. Des expériences nous permettent de valider le bien-fondé de notre approche.
Enfin, puisque la diffusion multimedia en direct gagne du terrain, nous proposons hICN-RTC, en intégrant hICN (hybrid ICN), une solution ICN-dans-IP, à WebRTC, accompagné du protocole RICTP (Realtime Information Centric Transport Protocol), un protocole de transport basé sur le contenu, qui minimise la latence. Bien que toujours en développement, les résultats des premières expériences sont prometteurs car ils montrent que le trafic induit par hICN-RTC ne croit qu'avec le nombre de locuteurs actifs plutôt qu'avec le nombre total de participants.

Title : Improving quality of experience in multimedia streaming by leveraging Information-Centric Networking

Keywords : ICN, MPEG-DASH, QoE, WebRTC

Abstract : Information-Centric Networking (ICN) is a promising architecture to address today Internet multimedia traffic explosion and increasing user mobility: not only to enhance the user's quality of experience, but also to naturally and seamlessly extend video support deeper in the network functions. However, to the best of our knowledge, a thorough assessment of the benefits brought by ICN to multimedia delivery has not been done yet. In this thesis, we aim at reducing the gap to such assessment, by considering ICN in various multimedia delivery scenarios.
First, we assess the benefits brought by an ICN-based Dynamic Adaptive Streaming (DAS) compared to TCP/IP based streaming, by means of an experimental campaign that includes multiple channels (e.g., emulated Wi-Fi and LTE, real 3G/4G traces), multiple clients (homogeneous vs heterogeneous mixture, synchronous vs asynchronous arrivals) and carefully selected DAS adaptation logics to cover the broad families of available adaptation algorithms. We also warn about potential pitfalls that are nonetheless easily avoidable.
Second, we show how network assistance helps improving the users' quality of experience. To do so, we leverage the in-network caching feature of ICN and propose a simple periodical network signal from the cache (i.e., per-quality hit ratio) to be exploited by DAS adaptation logic to enhance further the user's quality of experience by avoiding the known cache-induced quality oscillations. We confirm the soundness of our approach through experiments.
Finally, as live multimedia delivery is gaining momentum, we propose hICN-RTC by integrating hICN (hybrid ICN), an ICN-in-IP solution, to WebRTC and we design RICTP (Realtime Information Centric Transport Protocol), a content-aware transport that minimizes the communication latency. Although still in development, the results we gathered from early experiments are promising as they show that hICN-RTC scales with the number of active speakers rather than the total number of participants.

Université Paris-Saclay
Espace Technologique / Immeuble Discovery
Route de l'Orme aux Merisiers RD 128 / 91190 Saint-Aubin, France