Towards a scalable and programmable incremental deployment of ICN in the real world

Mauro Sardara

▶ **To cite this version:**

Mauro Sardara. Towards a scalable and programmable incremental deployment of ICN in the real world. Networking and Internet Architecture [cs.NI]. Université Paris Saclay (COmUE), 2019. English. NNT : 2019SACLT042 . tel-02497506

HAL Id: tel-02497506
https://pastel.hal.science/tel-02497506v1

Submitted on 3 Mar 2020

HAL is a multi-disciplinary open access archive for the deposit and dissemination of scientific research documents, whether they are published or not. The documents may come from teaching and research institutions in France or abroad, or from public or private research centers.

L'archive ouverte pluridisciplinaire **HAL**, est destinée au dépôt et à la diffusion de documents scientifiques de niveau recherche, publiés ou non, émanant des établissements d'enseignement et de recherche français ou étrangers, des laboratoires publics ou privés.

université PARIS-SACLAY

NNT : 2019SACLT042

Thèse de doctorat

Towards a scalable and programmable incremental deployment of ICN in the real world

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 17 Decembre 2019, par

MAURO SARDARA

Composition du Jury :

K. K. Ramakrishnan Professeur, University of California, Riverside	Rapporteur
Lixia Zhang Professeur, University of California, Los Angeles	Rapporteur
Stefano Secci Professeur, Cnam Paris	Président
Jean-Louis Rougier Professeur, Telecom Paris	Examinateur
Thomas Bonald Professeur, Telecom Paris	Directeur de thèse
Luca Muscariello Principal Engineer, Cisco Systems	Co-encadrant

Abstract

Information-Centric Networking (ICN) embraces a family of network architectures rethinking Internet communication principles around named-data. After several years of research and the emergence of a few popular proposals, the idea to replace TCP/IP with data-centric networking remains a subject of debate. ICN advantages have been advocated in the context of 5G networks for the support of highly mobile, multi-access/source and latency minimal patterns of communications. However, large-scale testing and insertion in operational networks are yet to happen, likely due to the lack of a clear incremental deployment strategy. The aim of this thesis is to propose and evaluate effective solutions for deploying ICN.

Firstly, we propose Hybrid-ICN (hICN), an ICN integration inside IP (rather that over/ under/ in place of) that has the ambition to trade-off no ICN architectural principles. By reusing existing packet formats, hICN brings innovation inside the IP stack, requiring minimal software upgrades and guaranteeing transparent interconnection with existing IP networks.

Secondly, the thesis focuses on the problem of deploying ICN at the network endpoints, namely at the end host, by designing a transport framework and a socket API that can be used in several ICN architectures such as NDN, CCN and hICN. The framework fosters cutting-edge technologies aiming at providing performance and efficiency to applications. An extensive benchmarking at the end of the chapter will present the performance of the transport framework.

Subsequently, the benefits that hICN network and transport services can bring to applications will be assessed, by considering two main use cases: HTTP and WebRTC. The former represents the de-facto protocol of the Web, while the latter is a new emerging technology increasingly adopted for real time services.

At last, the thesis proposes a solution for programmatically deploying, configuring and managing ICN networks and applications: Virtualized ICN (*vICN*), a programmable unified framework for network configuration and management that uses recent progresses in resource isolation and virtualization techniques. It offers a single, flexible and scalable platform to serve different purposes, in particular the real deployments of ICN in existing IP networks.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Information Centric Networking

ICN identifies a network architecture built upon named data, rather than host location, for a simplified and more efficient user-to-content communication. The idea of named-data networking is not new. It has inspired many proposals, such as TRIAD [71], CBN [48], DONA [87], before getting significant attention in the research community with CCN [82] / NDN [168] and with Pub-Sub based projects (e.g. PSIRP/PURSUIT [1]).

Despite important architectural differences in the mentioned architectures, a common shared idea characterizes ICN: scalable location-independent communications with data-centric security. This results in a native support for mobility, storage and security as network features, that are integrated in the architecture by design, rather than as an afterthought. Several years of research and in-lab experimentation have advanced the architectural design and contributed to show ICN potential. However, the ICN idea still divides the community because of the "gain/pain" trade-off [56] related to ICN introduction in existing IP networks.

Recently, a regain of industrial and academic interest in ICN has been generated by the need for network designs capable to face future 5G network challenges. The next generation of radio-mobile networks has the ambition to serve a large number of use cases across several vertical markets and ICN has been identified as one promising candidate

to bring the required benefits at the network edge in terms of performance, scalability and cost [25, 116, 8]. 5G architectural discussions have also revived the debate about deployment path and cost for ICN introduction in operational networks.

Even if virtualization and application-centric network slicing in 5G may accommodate the use of new data planes like ICN [128, 125], skepticism remains about short term clean slate ICN insertion.

1.2 Thesis statement

The aim of this thesis is to propose an effective solution for incrementally enabling ICN into the current Internet, considering three main crucial aspects:

- **Deployment of ICN in the current Internet**
- **Providing ICN transport services to the endpoints**
- **Enhancing and Automating Configuration, Management and Control of an ICN network**

Firstly, this thesis proposes an effective solution for **incrementally deploying ICN into the current Internet**. ICN is usually deployed exploiting an overlay approach, due to the difficult to change the network layer (Internet Ossification) and the effort required for standardize a new layer three protocol. In addition, all the hardware built over the last three decades is optimized for IP forwarding, and deploying new devices optimized for ICN is expensive for operators. Chapter 3 proposes Hybrid ICN, an elegant solution to the problem of ICN deployment. Hybrid ICN (hICN) is a novel ICN architecture built on top of the idea of using IP addresses as ICN names and mapping the ICN semantics inside the IP and TCP headers. This allows ICN packets to transparently traverse the internet, as they appear exactly like normal IP packets. At the same time, these packets can be processed by few hICN routers in the network, which can be strategically placed at the edge as extension of standard IP routers, for example in the form of Virtual Network

Function. With an extensive experiment campaign, the chapter demonstrates that hICN packets can traverse the current internet and at the same time the hICN architecture does not trade-off any of the ICN principles, that will be presented in chapter 2.

Secondly, the thesis focuses on the problem of **deploying ICN at the network endpoints**, namely at the end host. This is usually done through software libraries providing API to applications. However, most of the time these libraries only allow to develop applications on top of the network layer: although this approach gives a packet-level control of the communication, most of the time it results in a too complex challenge for application developers, forced to deal with typical layer 4 problems such as congestion control and segmentation/reassembly operations. What an application developer would like to achieve is to be able to exchange Application Data Unit (ADU) in a simple way, without dealing with network problems such as retransmissions, jitter, MTU discovery etc. What it is required is a transport layer offering a clean and simple API which allows developers to access transport services as they currently do with the Socket API on all the modern operating systems. One of the reasons of the success of TCP/IP is the fact that applications can use it through simple APIs allowing developers to communicate through the network as they were simply writing to a file. Chapter 4 will present the ICN transport framework and a set of Socket API allowing application developers to easily access the ICN transport services. In particular this chapter will describe the architecture of the transport framework and its implementation, fostering cutting-edge technologies such as VPP [152] and aiming at providing performance and efficiency to applications. An extensive benchmarking at the end of the chapter will present the performance of the transport framework, focusing on how ICN transport services may affect applications.

Subsequently, chapter 5 and chapter 6 assess the benefits that hICN network and transport services can bring to applications. In particular it will consider two main use cases: HTTP and WebRTC. Although the semantic of HTTP is Request/Reply, single HTTP messages are pushed from client to server and viceversa, since HTTP assumes the underlying transport to be push based: messages are sent from the sender to the receiver. For this reason HTTP cannot be directly deployed on top of ICN, being the ICN trans-

port based on the request-reply communication pattern. For doing that an additional transport service is required, placed as a middleware between the application itself and the transport layer: this allows to map HTTP semantics to ICN names, adding the ICN benefits to the unicast client-server communication pattern of HTTP. The chapter will then evaluate the scalability properties of HTTP over ICN, in particular with regard to the server endpoint, which must be able to serve a large number of clients. Then, it will compare the different behavior of the ICN transport with respect to standard TCP/IP one. Results show that HTTP over ICN scales better than the current HTTP over TCP solutions, in particular in the case of linear video broadcasting.

However, the solution presented for HTTP is not valid in case of applications with extremely low latency budget, such as gaming or videoconferencing. WebRTC protocol and architecture are increasingly adopted by media companies for real time services. WebRTC requirements in terms of latency and scalability cannot be met using the same solution adopted for HTTP. For addressing such requirements, chapter 6 proposes a new synchronization protocol tailored for real time applications, as well as a new architecture design able to scale with the content (e.g. the number of active media streams in a video conference) rather than the number of participants to the real time session. Chapter 6 will present the design of hICN-RTC, an integrated WebRTC over hICN system, as well as the hICN-RTC synchronization protocol, allowing to use the hICN pull based transport without introducing any additional latency to the content distribution. Results at the end of the section will confirm this and will assess the scalability properties of the designed solution.

At last, chapter 7 proposes a solution for **programmatically deploying, configuring and managing ICN networks and applications**: Virtualized ICN (*vICN*). In ICN, the binding between application and network layer puts a limits on many network automation/deployment tools, that need to be rethought accordingly. Most of the time such tools do not provide this kind of cross-layer collaboration (e.g. IP address provisioning for hICN names). In addition, they provide limited control on the granularity of the deployed components and they do not offer a clear way of mapping the abstract view of the desired

network model to the necessary steps to attain it. vICN is a preliminary work which tries to fill this gap, in particular in the case of hICN, where a tight collaboration between the application and the network layer is required, being ICN semantics mapped directly into the IP layer.

1.3 List of Publications

Journal Papers

[137] Samain, J., Carofiglio, G., Muscariello, L., Papalini, M., Sardara, M., Tortelli, M., and Rossi, D. Dynamic Adaptive Video Streaming: Towards a Systematic Comparison of ICN and TCP/IP. *IEEE Transactions on Multimedia 19*, 10 (Oct 2017), 2166–2181.

Conference Papers

[138] Sardara, M., Muscariello, L., Augé, J., Enguehard, M., Compagno, A., and Carofiglio, G. Virtualized ICN (vICN): Towards a Unified Network Virtualization Framework for ICN Experimentation. In *Proc. of the 4th ACM SIGCOMM ICN* (New York, NY, USA, 2017), ICN '17, ACM, pp. 109–115.

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

[46] Carofiglio, G., Muscariello, L., Augé, J., Papalini, M., Sardara, M., and Compagno, A. Enabling ICN in the Internet Protocol: Analysis and Evaluation of the Hybrid-ICN Architecture. In *Proc. of ACM SIGCOMM ICN '19* (2019).

Poster and Demonstrations

[136] Samain, J., Augé, J., Carofiglio, G., Muscariello, L., Papalini, M., and Sardara, M. Enhancing Mobile Video Delivery over an Heterogeneous Network Access with Information-Centric Networking. In *Proceedings of the SIGCOMM Posters and Demos* (New York, NY, USA, 2017), SIGCOMM Posters and Demos '17, ACM, pp. 22–24.

[29] Augé, J., Carofiglio, G., Enguehard, M., Muscariello, L., and Sardara, M. Simple and Efficient ICN Network Virtualization with vICN. In *Proceedings of the 4th ACM Conference on Information-Centric Networking* (New York, NY, USA, 2017), ICN '17, ACM, pp. 216–217.

[140] Sardara, M., Muscariello, L., and Compagno, A. Efficient Transport Layer and Socket API for ICN. In *Proceedings of the 5th ACM Conference on Information-Centric Networking* (New York, NY, USA, 2018), ICN '18, ACM, pp. 206–207.

[141] Sardara, M., Samain, J., Augé, J., and Carofiglio, G. Application-specific policy-driven 5G Transport with Hybrid ICN. *2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)* (2019).

Talks and Presentations

- *Tutorial: Community Information Centric Networking (FDio/cicn).* 4th ACM Conference on Information-Centric Networking. Berlin (Germany). 2017

- *(h)ICN Socket Library for HTTP - Leveraging (h)ICN socket library for carrying HTTP messages.* ICNRG Interim Meeting. London (UK). 2018

Open Source

- *FDio CICN*. Community Information Centric Networking. `https://github.com/FDio/cicn`
- *FDio HICN*. Hybrid Information Centric Networking. `https://github.com/FDio/hicn`

Patent Applications

- Luca Muscariello, Giovanna Carofiglio, Michele Papalini, Mauro Sardara.
  In-Network Content Caching Exploiting Variation in Mobility-Prediction Accuracy.
  USPTO Application Number 16/044722
- Luca Muscariello, Giovanna Carofiglio, Mauro Sardara.
  Scaling microservices communication over Information Centric Networks.
  USPTO Application Number 16/384110
- Jordan Augé, Jacques Samain, Mauro Sardara, Alberto Compagno, Giovanna Carofiglio.
  Network policy enforcement through hybrid Information Centric Networking.
  USPTO Application Number 62/868107

Chapter 2

Background on ICN

This chapter briefly reviews the set of key ICN compelling features as highlighted in previous research. The reference ICN design is CCN/NDN. The most updated reference for CCN is the set of RFCs [107], [108], while for NDN the reference is the online project specifications [1].

2.1 Named Data

In ICN, information is addressed by location independent identifiers and network operations are bound to named-data, not location. The basic idea is to enrich network-layer functions with content awareness so that routing, forwarding, caching and data transfer operations are performed on topology-independent content names, rather than on IP addresses. Data are divided into a sequence of packets uniquely identified by a name (called *data* packets) and fetched by the user in a *pull-based* connectionless fashion via named packet requests (called *interests*). Naming data packets allows ICN network to directly interpret and treat content according to its semantics, with no need for DPI (Deep Packet Inspection) or delegation to the application layer.

Even if ICN naming is still an open area of research, a few lessons can be drawn from past research and experimentation [34, 27, 65, 106, 66]:

[1] `https://named-data.net/project/specifications/`

(i) ICN does not need to specify a naming convention which can be instead application-specific;

(ii) a hierarchical structure is recommended for routing scalability to guarantee entries aggregation in name-based routing tables;

(iii) names are not necessarily human-readable but may be hash-based;

(iv) scalability of name-based routing in inter-domain use cases and in presence of producer mobility is still an open research area.

2.2 Dynamic Forwarding

Name-based data plane makes use of soft state [162]: user requests are routed by name and a trail of pending requests is temporarily left in the router to guarantee reverse path forwarding of corresponding data. Additionally, the presence of such pending requests in routers enables request aggregation (and native multicast), dynamic re-routing and application/network aware *forwarding strategies* (e.g. based on popularity, on network status, for multi-path load-balancing, etc.) [163, 132]. The question about feasibility of ICN forwarding pipeline has triggered various studies with promising results in the last years [119, 145, 165, 44].

2.3 Data-Centric Security

Instead of securing connections, ICN model is based on securing data at network layer. Each data packet is digitally signed by the producer, allowing consumers to verify *integrity* and *data-origin authenticity*. A producer is thus required to have and distribute at least one public key. Existing trust models (*e.g.* a PKI or Web-of-Trust) can be used to validate producer identity and key ownership. *Data confidentiality* can be guaranteed by encrypting data payload and preventing information leakage from the name as proposed in [64].

Beyond the still open research challenges surveyed in [103, 26, 110], one commonly recog-

nized benefit of ICN data-centric security approach is that it places trust in producers rather than in hosts that store and serve data. This enables in-network efficient data delivery operations, such as filtering, caching and multicasting, without affecting the data security properties enforced by the data producer.

2.4 Receiver-Driven Connection-less Transport

In contrast with the current sender-based TCP/IP model, ICN transport is receiver-controlled, it does not require connection instantiation and it accommodates retrieval from possibly multiple dynamically discovered sources (chapter 4).

ICN transport builds upon the flow balance principle, guaranteeing corresponding request-data flows on a hop-by-hop basis [7]. A large body of work has looked into ICN transport (surveyed in [129]), not only to propose rate and congestion control mechanisms [133, 166] – especially in the multi-path case [43] – but also to highlight the interaction with in-network caching [42], the coupling with request routing [79, 43], and the new opportunities provided by in-network hop-by-hop rate/loss/congestion control [159, 47] for a more reactive low latency response of involved network nodes.

2.4.1 Other features of the ICN architecture

A result of the above mentioned ICN properties is support for **in-network caching**. The ability to perform a name lookup in router buffers can be exploited for re-use (asynchronous multicast of data via cached replica) and repair (in-network loss control), which is an important differentiator w.r.t. existing end-to-end solutions. Many studies have proven its advantages [167, 171, 131, 49], but also the differences w.r.t. application-level CDN-like caching [45, 161].

Another important implication of ICN naming, security, forwarding and transport model is **native mobility** support. Previous work has highlighted the benefits of ICN seamless mobility, especially in 5G context [127, 85, 172, 57, 155, 30], as mostly deriving from

its “anchor-less” management approach in the data plane. As a result, ICN can handle mobility with no need for a stable point of passage of traffic, rendezvous point or name-location mapping system.

Cependant, la solution présentée pour HTTP n'est pas valable en cas d'applications avec un budget de latence extrêmement faible, comme les jeux ou la vidéoconférence. WebRTC protocole et architecture sont de plus en plus adoptés par les entreprises de médias pour de vrai services de temps. Les exigences WebRTC en termes de latence et d'évolutivité ne peuvent pas être rencontré en utilisant la même solution adoptée pour HTTP. Pour répondre à ces exigences, cref chap: rtc-hicn propose un nouveau protocole de synchronisation adapté au temps réel applications, ainsi qu'une nouvelle conception d'architecture capable d'évoluer avec le contenu (par exemple, le nombre de flux multimédias actifs dans une vidéoconférence) plutôt que le nombre de participants à la session en temps réel. Cref chap: rtc-hicn présentera la conception de hicnrtc , un webrtc intégré sur hicn système, ainsi que le protocole de synchronisation hicnrtc , permettant d'utiliser le hicn transport basé sur l'extraction sans introduire de latence supplémentaire dans le distribution de contenu. Les résultats à la fin de la section le confirmeront et évaluera les propriétés d'évolutivité de la solution conçue.

Enfin, cref chap: vicn propose une solution pour textbf déployer par programmation, configuration et gestion des réseaux et applications ICN: ICN virtualisé (emph vICN). Dans ICN, la liaison entre l'application et la couche réseau limite les de nombreux outils d'automatisation / déploiement de réseaux, qui doivent être repensés en conséquence. La plupart du temps, ces outils ne fournissent pas ce type de couche croisée collaboration (par exemple, fourniture d'adresses IP pour les noms hicn). Dans En outre, ils fournissent un contrôle limité sur la granularité des composants et ils n'offrent pas un moyen clair de cartographier la vue abstraite de la modèle de réseau souhaité aux étapes nécessaires pour y parvenir. vICN est un préliminaire travail qui tente de combler cette lacune, en particulier dans le cas de hicn , où un une collaboration étroite entre l'application et la couche réseau est requise, étant la sémantique ICN mappée directement dans la couche IP.

Chapter 3

Hybrid Information Centric Networking

This chapter focuses on the ICN deployability problem. With respect to the state of the art, the solution presented here leverages the integration of ICN semantics *inside* the IP protocol, by mapping ICN semantics into the IP/TCP headers. This is different with respect to the common ICN deployments, which build the ICN network as an overlay on top of the IP infrastructure.

We do not claim that an overlay approach is inferior, very far from that we believe that an overlay approach is instrumental to design and experiment in a green field. This is the case for NDN/CCNx architectures which are used as reference point for hICN. The motivation for an Hybrid ICN design is to reuse as much as possible existing infrastructure, software, protocols and also development processes already in place in the industry. hICN leverages the state of the art of the research in NDN/CCNx and brings it into a simpler and smoother evolutionary deployment path. It is worth mentioning that inserting ICN as an architecture in the current Internet infrastructure and in industrial development processes requires significant efforts in terms of software integration, protocol interoperability, network management and control. A fundamental requirements in the industry is the ability to insert a new component that can be developed, deployed, monitored and troubleshot using the same processes and tools already in place.

This is the core set of constraints put ahead of us while designing the Hybrid ICN architecture that allowed rapid development and integration in existing systems.

Nevertheless we believe that a green field context is extremely useful to design systems without the additional constraint required by evolutionary deployments. NDN/CCNx as a whole would not have been possible without such as methodology. hICN is designed to continue to benefit from research results in the ICN field as a whole.

We can predict that if hICN deployment is successful and the industry get used to operate an ICN network incrementally, this will pave the way for green field approaches as NDN. This is also true for application developers that may find ICN network services in place to exploit while deploying their novel application.

3.1 Introduction

A partial integration of ICN semantics into IP has been looked in the past to provide an easier introduction in the existing protocol stack at the cost of modified ICN behavior and tradeoff of its benefits. This work shares the opinion that the definition of an incremental solution for ICN insertion into existing IP networks is a *sine qua non* condition for ICN success, even in the long term perspective of a wholesale replacement of IP.

To this aim, one the main contribution of this chapter is to propose a solution for ICN deployment directly inside the Internet Protocol: hICN (Hybrid-ICN).

This solution:

(i) preserves all features and benefits of ICN communication by mapping content names into IP addresses,

(ii) guarantees transparent interconnection of hICN routers with standard IP routers and reuse of existing control and management plane tools,

(iii) minimizes software modifications required to enable hICN in network and client devices.

It is worth remarking that, unlike previous proposals, hICN deployment solution relies on a full ICN integration inside IP packet format/protocols that does not use any encapsulation or tunneling mechanism, rather enriches IP network and transport layers with the ICN

semantics.

hICN design is presented in section 3.2, showing how the architecture preserves all the core features of ICN (described in chapter 2). The deployment strategy for hICN should target few nodes at the network edge, leveraging the transparent interconnection between hICN and standard IP routers. A proof of the feasibility of such deployment is presented in section 3.3, which shows that hICN traffic can traverse a large fraction of ASes in the Internet. Section 3.4 further illustrates some of hICN potential benefits for live video streaming over mobile and heterogeneous networks, as inherited by ICN, and quantified over traditional IP network and transport layer approaches. Rather than on the novelty of demonstrated ICN advantages, the focus of the assessment is on the capability to fully realize ICN gains at minimum integration cost in the existing IP infrastructure, both from the network and the application point of view. Finally, section 3.6 provides a review of related work and a short discussion of next steps.

3.2 hICN Design

The main goal of the hICN architecture is to bring ICN capabilities into existing IP networks, while guaranteeing incremental deployment in networks where only few strategic nodes are hICN enabled. More specifically, hICN *integrates ICN in IP* and, unlike other proposals which are described in section 3.5, it does not use any encapsulation nor tunneling techniques, and does not run as an overlay network. hICN by design assumes to share the same infrastructures with regular IP traffic.

Three major principles can be identified in the design:

(i) do not sacrifice any of the ICN features;

(ii) transparently interconnect hICN routers with standard IP routers, that will forward hICN packets as normal ones, as well as hICN routers will forward standard IP packets;

(iii) reuse most of the existing software and hardware technology, in order to minimize the effort to adopt hICN in the near future.

The rest of this section will describe the hICN design and validate if and how ICN key features, as presented in chapter 2, are preserved.

3.2.1 Named Data

As in ICN, hICN addresses each piece of data by name. In hICN, content names are network level names used by hICN routers to forward packets. We stress that content names are different from application level names (e.g. the URI for the identification of a web object) which are application dependent and hICN routers are oblivious to them.

An hICN name is made of two parts: the *name prefix* and the *name suffix*. The name prefix is used by routers for the forwarding operations while the suffix contains the segmentation information and it is mostly used for transport purpose. The concatenation of those two components generates unambiguous names, which are used to uniquely identify each data. hICN name prefixes are standard IP addresses that are assigned by the network administrator for this specific purpose. In particular, it could be envisaged the creation of a new address family `AF_HICN` to carry such names. As described later in this section, this can help during the forwarding operations to distinguish between standard IP and hICN packets. Notice however that this is not mandatory: to identify an hICN packet is sufficient to know the list of prefixes used to route content and they are available in the hICN routers FIB which could be entirely managed by the control plane.

hICN inherits the ICN request/reply protocol semantics [82]: an interest packet is used to request a data packet carrying the actual payload. The definition of the two protocol data units encompasses both network and transport headers as illustrated in figures 3.1 and 3.2. They are standard IPv6 and TCP headers with modified semantic of few fields (underlined in the pictures). The most important fields are the `Name Prefix` and the `Name Suffix`. The former is carried in the IP destination address field for interest packets, whereas it is placed in the IP source address field for data packets. The latter is written in the TCP sequence number field and carries the segmentation information used by the hICN transport layer. Additional fields are `Path Label`, `Loss Detection and`

`Recovery`, `Lifetime` and `Time Scale`. `Path Label` is used by the transport to handle multiple paths [43]. `Loss Detection and Recovery` is used by the WLDR protocol [47] to recover losses on wireless channels (see Section 3.4). `Lifetime` and `Time Scale` are used to specify the caching time of the packets in the hICN forwarders. Notice also that we modify two of the TCP flags: the `MAN` flag, in replacement of the `URG` flag, is used to specify that the packet payload is a Manifest and the `SIG` flag, that replaces `PSH`, specifies that the payload starts with an authentication header that contains the packet signature. Manifest and authentication header are described in section 3.2.4.

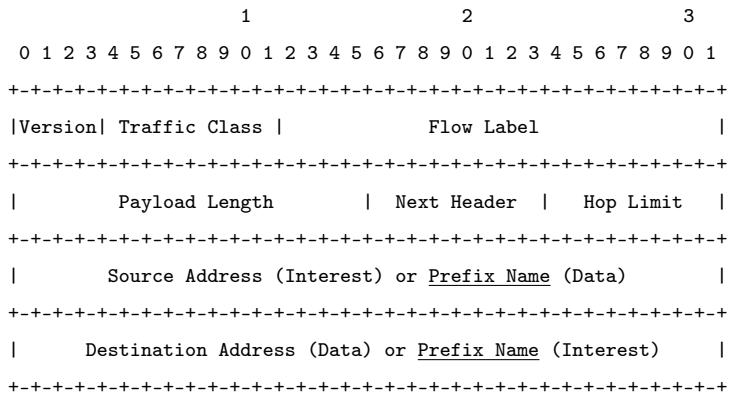


Figure 3.1: hICN interest/data packet inside IPv6 header

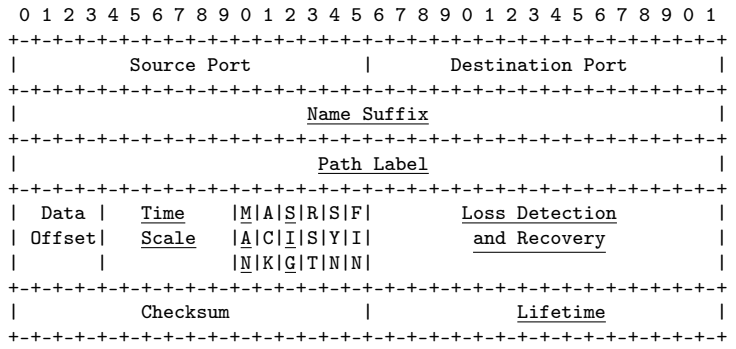


Figure 3.2: hICN transport header inside TCP header.

The modifications presented above allow to carry the information required by the hICN forwarding and transport layer, while preserving protocol layer separation and compatibility with standard IP routers. This will be further analyzed in Section 3.3 by experimentation in the public Internet. IPv6 is presented as the reference technology for hICN, although the design should be able to run hICN using IPv4 packets too. Of course, the

greater addressing space of IPv6 allows for more flexibility in name encoding. The use of the TCP header for hICN is mostly due the need to exploit segmentation and reassembly hardware offloading functions already present in network equipment, while keeping the stack stateless and simple [115, 53]: indeed only the TCP envelope is actually used without implementing the TCP protocol.

3.2.2 Name management

End hosts must provision additional IPv6 prefixes (at least one /128) to produce named data. Provisioning several IPv6 prefixes is a standard operation that does not need any new special mechanism. The hICN host stack is responsible for provisioning prefixes that will be used as names by applications running in the host. For clarity, we can assume that a host requests name prefixes by sending a provisioning request to a network service similar to standard DHCPv6, or an extension of it. Such a network service is the actual owner of the prefixes that are temporarily leased to the host. Similar to what happens today for interface identifiers, the lease can be static or dynamic and may require the host to authenticate (e.g. 802.1X). The actual owner of the prefix is the entity that guarantees that prefixes are routable in a given domain, private or public. This does not constitute any difference with what happens today in IPv6 address space management. In hICN a host does not announce prefixes to the network, it is the local autonomous system to announce routable name prefixes to neighbors. The latter may look like a tautology as current network management works exactly in this same way. This means that hICN name prefix management inherits all protocols and mechanisms currently used for interface identifiers (IPv6 addresses). The consequence of that is that routing over name prefixes can reuse all routing protocols currently used in the Internet. The ability of today routing protocols to provide several routes to reach a given prefix is significant, even though poorly used mostly to avoid traffic instabilities and out-of-order delivery that IP and TCP cannot manage well. These two problems are addressed well in hICN thanks to local flow balance which provides a simple way to make traffic engineering dynamic, yet stable. The consumer end point is also responsible for taking care of out of order

delivery.

As clarified above, in hICN it is not the application that owns name prefixes, but it is the host that leases them from a network administrator. In this respect, hICN may resemble to protocols such as ILNP [28], LISP [55] where the host provision an hostname or a host identifier (ILNP or LISP) from a network service.

3.2.3 Dynamic Forwarding

The data structures implemented inside an hICN forwarder are similar to those required in an ICN one. A major difference between hICN and ICN forwarders is that the former can reuse some of the existing IP data structures. An example is the Forwarding Information Base (FIB), which in hICN is in fact a regular IP FIB, where hICN name prefixes coexist with normal IP addresses. The FIB can be populated with hICN names using standard IP routing protocols to distribute them in the network (such as ISIS, OSPF and BGP). However, multi-path and multi-source are two important properties that routing algorithms should provide in order to support to hICN forwarding strategies. Unfortunately, multi-path and multi-source are poorly supported by current IP routing. Approaches leveraging BGP for anycast routing can be instrumental to obtain multiple routes to program forwarding strategies, with no guarantees of optimality though. Recent work in the field has shown this to be feasible in an efficient way [62]. An ICN forwarder also contains the Content Store (CS) and the Pending Interest Table (PIT): the former stores the data packets received by the router to reuse them for future requests; the latter keeps track of the forwarded interests to route the corresponding data packets on their reverse path. These two data structures are required in hICN forwarders. In order to minimize the modification required to a standard IP router the CS and the PIT can be merged in a single data structure, called *packet cache*, that can be used to store both kind of packets, with different insertion/eviction policies. The packet cache is indexed by full name and is implemented exploiting the memory buffers already available in the IP routers.

Forwarding in hICN extends the ICN forwarding operation by adding two more steps: (i) *classification and punting* of the packets when they arrives at the forwarder, (ii) source/destination *address translation* when the packet leaves. In the following we describe the hICN forwarding process in details, starting with these two operations which are characteristic of the hICN architecture. For the sake of completeness, we also report the interest and data packet forwarding, which is similar to forwarding in ICN.

Packet arrival: classification and punting

At packet arrival, an hICN router has to recognize the packet type (standard IP, interest or data) in order to process it in the proper way. The packet classification is done using the Access Control List (ACL) functionality already available in routers. The packets are classified using their source (src) and destination (dst) address: (i) if only src belongs to `AF_HICN` the packet is a data, (ii) if only dst belongs to `AF_HICN` the packet is an interest, (iii) if neither src nor dst are `AF_HICN` addresses, the packet is a regular IP packet and it will be forwarded as usual, finally, (iv) if both src and dst are `AF_HICN` addresses the packet is invalid and should be dropped.

Packet departure: address translation

When a packet leaves the router the output face modifies the packet header. For interests, the source address is replaced by the local router's address on the egress interface (source address translation); for data packets, the destination address is replaced by the IP address of the previous hICN node (destination address translation), which is the source address of the matching interest packet. Using this simple address translation scheme, at each hop a data packet is forwarded to the previous hICN node traversed by the corresponding interest, back to the requesting consumer. This guarantees local flow balance between hICN nodes; however, data packets will not necessarily follow the reverse path of the interest in eventual intermediate regions of regular IP routers.

Interest packet forwarding

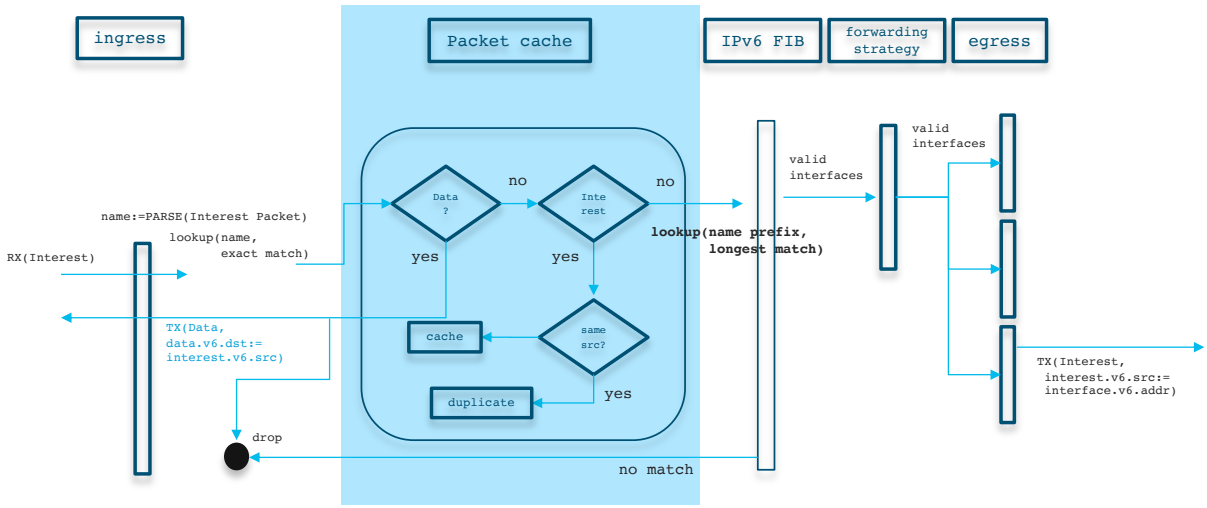


Figure 3.3: Interest pipeline in hICN router

```
Data hit
CachedDataPacket.dst ← InterestPacket.src                    /* dst addr translation */
TX ( CachedDataPacket → InterestPacket.InputFace )
Drop ( InterestPacket )

Interest hit
if InterestPacket.src = CachedInterestPacket.src then
    ManageDuplicate ( InterestPacket )
else
    Cache ( InterestPacket, InputFace )
    ManageAggregation ( InterestPacket )

No hit
AvailableFaces ← LPM ( FIB, InterestPacket.dst)
OutFaces ← ForwardingStrategy ( AvailableFaces )
for ( Face ∈ OutFaces )
    InterestPacket.src ← Face.address                        /* src addr translation */
    TX ( InterestPacket → Face )
Cache ( InterestPacket )
```

Figure 3.4: Interest forwarding path pseudo code.

The forwarding pseudo code for interest packets is presented in Figure 3.4 and it is depicted in Figure 3.3. When an interest packet arrives, an exact match lookup on the full name is performed on the packet cache. In case of *data hit* (the packet cache returns a data packet), the destination address of the data packet is substituted with the source address of the interest packet, which corresponds to the IP address of the previous hICN router. The interest packet is dropped and the data packet is sent from the interest input face.

In the case of *interest hit* (the packet cache returns an interest), the router compares the source address of the received interest with the source address of the matching one. If the two addresses are the same, the incoming interest is a duplicate, otherwise the request comes from a new source. Several ways to handle these two cases already exist in ICN. In hICN, we adopt the one described in [106]: in the first case the interest is considered a re-transmission and it is forwarded, while in the second case it is aggregated in the packet cache and dropped. In this way the hICN node avoids to send upstream multiple requests for the same content. Finally, in case of *no hit* in the packet cache, the interest packet is matched with the FIB of the router. If no match is found the interest is dropped, otherwise it is forwarded to the next hop. hICN is prone to exploit multiple dynamic forwarding strategies to select the next hop among a list of possible candidates. Standard IP has to provide very stable network paths to the transport layer: current IP load-balancers are designed to support strong affinity to previous forwarding choices. This is no more a constraint in hICN, that can expose a more programmable strategy framework.

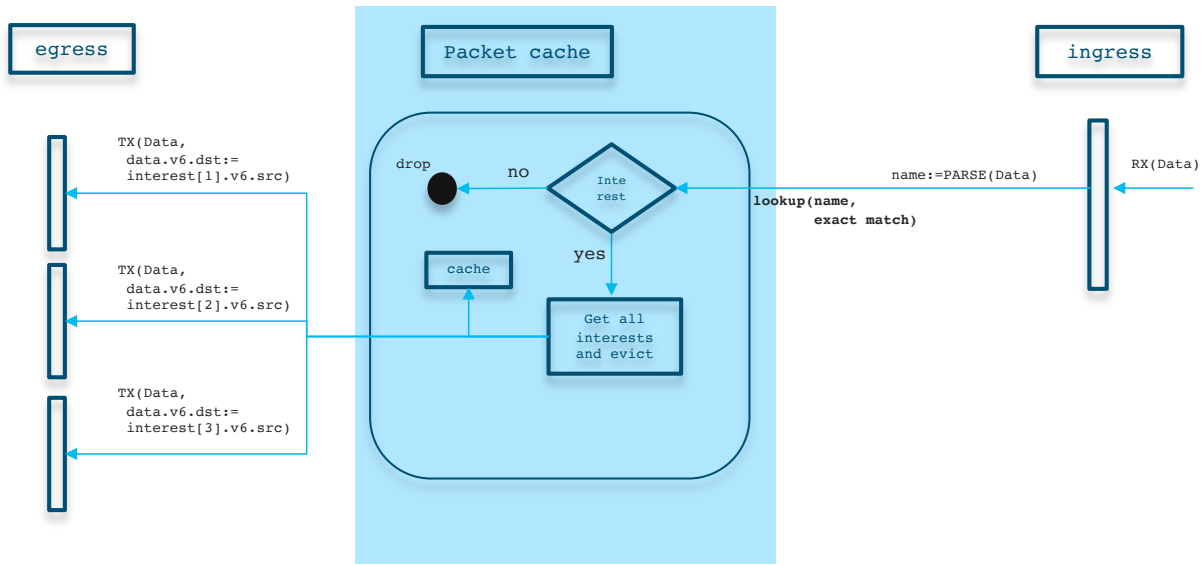


Figure 3.5: Data packet pipeline in hICN router

```
Interest hit
for ( InterestPacket ∈ CachedInterestPackets )
    DataPacket.dst ← InterestPacket.src          /* dst addr translation */
    tx ( DataPacket → InterestPacket.InputFace )
    Evict ( InterestPacket )
Cache ( DataPacket )
```

Figure 3.6: Data forwarding path pseudo code.

Data packet forwarding

The pseudo code for the data packet forwarding is presented in Figure 3.6 ad depicted in Figure 3.5. When a data packet is received, an exact match lookup is performed in the packet cache to locate all matching interests. If no interest packet is returned, the data packet is dropped. Otherwise (*interest hit*) the data packet is cloned to match all found interests, which are then removed from the packet cache. The data packet is forwarded using the input face of each matching interest, after replacing the destination address of the data packet with the source address of the interest. The data packet is also stored in the packet cache.

3.2.4 Data-Centric Security

hICN inherits ICN data-centric security model: integrity, data-origin authenticity and confidentiality are tied to the content rather than to the channel. In particular it is possible to provide integrity and data-origin authenticity in two different ways: (i) using an authentication header or (ii) a transport manifest.

The authentication header carries the signature of the data packet and some information about the original producer. The signature is computed over the immutable fields of the data packet, while the others, including the signature are set to zero. This header is added at the beginning of the data packet payload, and is meaningful only to hICN-enabled routers. In this situation, a bit in the IP/TCP header is set to one.

The transport manifest, designed for ICN in [36], is a L4 entity generated by the producer which contains the list of names in a group of data packets. Each name is associated to a cryptographic hash computed over the corresponding data packet. A client has to

request a manifest to the producer to know the available data packets and to verify them. Data packets carrying a manifest have the `MAN` flag set to one. Using this method, the producer needs to sign only the manifest packets, minimizing the overhead due to packet signature. This approach in fact guarantees a level of security equivalent to individual packet signatures. Not every application can take advantage of the manifest, such as voice over IP.

The ICN data-centric security model mandates that the *linkage* between name and data be authenticated in order to guarantee secure location-independent content retrieval [144]. NDN or CCNx, create a secure bind between the name of the data, the data itself and the producer identity. Therefore, a consumer can verify if the data is signed with a trusted key and can validate the signature to check authenticity of the data with respect to the name carried in the packet. In hICN this mechanism is left unchanged as the signature covers both network name and content payload.

The mapping between application and network names must honor this security feature. Signature verification validates the linkage between the hICN (network) name and the data, but it does not give any guarantee about the linkage between the application name and the data. If the mapping between application and network names is not verifiable by a consumer, this might expose the hICN architecture to an attack in which the consumer requires data with an application name A, but it is actually translated into a network name that correspond to an application name B. For the attack to work, data B must be signed with the same trusted key expected for content A such that the linkage between the network name, the data and the producer is respected. In order to prevent this attack, the mapping between the application names to the network names must be an injective function defined by the producer and validated by the consumer. One way to achieve such secure mapping is to exploit a global name resolution service, such as GNRS [93]. However, deploying a new global system is not an easy task and it might prevent a simple deployment of hICN. A second approach is to exploit the record Address Prefix List (APL) [86] of the DNSSec system in order to map an application prefix to a hICN name prefix. Once a prefix is mapped, each application can define its own mapping

function to further map an application name to the obtained hICN name prefix. A third approach consists in letting applications exploit their own mapping system as in the case of applications that use the Session Initiation Protocol.

Trust management is also similar and hICN can benefit from the most recent work in ICN research [164] on the topic. Additionally, the network service described in Section 3.2.2 serves to bootstrap trust between applications acting as a Certification Authority. When requesting a network prefix, the producer will send its public key to the network service as well as his identity. The network service will create and sign with his private key an hICN data including the producer's identity, the producer's public key, the prefix assigned to the producer, and the time of lease. The hICN name of such data is assigned by the network service and used as key locator for the producer's public key. Trust on the network service can be achieved with any existing trust model (e.g., PKI or Web of Trust), or exploit existing trust systems already deployed (e.g., the BGPSec trust model where the private and public key are those corresponding to the RIR in the RPKI).

For what concerns confidentiality, this is delegated to an upper layer secure transport that we do not further discuss in this thesis. ICN and hICN do not differ on this respect. More research is needed in this area and left for future work.

3.2.5 Receiver-Driven Connectionless Transport

Transport in hICN is similar to ICN: it is *receiver-driven*, *connection-less* and supports *multiple point*. All these features allow for in-network caching, in-network loss and congestion control as well as bandwidth aggregation over heterogeneous networks. The current hICN software uses RAAQM, the receiver-driven congestion control proposed for ICN [43]. The protocol discovers and exploits available content sources in the network, maximizing the bandwidth available at the consumer. The implementation also includes wireless optimizations such as in-network loss control mechanisms introduced in [47]. Both protocols are used in the evaluation, in Section 3.4.

In addition to congestion and flow control, the ICN transport layer, and so the hICN one,

needs to provide data authentication and data integrity verification, as discussed in the previous section. More details about hICN transport design and implementation can be found in chapter 4.

3.2.6 Other features of the hICN architecture

Similarly to for ICN, the design of hICN allows for in-network caching and native mobility support. We already discuss about caching in Section 3.2.3: each hICN forwarder is equipped with a packet cache that stores data packets which can be reused to satisfy future requests. Consumer mobility is fully supported by hICN thanks to name-based addressing. Producer mobility instead requires specific management protocols as in ICN [172]. In particular, the current software implementation provides the anchor-less Map-Me micro-mobility service as described in [30]. Additional work in progress on this topic is done in the IETF DMM WG [32, 31], which is not analyzed in this thesis.

3.3 Feasibility assessment

This section reports the results of an experimental campaign of Internet measurements, performed in order to verify the support for hICN traffic on existing IP networks. End-to-end reachability, middlebox traversal and compatibility with regular IP routers are the target of this evaluation to prove feasibility of hICN insertion in an increasingly "ossified" Internet architecture [76, 77]. These experiments collect empirical evidence that semantic changes in IP/TCP header fields, as well as the lack of underlying TCP state machine, does not result in intermediate nodes dropping, corrupting or interfering with hICN traffic. The observations reported in this section are related to IPv6 measurements. however results apply to the IPv4 context, despite the large number of encountered NATs and connection trackers. A more detailed report is left for future work.

3.3.1 Controlled End-to-End Deployments

For the experiments, hICN has been enabled in a set of representative nodes in academic, residential, enterprise and cloud environments. The tests consist in a content transfer from an hICN-enabled producer to an hICN-enabled consumer over an IP only path, using the producer's IP address as unique content name. The aim of these tests is to validate our open source implementation [6] and tune hICN in order to traverse the most common types of middleboxes.

Context	**Issues**	**Counter-measures**
Academic	None	None
DC/Cloud	None	None
Residential	Stateful firewall	SYN for Interest, RST/ACK for Data
Enterprise	Security appliance	First-hop tunnel

Table 3.1: Summary of end-to-end hICN measurements.

All the tests conducted were successful, meaning that the consumer nodes were able to retrieve the required content. However, some devices were interfering with hICN traffic. Table 3.1 summarizes the types of devices found during the tests, and the counter-measures to be put in place. Stateful firewalls can be traversed by setting well-known destination ports, as well as setting the `SYN` flag on interests, and consequently `RST`/`ACK` flags on data packets. The `RST` flag is the preferred choice, in order not to overwhelm connection trackers. Enterprise contexts are the most problematic, as security appliances can for instance mangle TCP sequence numbers, altering the name of the hICN packets. In these scenarios the only solution is to use tunnels, which have been implemented as new face types in the forwarder, so that they can be selectively applied in a hop-by-hop fashion, thanks to the connection-less nature of hICN. In addition, the test reported that devices performing deep or stateful inspection of traffic are most likely situated close to endpoints. This means that the overhead introduced by these faces can be limited to the first hICN-hop only.

3.3.2 Large Scale Measurements

The tests in the previous section have the intrinsic limitations of the end-to-end approaches [33] and they have been conducted on a small number of controlled nodes. A traceroute-like test is a better option to scale up, where TTL-limited probes are sent towards every announced IP prefixes. Upon expiration, those packets eventually elicit an ICMP response from the routers on path. The ICMP response contains a copy of the original headers and it both acts as a proof that the packet made it up to that point, and also reveals alterations, if any. The test has been performed using hICN packet headers populated with values characteristic of interest and data packets and using different variations on TCP flags.

To run the test, the IP prefixes have been extracted from Routeviews datasets [10]. The dataset contains 48619 prefixes announced by 14398 ASes. According to the CIDR IPv6 report [2] there are 14455 ASes in the routing system at the time of writing so this dataset covers almost the entire Internet. To map each IP address discovered during the tests to its AS, Team-Cymru IP-to-ASN service [24] has been used. In addition, PeeringDB information [9] allowed to further annotate and categorize the ASes.

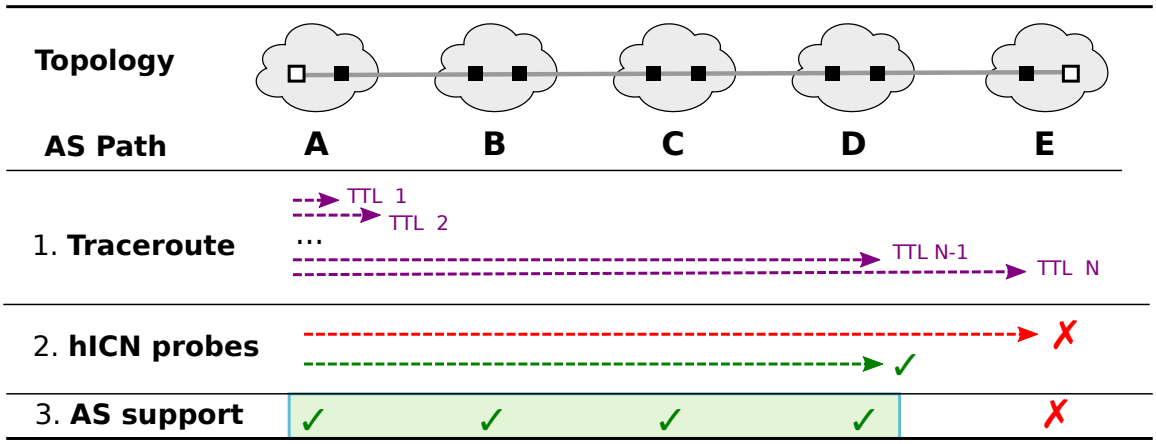


Figure 3.7: Illustration of one radar-like measurement to validate hICN packet support by intermediate AS.

The procedure adopted in our the is illustrated in Figure 3.7. In the first phase, traceroute-like traffic with increasing TTL has been sent towards a representative IP address of each prefix in the data set (the first address in the subnet). The TTL has been increased until

either the probes enter the destination prefix or they cannot go further. This allows to map the set of responsive hops at a given distance (load balancers are taken into account by using paris traceroute [5]), and to verify the absence of rate limiting by sending small packet bursts. In the second phase TTL-limited hICN probes are sent to the responsive hops by proceeding backwards from the stopping point of the previous phase. whether hICN traffic is accepted, dropped, or altered by intermediate nodes. A reply to the hICN probe is successful if it is an ICMPv6 packet of type 3 (time exceeded) with code 0 (hop limit exceeded in transit). In addition, the header in the payload of the reply must be the same as the header of the hICN probe, except for its hop limit counter. A successful response terminates the measurements and mark all previous hops as hICN compliant. Finally, IP-to-AS mapping uncovers the underlying topology and help us to infer AS-level metrics, taking into account border effects resulting from uncertainties at the AS border.

	Total	**Coverage**		**hICN support**	
	#AS	**#AS**	**ratio %**	**#AS**	**ratio %**
Cable/DSL/ISP	2291	1169	51.03	1032	88.28
Content	914	480	52.52	423	88.13
Academic	290	181	62.41	160	88.40
Enterprise	257	90	35.02	81	90.00
Non-Profit	211	103	48.8	85	82.52
Not Disclosed	850	355	41.76	309	87.04
Service Provider	1483	991	66.82	912	92.03
Route Server	18	10	55.5	10	100.00
Unknown	8104	2530	31.21	2155	85.17
TOTAL	14418	5909	40.98	5166	87.4

Table 3.2: AS-level support of hICN as revealed by our large scale measurements

Table 3.2 reports the results of the tests, aggregated at AS-level. Despite its simplicity, this approach manages to sample a representative subset of the overall IPv6-enabled ASes, both in number and diversity. Analysis of these data in light of AS-level topologies provided in [15] revealed that most missing AS are stubs without customers, confirming that we are indeed well-covering Internet core. We noticed that this approach reports missing measurements when approaching the targeted destination: in 95% of the cases

this happens when reaching a small AS and suggests that the traceroute traffic is filtered. Finally, in several cases the IP addresses used belonged to a non-routed prefix. This is mostly due to prevalent prefix aggregation close to destinations. In the future, the usage of a hit-list will allow us to increase the AS coverage, especially of stub and small ASes.

The results obtained also confirm the findings in the end-to-end experiments: most of the middleboxes that may interfere with hICN traffic are deployed at the edge. In fact, even after testing all the counter-measures identified in the previous section, results reveal only negligible differences with respect to plain hICN packets. This is due to the fact that the experiment mostly covered the core of the Internet, where there is a small chance to hit a middlebox.

As a last analysis, only ASes carrying more than 1Tb/s of traffic are taken into account (they are 138 according to PeeringDB). The results show that hICN is able to traverse all of them. It is important to remark that these results only provide a lower bound of success, since the hICN support of an AS supports cannot be neither validated nor denied in absence of response to our probes.

Overall, these results are promising and confirm the suggestion to deal with problematic equipment close to the edge through specific faces, leveraging the hop-by-hop capabilities of hICN. Core routers and servers can then only be limited to the plain version of the protocol for minimal state and maximum processing performance.

3.4 Linear video distribution

Linear Adaptive Bit Rate (ABR) video distribution is a challenging use case in general as it requires provable QoE in terms of video quality, application responsiveness and it is also supposed to scale to a very large number of watchers. Serving millions of concurrent video streams with the usual broadcast TV quality is a challenge faced by all big players in content distribution, often via proprietary in-house CDN-like solutions [12]. These technologies are at an early stage, providing limited support for rate adaptation and mobility. ABR video streaming for linear video is a use case which has already triggered

attention in the ICN community and different works have shown several advantages in using this technology [160, 137, 97, 120, 124, 37, 70, 90, 91, 122].

This section considers the ABR linear video distribution use case to show the benefits of hICN and verify that the current design and implementation meet all the initial design principles:

(i) hICN does not trade-off any of the ICN features
(ii) it allows for interoperability between hICN and plain IP nodes
(iii) it can be incrementally deployed, enabling hICN only on few selected nodes.

The open source hICN software distribution provides ABR application support, that we use in this set of experiments.

3.4.1 Workload and Implementation

In these experiments the content source consists in a live video feed sent by the Open Broadcaster Software (OBS) [21] sending a RTMP stream to a *nginx* [14] server providing multi-quality HLS streams through the `nginx-rtmp` [20] module. During the tests 48 channels are streamed, each one encoded in 4 qualities (using bit rates suggested in [17]) with 2 seconds segments, ranging from 360p at 1Mbit/s to 1080p at 6Mbit/s.

The full hICN stack is based on our open source project *Fast Data* project [1] [6]. The server side makes use of a forwarder based on the high-performance Vector Packet Processing framework (VPP) [152]. This implementation consists of a plugin that adds hICN-specific processing nodes to VPP section 4.5.1. Initial benchmarks with a point-to-point workload and realistic mixed packet sizes show that this prototype can easily saturate a 10Gb/s link using a single worker thread. At the client side, the forwarder is implemented as an user-space library (hicn-light). The ABR video HTTP cache is based on the Apache Traffic Server which uses hICN through a plugin which is also available in the Fast Data project. This forwarder achieves about 400Mb/s throughput with a single thread and runs on all major operating systems. These experiments make use of Ubuntu Linux

[1] `https://fd.io`

clients running the VIPER video player, also available in the Fast Data project. This player provides different adaptation logic strategies for ABR video and these experiments use the ADAPTECH strategy [137]. The player is able to retrieve content using the default IP/TCP stack, as well as the ICN and hICN ones. In ICN/hICN mode, the end-points use RAAQM [43], a receiver-driven multi-path congestion control that allows to use multiple paths. For better parameter control, all radios are based on realistic emulation capturing effects of distance, path loss and fading. Details of the emulator can be found in section 7.4.4. These experiments have been setup using vICN, which we describe in chapter 7

3.4.2 In-network Control

A compelling feature of ICN is that it enables efficient in-network rate/loss/congestion control operations [159, 41, 47], resulting from the combination of pull-based request, symmetric hop-by-hop forwarding and in-network caching.

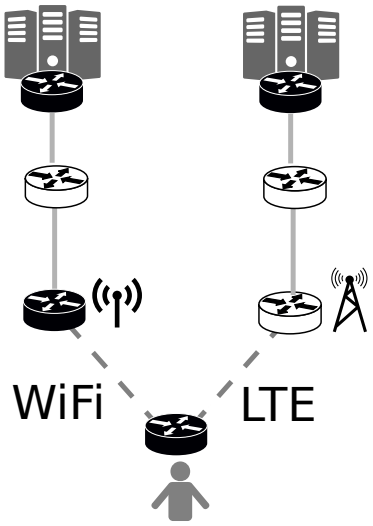


Figure 3.8: Video distribution over HetNet. Black routers are hICN enabled.

In this section, we consider the network in Figure 3.8 where a client is connected to WiFi only. hICN is enabled both at the user and server size, and also in the Access Point (AP), leaving a regular IP router between the AP and the server.

Enabling hICN in the AP allows to benefit of the Wireless Loss Detection and Recovery (WLDR) algorithm introduced in [47] which is available in the open source implement-

ation. The experiments show that enabling hICN only on few nodes lead to the same advantages that we could have gained using a full ICN network with respect to a standard TCP/IP transport.

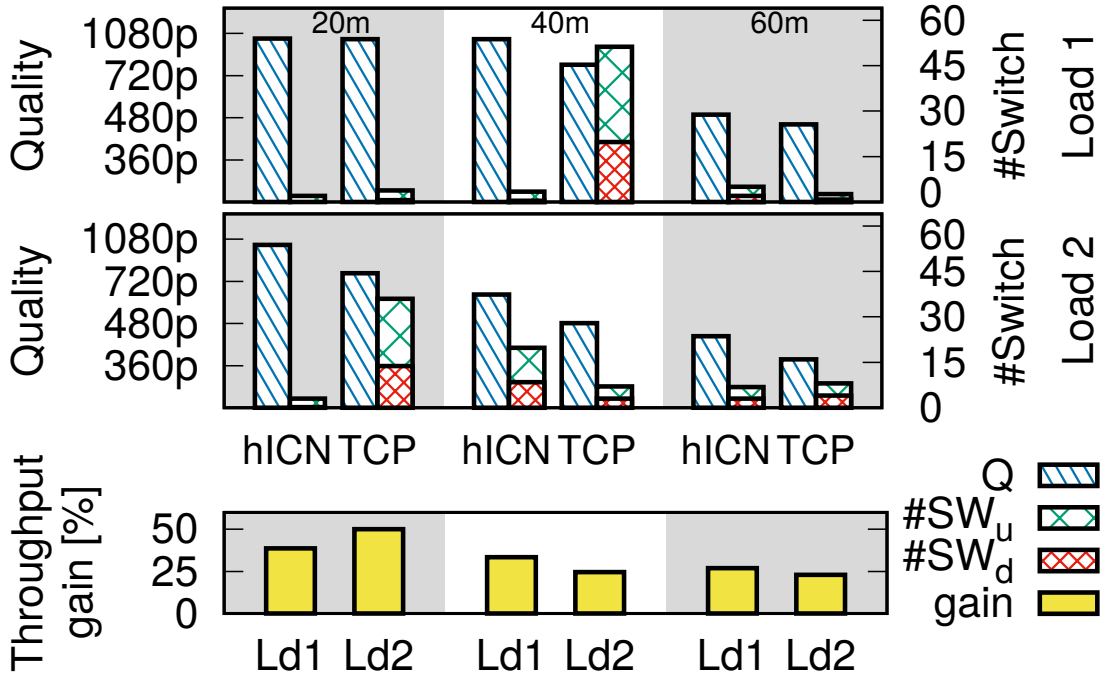


Figure 3.9: TCP vs hICN over WiFi. Average video quality retrieved by the client (Q) and number of quality switches up ($\#SW_u$) and down ($\#SW_d$) with different cross-traffic load on the WiFi channel. The bottom plot shows the throughput gain of the hICN flow with respect to TCP.

Figure 3.9 compares the performance of one CUBIC TCP and one hICN flow over WiFi with the user at 20, 40, 60 meters from the AP, as indicated in the top part of the plot. During these experiment a certain percentage of UDP cross-traffic on the wireless channel is generated using MGEN [19]. This traffic accounts for average loads of 25% and 75% of the available bandwidth (resp. *Load 1* and *Load 2* in the plots). The experiments are repeated 5 times, during which the client watches 5 minutes of a single live channel.

The charts report the average values over all runs. The top part of the figure reports the average video quality Q downloaded by the client and the number of video quality switches, to a higher (switch up) and lower (switch down) quality, respectively $\#SW_u$ and $\#SW_d$.

The tests demonstrate that hICN gets a better average video quality with respect to TCP and, most of the time, less quality switches, which means an improved QoE for the

client. This is thanks to WLDR that recovers losses locally, instead than end-to-end as in TCP. The higher number of switches with hICN at 40m, 75% load can be ascribed to a limitation of the adaptation logic we use. This can be confirmed by measuring the average throughput measured by the application, which we plot as the relative gain of hICN over TCP at the bottom of Figure 3.9. In this smaller plot Load 1 and Load 2 are indicated with $Ld1$ and $Ld2$ respectively. hICN gets consistent superior performance with respect to TCP, ranging from 23% (60m, 75% load) to 50% (20m, 75% load) gain, demonstrating that we can achieve the same benefits of ICN deploying hICN only on few nodes.

3.4.3 Seamless Mobility

This section considers the HetNet access scenario in Figure 3.8. In this scenario the user has access to WiFi and LTE radio access technologies. The two radios are connected, through different networks, to two distinct live feeds, providing the same video channels. Both radios in the experiments use realistic emulation. As for the previous, this test os repeated 5 times, during which the user watches 5 minutes of a single channel. The aim of the experiments is to compare hICN with ICN, to highlight that the two network architectures are actually equivalent and they bring the same benefits. In the ICN scenario ICN is enabled on all the nodes (using the CICN implementation [149]), while, for hICN, only the user, the AP and the two servers are hICN enabled. WLDR is active on the WiFi channel both for ICN and hICN.

The results, reported on Figure 3.10, compare the two architectures: hICN is displayed at the top, and the standard ICN at the bottom. The experiment starts with the two baseline behaviors over WiFi or LTE only, without any mobility event. The results are displayed on the left and labeled *single radio*. As for the previous, this test shows the average quality and the number of quality switches. The results for the two architectures are comparable. For the WiFi channel WLDR works in the same way both in ICN and hICN. Using only LTE the client gets a higher average quality at the cost of more switches, and again this instability can be attributed to the application adaptation strategy. The middle plots, labeled *mobility*, show the same metrics when the consumer performs handovers

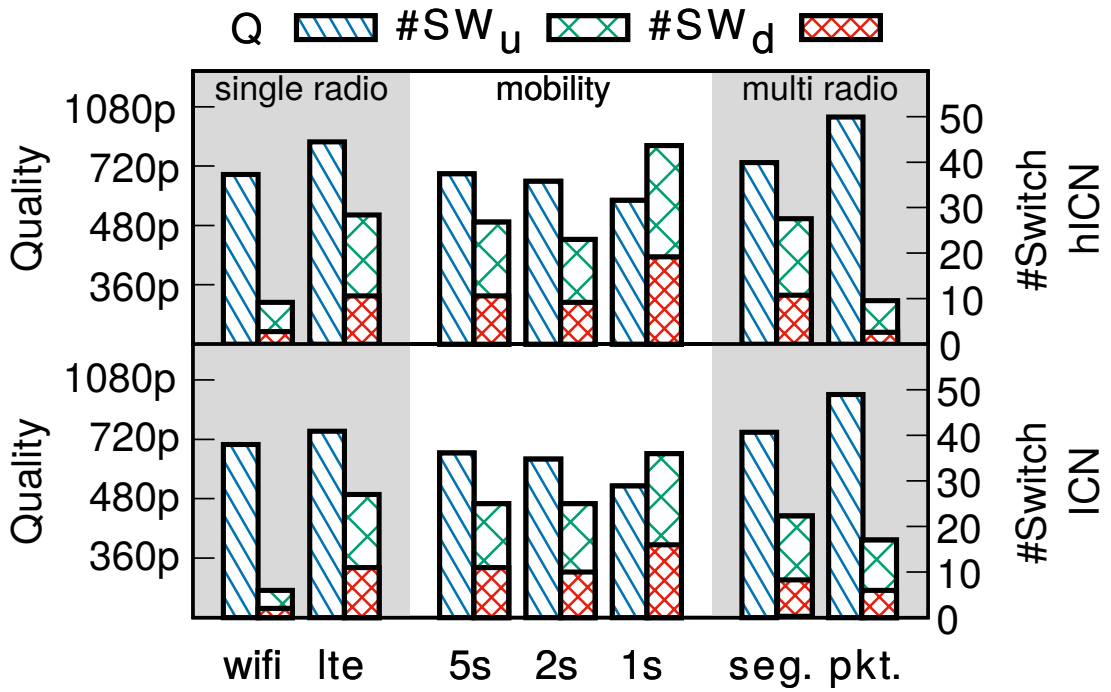


Figure 3.10: Hetnet Access. Comparison between hICN and ICN in case of mobile client and bandwidth aggregation.

between WiFi and LTE every 5, 2 and 1 seconds respectively. Again, the results are almost the same for the two architectures. In particular, thanks to the native consumer mobility support of ICN/hICN, the video quality is not highly affected, despite the fast mobility of the client. Being in control of the request process, the consumer can adapt its interest sending rate according to measured available bandwidth on each path, eventually accounting for newly available interfaces. The performance shown in the chart might be further improved by supporting the client-side mobility recovery mechanisms described in [47], though not available in the current implementation.

3.4.4 Bandwidth Aggregation over HetNet

The receiver-driven transport property of ICN/hICN permits a simple but efficient realization of channel bonding over heterogeneous radios, through the use of congestion aware load-balancing, implemented as a forwarding strategy [43]. It acts at packet level with the objective of minimizing the residual latency on every available path. This scheme can furthermore be applied network-wide and brings full multi-homing/multi-path/multi-source support. This experiment uses the same setting as in the previous section, showing that

the performance of ICN and hICN are consistent. The test involves two different forwarding strategies implementing two different load-balancers: a per-dash-segment load-balancer, which tries to mimic the granularity available for applications today, and the per-packet load-balancer just described. Results are in the right-hand side of Figure 3.10, labeled *multi radio*. The load balancing at segment level (labeled *seg.*) leads most of the time to performance degradation, which is consistent with different observations made in previous work on MPTCP [83], suggesting that those technologies are more appropriate for fast recovery than for channel aggregation in the case of DASH video streaming applications. Per-packet load balancing (labeled *pkt.*) instead achieves an optimal traffic split over the two channels, fully exploiting available bandwidth without any incidence on the application, despite the intrinsic differences between radios. By bonding WiFi and LTE, the video client obtains the highest quality with a minimum number switches. Also in this case, both the hICN and ICN implementation are comparable.

It is also worth highlighting that the client downloads from two different sources as the same time, using disjoint paths and no proxy at junction points. No transport protocol nowadays is able to do the same. This is another nice property of ICN inherited by hICN.

3.5 Related work

Among ICN deployment strategies we can classify them in two broad classes: *full integration* and *partial integration* proposals.

Full integration. In the following, we discuss pros and cons of the main classes of full ICN deployment options [123]:

- **ICN as an overlay** – Envisaging the same transition model as IPv4-to-IPv6, the common deployment proposed for ICN is as an overlay (also "tunneling approach" in [123]): the new ICN protocol stack is transported on top of IP between pre-identified adjacent ICN routers, hereby creating islands of ICN deployments connected to each other via ICN/IP tunnels over existing IP-based infrastructure. The overlay approach considers ICN directly over IP or over UDP and requires the definition of a convergence layer to

map ICN semantics onto application semantics (e.g. HTTP). To improve connectivity and control within and across ICN domains in terms of reliability and of scalability, different SDN-based approaches, and more specifically OpenFlow extensions, have been proposed for ICN deployment as an overlay on top of IP [157, 174, 156, 54]. While it prospects a rapid and easy deployment of ICN in fixed and in mobile networks [148], such deployment configuration requires the standardization of ICN packet format and protocols and, depending on the scale of the ICN deployment, the interoperability between IP and ICN routing protocols.

▪ **ICN as an underlay** – To overcome the limitations of overlay approaches via a native but scoped integration of ICN, proposals have emerged for an ICN deployment as an underlay in given islands of existing IP-based networks (e.g., inside a CDN or edge IoT network) [154]. The connection to the rest of the Internet is guaranteed by gateways or proxies translating semantics from ICN to IP routing domains. Unfortunately, that also implies dual stack challenges and a long timescale for expected adoption of the new stack in network equipments.

▪ **ICN in a slice** – Recently advocated in 5G context, this approach leverages the advances of network virtualization to realize slicing of network (compute, storage, bandwidth) and spectrum resources among applications and introduces ICN for the support of specific services (e.g. low latency, mobile, caching-aided). In [125, 128] the authors suggest creation of service slices using both IP and ICN and discuss the requirements for ICN introduction using programmable data planes.

Partial integration. Other proposals share the spirit of hICN and have suggested to reuse existing protocols to integrate ICN features in IP/TCP/HTTP. However, they only consider a subset of ICN aspects, trading-off some of its benefits, and consequently inherit inefficiencies of the layers underneath.

▪ **ICN semantics in IP** – The following prior art considers ways to embed resource names into IP packets for name-based forwarding. [147] suggests embedding content names in the IPv6 destination address via a proxy mapping HTTP URLs to IPv6 addresses: the

FQDN is resolved (through DNS) and mapped to the first 64 bits of the IP address, while the path section is hashed to form the second half of the IP address. Their idea is to inherit some IPv6 functionalities such as mobility and security, while preserving routing scalability thanks to the two-level hierarchy of names.

CLIP [75] proposes to reserve an IPv6 subnet for content, and to split a content name into publisher label and content label. The publisher label is mapped into source and destination addresses for standard IP forwarding, while the content label is inserted into an ICN header extension and recognized only by ICN-compliant routers. CLIP also suggests a data-centric security model based on IPSec (AH for signing and ESP for encryption), decoupling privacy, authenticity and integrity.

CONET project [54], which considers an SDN-based overlay deployment of ICN with OpenFlow extension in the long term, suggests as short term alternative to use new IP options to carry content-level information. This would require standardization and might suffer from packet drop by non-compliant transit routers.

Unlike [147, 75] that only inherit ICN naming and caching properties, neglecting ICN stateful forwarding and pull-based connectionless transport, [54] aims at preserving ICN transport model and thus requires data packets to flow back to the consumer in the reverse direction. Temporary PIT state is encoded in the packet, in a specific CONET header extension or within payload. This solution has major drawbacks, since it prevents routers from aggregating requests, estimating of the congestion status of a path or performing in-network loss and congestion control.

▪ **ICN semantics in TCP/HTTP** – All previously presented approaches require the introduction and standardization of IP header extensions, which might cause packets to be dropped by routers, or the introduction of new layer 4 or 7 protocols, to be deployed as an overlay on top of IP. A different class of proposals has suggested integration of ICN semantics into transport or application layer protocols.

[146] proposes to use a transparent opportunistic interception of traffic at layer 4 or 7, in order to implement content-level functionalities in TCP. Unfortunately, those operations

are costly and deemed not to scale beyond the network edge.

[121, 56] start from the observation that at application-level, HTTP shares some key aspects with ICN: data addressing by name, pull-based communication model and coupling of request routing with caching. The content-centric nature of HTTP has generated a long debate in the research community about the benefits of a network/transport-layer approach such as ICN versus application-layer HTTP-based approaches for content delivery. In [121], authors develop the thesis that HTTP is already a content-centric protocol, providing middlebox support in the form of reverse and forward proxies, and leveraging DNS to decouple names from addresses. In the same CDN context, [56] demonstrates that little additional benefits come from pervasive caching and nearest-replica routing features of ICN, while still paying the cost for its integration in existing IP infrastructure. All these proposals have the merit of raising the question of the incremental deployability of ICN. However, they mostly target in-network caching and request-to-cache routing features of ICN, trading off other - in our view - key aspects of ICN network/transport layer such as in-network multicast/broadcast, network-assisted loss recovery and congestion control and native mobility support. Our attempt with hICN is to make the full ICN approach incrementally deployable in existing IP networks, without compromising any of the ICN architectural principles and related potential benefits.

3.6 Discussion and Conclusion

Motivated by the importance for short to mid term deployability of ICN as is, this chapter analyzed Hybrid-ICN which has the ambition to bring ICN inside the Internet protocol. Unlike other proposals, hICN focuses on preserving ICN communication model at network and transport layer, to inherit all intrinsic good properties of ICN that past research has highlighted, such as native security, mobility support, dynamic hop-by-hop forwarding and agile multi-path/multi-source transport, coupled with in-network caching.

hICN aims at deploying ICN at the end-points and in a few points at the network edge where beneficial, guaranteeing transparent interconnection with existing IP elements and

reuse of IP routing and management tools. A major contribution of this work is the open source implementation of hICN in the Fast Data project [6], which hash been used to show the feasibility and scalability of the hICN core elements. The use-case of linear video streaming highlights the higher user experience, resulting from in-network loss control, seamless mobility and multi-source/multi-path support over hetnet access, with better usage of network and system resources.

Chapter 4

Transport layer and socket API for ICN

4.1 Introduction

While the previous chapter focused on the design of an architecture for incrementally deploying ICN by re-using the current network infrastructure, this chapter will focus on the problem of deploying ICN on the end hosts, and in particular on the design and implementation of a ICN transport framework and the definition of clean and precise socket APIs.

There has been a considerable amount of work on developing novel applications on top of ICN. However, there has been little effort into developing an intermediate stack, namely the session and transport layers as well as an API that is feature reach and simple at the same time.

The network stack as implemented in current operating systems, exposes Internet sockets (INET) using the BSD socket API for all *NIX OS or Windows Socket API for Windows OS. They are all very similar and application developers are used to deal with this kind of API to develop networked applications.

There has been a remarkable amount of work to develop new transport services in the Internet in order to meet different applications' requirements, such as LEDBAT, QUIC ([89]), MPTCP. In the future, it is likely that novel transport services will be developed

to respond to upcoming needs and to adapt to different environments, namely mobile networks, delay tolerant networks, multi-homed access use cases, just to cite some examples.

The advent of new transport services are also expected to proliferate to serve the development of new applications for the next decades. In this case, developing and defining an API that can be inserted in applications in a simple manner, with extend-able features, is a mandatory requirement for the success of new transport services. The current IETF WG TAPS [118] is the most notable effort in this direction and also the work done in the NEAT project [23].

ICN is not an exception, and its proliferation also depends on the definition and implementation of a simple and easy to use session and transport layer. Currently, ICN lacks of this definition as well as the implementation of a socket API that can facilitate its insertion in existing applications as well as the development of new ones. This chapter describes one such layer that is based on the consumer/producer communication abstraction model and shows how it gracefully fits into a comprehensive middle-ware between applications and the ICN network layer. Moreover, the chapter discusses several transport services of which an implementation is provided. Such implementation is then compared with the corresponding services available in TCP and UDP and results show that performance are comparable in terms of goodput.

The chapter is organized as follows. Section 4.2 gives an overview of the ICN namespaces and their relation with the transport and network layers. Section 4.3 describes the overall architecture with assumptions and constraints; Section 4.4 presents the transport layer as well as the socket API. Section 4.5 describes the details of an high speed implementation of the proposed transport stack that is benchmarked and analyzed in Section 4.6. Section 4.8 concludes the chapter.

4.2 Namespaces in ICN

In ICN it is often assumed that application names are the same names used by the routing plane in the form of routable prefixes. In this section, as already introduced in section 3.2,

we argue that application names and network names should not coincide in order to provide a greater flexibility to organize data at application layer, while maintaining routing scalability, network domains separation and better supporting mobility.

This comes at a cost: the mapping requires a security linkage which is native in architectures such as NDN/CCNx and also the mapping requires to be designed and maintained. This is part of the tradeoff and advantages in the Hybrid ICN architectures.

Today applications manage data using different kind of namespaces that are used for the purpose and functioning of the specific application itself. For instance a web server typically makes data available using a URL which embeds the hostname of the server and defines a locator to determine where the data can be retrieved. The data itself inside the server then may be organized using a namespace that is inherited by the local file systems. Domain names are managed by an authority that maintains a registry of allocated names to registrants.

Other collections of data are organized using URN for objects (RFC 3061 [100]), ISSN (RFC 3043 [101]), DOI (ISO 26324 [58]) and many mores. It is convenient to organize data using a standardized namespace as it allows simple migration of the data, interoperability and integration of different applications among themselves. However, this is not always the case, and most of the time each application develops namespaces autonomously.

The same level of flexibility and customization can be hardly achieved with network names, whose definition usually follows some strict rules and their allocation is handled by the network administrator (who is usually unaware of the applications running at the end-host devices). Additionally, using the same names at network and application layers also issues severe concerns to the routing system. In this case names (prefixes usually) used by applications would be distributed in the routing systems in order to set the forwarding plane. The variety of namespaces used by different applications would make complicated to exploit aggregation to maintain FIB small in the routers, thus affecting routing scalability. Lastly, the data producer would have to attest the ownership of the prefix to the routing system and the different network domains exchanges prefixes to assure reachability.

In order to avoid the issues raised above, an efficient solution is multiplex/demultiplex application names into network names. In the case of hICN, routable name prefixes are IPv6 prefixes and follow the usual rules on IPv6 routing including prefix attestation. The way name prefixes are assigned to a data producer is described in section 3.2.2.

The mapping between network and application namespaces has a direct influence on the ICN data aggregation, routing scalability and data transmission. Application data moves from one location to another by means of data segmentation that has to fit into the minimum maximum transfer unit across the link traversed from the source to the destination(s). If the data does not fit into the MTU of a link it is either discarded or reassembled before retransmission. Instead, the network namespace is data agnostic.

In ICN data packets are directly indexed using a hierarchical name. The hierarchy allows to organize application level data inside name prefixes, but also to better scale routing by name (aggregation) and to define lower level indexes as segment identifiers. One of the compelling usage of ICN architecture is to reduce redundant transmissions because multiple requests for the same data can be satisfied by a single transmission. Immutability of the data that is associated to a given name is a strong requirement at least for the lifetime of a transport session, otherwise reassembly of the data at consumers would not complete successfully.

The segmentation information is included in the L4 header: name suffix, lifetime etc. Fragmentation poses issues, as in IP, as immutability is not preserved and data sharing is suboptimal (some solutions are proposed in [109], [63]). The risk is that, as in IP today, fragmentation makes multi-path transport difficult to optimize, as the characteristics of a network path can influence the way data is segmented in the namespace. For ICN this can be a significant limitation. Hop-by-hop reassembly of fragments seems the best solution at a non negligible cost. Moreover, end-points can charge the network with additional cost for segmentation/reassembly operations.

MTU path discovery protocols can eliminate this cost if used by the end-points when possible. The design of this kind of protocols needs additional work for ICN as a key requirement is to preserve location independence and multi-path communication efficient.

In summary, data namespaces in ICN have an impact on the full stack: from the application, through transport and down to the single data packet in case of segmentation or fragmentation. Each layer has different objectives and constraints. Each layer manages resources of different capacities implying trade offs while multiplexing/demultiplexing name data from one layer to another. Some of these trade offs have been discussed in this section, others will be described in detail concerning specific transport layer functions. In this chapter we consider the transport layer that has adjacencies with application and network layers and has a key role in the overall architecture.

4.3 Architecture

This section presents the network layer requirements to design the socket API and transport Services. The transport layer sits on top of a network layer that provides the semantics of the ICN architecture. The services provided by the network layer protocols to the upper layers are characterized by request/reply semantics, meaning that data is transmitted only upon reception of the corresponding request. Each data is transmitted in a Data Packet that is unambiguously identified by a hierarchical name. Such hierarchical name is used at the network and transport layer to realize location independent name-based forwarding, as well as multiplexing and demultiplexing of communication flows, data segmentation and reassembly. We assume that the network layer does not perform fragmentation, i.e. the L3 PDU fits into link-layer maximum transfer unit (MTU). For this reason, we assume as well that the transport layer relies on a path MTU discovery protocol (PMTUd). The more general problem of designing a PMTUd mechanism that can be effective for multi-path and multi-homed end-points is an important problem but it is out of scope for this thesis, where simple naive approaches are adopted.

Moreover, we assume that application level data is associated to a namespace that applications use to organize, uniquely identify and securely exchange each data. We believe, and encourage, that application level namespaces and names differs from the ICN network layer prefixes and names. This allows ICN to deal with the several orders of

magnitude bigger application level data that can be permanently addressed in an end-host, in particular in case of hICN. Additionally, decoupling application-layer names from their network-layer counterparts provides several benefits in term of packet processing and security [65]. Translation between the application level names to the corresponding network layer names is done in the transport layer. How the translation between network and application name is done is left for future work, but in principle the transport will use the names provided by the network service described in section 3.2.2. This is not different to the current application layer in today's Internet which relies on session and transport layers to conceal details of the networking semantics.

For a given name prefix, used to exchange an application data unit, the name hierarchy is also used in the transport layer for segmentation and reassembly operations, using name suffixes as part of the network namespace to unambiguously identify Data Packets in the network and in a communication flows.

4.4 Transport Services

In this section we present in details how communication sessions are created, mapped into network namespaces and prefixes, and for how long. We describe also how resources are allocated at the end-points in order to serve the communications needs of the applications. We highlight that our socket API and transport services can run on every ICN architecture, e.g., NDN/CCN as well as hICN.

4.4.1 End-points description

We identify two kinds of communication sockets each with a specific API: the producer and consumer sockets. These socket types are designed to exchange data in a multi-point to multi-point manner. The producer-consumer model is a well-known design concept for multi-process synchronization where a shared memory is used to let multiple consumers to retrieve the data that is made available by producer processes into the same memory. In

ICN we have the same concept that is applied to a network where memories are distributed across the communication path. The first memory in the path is the production buffer of the producer end-point that forges Data Packets and copies them into a shared memory isolated into a namespace. Consumer sockets can retrieve data from such memory by using the ICN network layer. The model just described is an inter-process communication example (IPC) that requires data to cross a communication network by using a transport protocol.

The way consumers and producers synchronize depends on application requirements and the transport layer exposes a variety of services: **reliable** / **unreliable** / **realtime**, with or without latency budgets etc.

Independently of the specific requirements of the applications, producer sockets always perform data segmentation from the upper layer into Data Packets, as well as compute digital signatures on the packet security envelop. This envelop can also be computed across a group of packets, by including a cryptographic hash of each packet into the transport manifest (fig. 4.1), and eventually signing only such manifest. This is a socket option that can bring significant performance improvement.

The consumer socket, on the other end, always performs reassembly of Data Packets, hash integrity verification and signature verification. The usual assumption is that the producer socket uses an authentic-able identity while using namespaces that it has been assigned. The end-point must be able to manage the mapping of its identity and the allocated namespace by issuing digital certificates about the mapping. The consumer end-point must retrieve the associated certificate to perform the basic operations. It is out of scope for this thesis how to design and implement a scalable system to perform such certificate operations.

4.4.2 Network namespaces

The session layer takes care of sharing local resources among all communication sessions such as consumer and producer sockets. Any time an application wants to open a socket,

the session layer allocates space from a memory pool and securely isolate it within the valid namespace. Both producer/consumer sockets are successfully instantiated to match against a valid prefix that must be available in the local FIB. Communication flows are multiplexed into the network layer using the namespace itself and do not require the usage of L4 ports as in TCP/IP. The new session is multiplexed to the network stack by registering the new session as a unidirectional application face. FIB entries have to be configured accordingly in case the new application face interconnects a producer or a consumer socket. This kind of faces can be seen as shared memories bound to a name prefix with read/write permissions.

The session is instantiated by passing socket types, options, parameters and is released to the resource pool at closure, including the used namespace. By consequence it implies that the certificate used by a producer end-point is no longer valid and revocation of the certificate must be enforced.

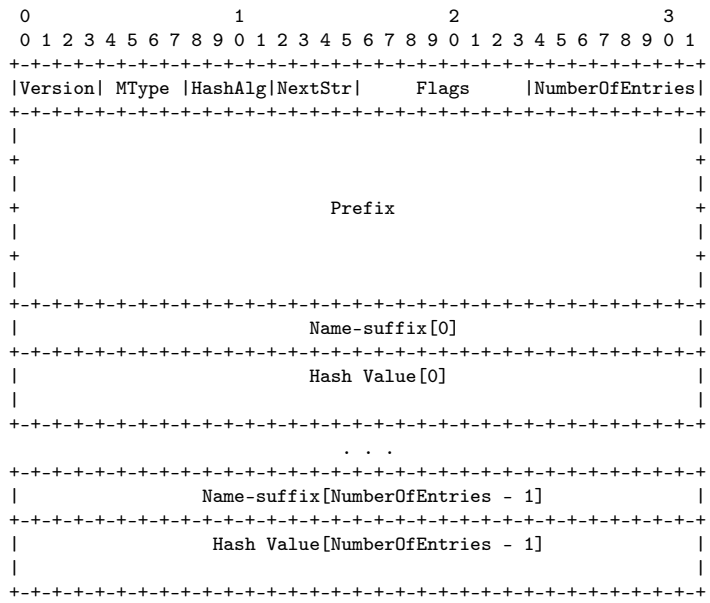


Figure 4.1: Manifest encoding: compact encoding.

4.4.3 Socket API

The system calls of the socket API are based on the socket interface extensions for IPv6 [68] and are shown in Figure 4.2.

```
Common API
int socket (int domain, int socket_type, int protocol);

int bind(int sockfd, struct sockaddr *addr,
         socklen_t addrlen);

ssize_t sendmsg(int sockfd, const struct msghdr *msg,
                int flags);

ssize_t recvmsg(int sockfd, struct msghdr *msg, int flags);
```

```
Consumer specific API
ssize_t recvfrom(int sockfd, void *buf, size_t len,
                 int flags, struct sockaddr *src_addr,
                 socklen_t *addrlen);
```

```
Producer specific API
ssize_t sendto(int sockfd, const void *buf, size_t len,
               int flags, const struct sockaddr *dest_addr,
               socklen_t addrlen);
```

Figure 4.2: System calls for socket initialization and binding to a socket address, as well as for data reception and transmission.

Applications are supposed to call the `socket()` system call to create descriptors representing a communication end-point, i.e., a consumer or a producer. The domain `AF_ICN` defines the address family and the socket type defines the communication semantics: `SOCK_CONS` will create a consumer socket, `SOCK_PROD` will create a producer socket. The consumer socket takes care of data reception, while the producer socket of data transmission. The protocol parameter specifies the protocol used for the data retrieval and the data production: `CONS_REL`/`CONS_UNREL`/`CONS_RTC` for reliable / unreliable / real-time content retrieval and `PROD_REL`/`PROD_UNREL`/`PROD_RTC` for reliable / unreliable / real-time data production. Section 4.5 clarifies the meaning of such parameter.

Both sockets bind to a socket address, who is initialized by specifying the address family `AF_ICN` and the name prefix (`struct sockaddr_icn`). The name prefix enforces the namespace in which a producer socket is allowed to publish data and a consumer socket is allowed to request data.

The `bind()` system call takes care of setting up a local face to the forwarder, and in the case of the producer it also sets a FIB entry *(name_prefix, socket_id)*. The `recvmsg()`

and the `recvfrom()` system calls are used by a consumer for retrieving a content, while `sendmsg()` and `sendto()` are used by a producer for publishing data and making it available for the consumers. Notice that the consumer socket can just use the `recvmsg()` and `recvfrom()` as they are the two system calls capable of specifying the name of the content to be retrieved, and the producer socket can use only the `sendmsg()` and the `sendto()` for publishing data under a certain name. For both cases, the name used for pulling/publishing data has to be in the range previously specified in the `bind()` system call. The `setsockopt()` allows application to tune the available socket options: its semantic with respect to the current sockets API does not change. Some of the available options will be described in Section 4.5.

The consumer socket can use the `sendmsg()` for sending arbitrary interests in a datagram fashion, while the producer socket can call `recvmsg()` for receiving and processing requests from the consumers.

It is worth to highlight that this socket API design is general and it does not restrain developers to develop a variety of different applications patterns that differs from the content distribution applications (in which a set of clients that retrieve content from one or multiple replicated servers). A relevant example of collaboration applications, in which a number of users share data in a real-time fashion with each other will be presented in chapter 6. The design of a specific transport protocol, and its selection through the socket API, will guarantee the communication-delay constraints required by the real-time nature of the communication.

4.5 Implementation

The ICN transport service has been implemented in most of the components in a C++ userspace library. It can be used to connect to the ICN forwarder (hicn-light) or the VPP [152] based forwarder. The source code of the library and the forwarders is available in the two *fast data* projects CICN [149] and HICN [6]. This chapter reports experiments

made by using the VPP-based hICN plugin [1] as it provides best performance compared to the others.

To connect an application with the underlying forwarder, we developed a *Forwarder Connector* which exposes a uniform interface for sending and receiving packets to/from the network stack. The pipeline is reported in Fig. 4.4 and 4.5. A more detailed description is presented in Section 4.5.2. Below we report a short introduction to VPP which constitutes the packet processor that allows to obtain scalable performance in software.

4.5.1 VPP: vector packet processor

VPP is a high-speed software based packet processor that provides advanced data plane function in commercial off-the-shelf (COTS) hardware. VPP design has two major pillars: *(1)* completely bypassing the kernel in order to avoid the overhead associated with kernel-level system calls, *(2)* maximize the number of instructions per clock cycle (IPC). Kernel-bypass is achieved through low-level building blocks, such as Netmap [94] and DPDK [153]. These mechanisms provide Direct Memory Access (DMA) to the memory region used by the NICs, therefore avoiding the need of the kernel to interact with the underlying hardware. Maximization of IPC is achieved carefully designing the VPP implementation and exploiting data pre-fetching, multi-loop, function flattening and direct cache access. In the following we briefly present the VPP architecture and the design of our hICN plugin for VPP.

VPP Architecture

The VPP code is organized in a set of nodes, each implementing specific functions (e.g., ip4 forwarding or fib lookup). There are three types of nodes in VPP: namely *internal*, *process* and *input*. Internal and process nodes form a *forwarding graph* which determines the processing paths each packet follows during it processing. Input nodes interact directly with the NICs, reading packets from the rx ring buffer and injecting them in the forwarding

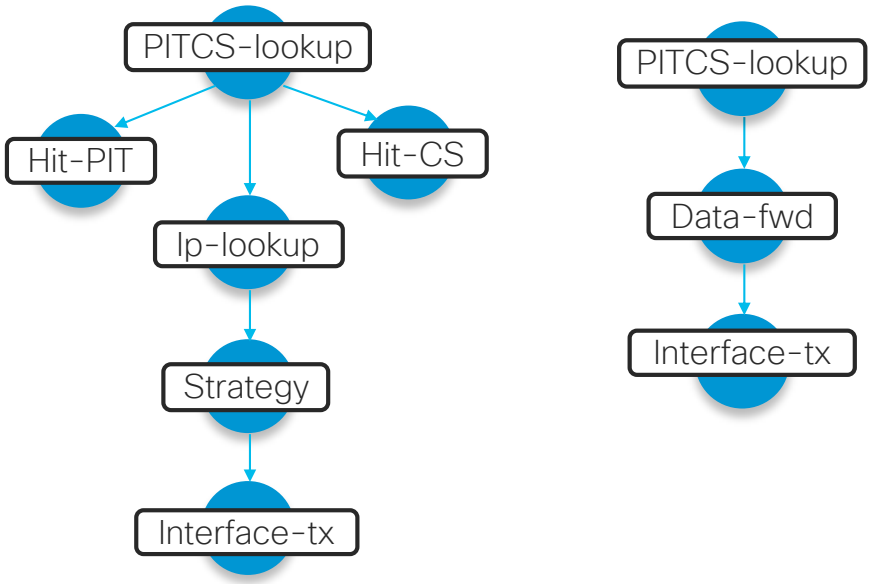
[1] `https://github.com/FDio/hicn/tree/master/hicn-plugin`

graph. Internal nodes implement packet processing functions (e.g., packet forwarding, address rewrite, fib lookup) as well as they move packets to the tx ring buffer in order to let the hardware to forward a packet. VPP processes packets in a *vectorized* fashion: input nodes create a vector of packets, which is moved from internal node to internal node by the graph node dispatcher. Thus, every node executes its processing function on the entire vector. This design allows to minimize cache misses, to reduce the overhead of selecting the next node in the graph, as well as to simplify pre-fetching [92]. Beside supporting DPDK and Netmap compatible NICs, VPP provides other several additional types of interfaces. Among all, memory-based interfaces (in short *memif*) are designed to allow applications to send packets to a local VPP forwarder. Memif interfaces are designed to be efficient in order to forward several gigabits per second. Connectivity between an application and a VPP forwarder is provided by a pair of memif interfaces: one managed by VPP and the other by the application. Such pair of interfaces are virtually connected through a bidirectional link, thus letting packets to flow from one interface to the other. The virtual link is implemented using two circular buffers (one per each direction of the link) stored in a portion of virtual memory shared between the VPP forwarder and the application.

hICN plugin

The ICN forwarder is implemented as a VPP plugin which adds six new nodes in the forwarding graph, enabling two new forwarding sub-graphs: the interest and the data forwarding pipeline and shown in Fig. 4.3.

We exploit the VPP memif interface to implement the application faces, i.e., ICN faces that forward traffic to/from local application. In particular, we designed two different APIs that the transport layer can use to connect to a consumer face and a producer face. Both APIs instantiate a new memif interface, exposing the memif shared memory to the transport layer. Additionally, the producer face API also creates a new entry in the FIB in order to forward Interest Packets to the producer socket.



(a) Interest sub-graph (b) Data sub-graph

Figure 4.3: hICN plugin for VPP

4.5.2 Forwarder Connector

The forwarder connector is the software module in our transport services that interacts with the underlying forwarders. We designed two specific connectors, one for our hICN-VPP forwarder, and one for the client forwarder. Both connectors expose the same northbound interface to the consumer socket and the producer socket. In particular, the interface is composed of four APIs that the transport layer can use to create a consumer/producer connector and to send/receive packets: `connectConsumer()`, `connectProducer()`, `sendPacket ()`, `recvPacket()`. The purpose of the first two API is to create a (or connect to an existing) consumer or producer application face in the hICN forwarder. The `connectProducer ()` also takes care of setting a new FIB entry in order to allow the forwarder to send Interest Packets to the producer socket itself. The `sendPacket()` and `recvPacket()` are intended to send/receive packets to/from the hICN forwarder once it is connected.

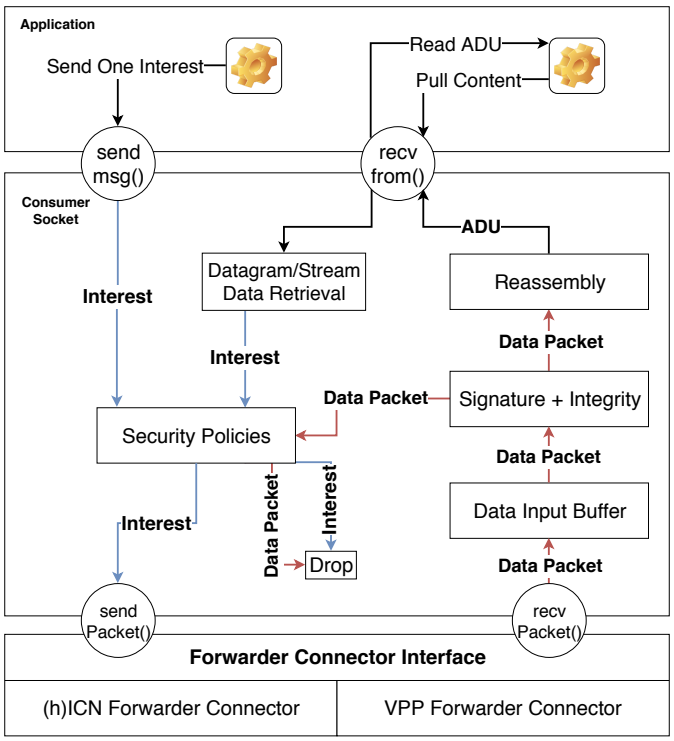


Figure 4.4: Consumer socket processing pipeline

4.5.3 Consumer Socket

Applications can instantiate a consumer socket for *(1)* retrieving a specific content with a certain name (e.g. DASH client) or *(2)* directly sending an interest to retrieve one specific Data Packet (e.g. ping).

Figure 4.4 shows the internal processing pipeline of the socket and points out the path followed by Interest and Data Packets during the content retrieval. The content download is triggered by the call to the `recvfrom` function: the application specifies the name of the content to pull and some download options, such as the actions to perform in case of signature verification failure. After the initial setup phase, the control is taken by the *Data Retrieval protocol*, specified at the socket creation with the `socket` system call in the protocol parameter (`CONS_REL`, `CONS_UNREL`, `CONS_RTC`). The protocol then starts generating Interest packets for retrieving the content requested by the application. Before being forwarded to the forwarder connector, the *Security Policies* block applies the policies specified by the application as socket options: this includes for instance signing the Interest or verifying that the Interest matches all the application and socket constraints

(e.g. the interest name belongs to the prefix specified in the `bind` system call). If all the requirements are satisfied, the Interest is passed to the forwarder connector that takes care of delivering it to the next hICN node.

In the same manner, the forwarder connector takes care of delivering the Data Packets coming from the hICN forwarder to the application. As soon as the Data Packet is received, it is stored it in the *Input Buffer* and then passed to the *Verification Routine* that performs the data-origin authentication check. As mentioned in Section 4.4, this operation can be performed either by using a transport manifest or verifying the signature of each Data Packet, depending on how the producer decided to sign the content. If this check fails, it means either the Data Packet has been corrupted or has been produced by a malicious producer: in this case the security policies set by the application will define the action to perform:

- dropping the Data Packet and sending a new Interest;
- using it anyway;
- aborting the download;

If the data-origin authentication check succeeds, the Data Packet is used for reassembling the application content. When all the Data Packets are successfully reassembled and verified, the consumer socket returns the content to the application.

Since the operation of verifying the data origin authentication is expensive, as we show in Section 4.6, the flow control operations performed by the *Data Retrieval Protocol* must be decoupled from the operations performed by the *Verification Routine*, otherwise the crypto operations would limit the rate of the flow control protocol.

4.5.4 Producer Socket

The producer socket is responsible for producing Data Packets and making them available for the consumers. Data Packets can be published in two different fashions:

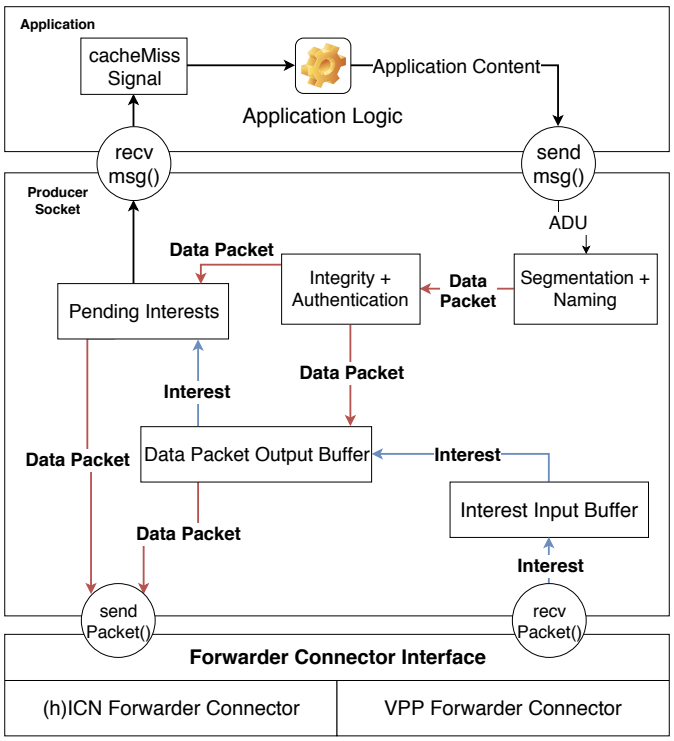


Figure 4.5: Producer socket processing pipeline

(i) **asynchronous production**, if the application publishes *unsolicited* data without receiving any Interest (e.g. real time applications)

(ii) **synchronous production**, if the application publishes data upon the reception of one Interest (e.g. file repository)

We describe first the case of asynchronous content publication and then we will see how applications can use the producer socket for dealing with dynamic requests.

Application can publish data by calling the `sendto` API with:

(i) Application Data Unit

(ii) Application Name

(iii) Crypto suite for signing the Data Packets

(iv) Some other optional publication options (e.g. Transport Manifest)

The first operation performed by the producer socket is the *data segmentation and naming*: the application content is segmented into MTU-size Data Packets which are named by combining the prefix received by the application and a suffix, for unambiguously identifying the packets in the network. Each Data Packet is then processed by the *Signature*

Routine: depending on the publication options provided by the application, the routine decides if signing every packet or groups of packets by using a transport manifest ([36]).

Generally speaking, applications producing a large amount of data such as HTTP servers generally should opt to transport manifest, since it avoids to compute the signature of every Data Packet. Other applications, such as RTP clients, could opt for a per-packet-signature approach, but this are choices made by the application developer. We will analyze the cost of signing packets in section 4.6.

As soon as the Data Packet is authenticated, it is stored in the *Data Packet Output Buffer*, which function is to store the Data Packets for matching incoming Interests. Since the size of this buffer can significantly increase, it likely cannot fit in the main memory: for this reason we designed it as a two layer cache, where the L1 cache has a limited size and it is stored in the main memory, while the L2 cache can be really large and is saved on permanent storage. The replacement policy of the L1 cache is a socket option defined by the application itself, and can be FIFO, LRU or LFU. If the application creates the socket selecting the `PROD_UNREL` protocol, the L2 cache is disabled and whenever a Data Packet is removed from the L1 cache is lost. This case is typical for live streaming applications, where the *Data Packet Output Buffer* needs to always store new Packets and discard the old ones. In the case of `PROD_RTC`, both the L1 and L2 caches are disabled and packets are directly pushed to the local ICN forwarder, for minimizing the latency by serving the packet directly from the content store of the local forwarder. At last, applications like a File Repository may need their data to be always available: in this case the L1 and L2 caches are required and the `PROD_REL` protocol has to be used.

When an Interest Packet is received through the forwarder connector's `recvPacket` API, the Producer Socket puts it in an input buffer and then tries to perform a lookup for it in the *Data Packet Output Buffer*: if a corresponding Data Packet is found, the socket replies directly by forwarding it to the connector through the `sendPacket` call. In this way the application does not perform any computation for the incoming request, as they are satisfied directly by the transport layer. If the output buffer does not contain any matching Data for the interest, the latter is registered in a *Pending Interests* table and

the socket signals to the application that the cache miss happened: the signalization is done through the `recvMsg` system call, by passing to the application the name of the Interest and other information (such as a possible interest Payload).

The application can then react to the cache miss by publishing the content required by the interest, with the production procedure described above. At the moment of storing the Data Packet into the *Data Packet Output Buffer*, one further lookup is done on the Pending Interests table for checking if it contains a pending request for the data that is going to be published: if the lookup succeed, the data packet is forwarded directly to the underlying forwarder connector. This whole procedure allows the socket and the application to jointly perform the dynamic production of content.

4.6 Performance Analysis

In the following sections, we report the performance evaluation of our transport layer. We show the performance we achieve using the per-packet signature and the manifest approach, reporting the average goodput as well as the communication latency. Moreover, we compare the performance of our implementation with the TCP implementation in the Linux kernel and the TCP stack implemented in VPP. Our goal is to show the state of the art performance of our transport layer using as a reference the today's transport layer.

4.6.1 Experimental settings

The experiments presented in this section are performed using the vICN framework, described in chapter 7. The setup is the following: two Linux Containers [18] deployed on a Cisco UCS Type-C server with an Intel(R) Xeon(R) CPU E5-2695 xv4 and 256 GB of RAM. The two containers manage an Intel 82599ES 10-Gbps NIC each and are connected together through a 10Gbps Cisco-Nexus 5k. The two NICS are installed in the PCI of two different NUMA nodes, in order to distribute the workload on 2 different CPUs and improve the memory usage. Processes running inside LXC, such as VPP and applications,

are run with CPU affinity to cores that are installed into the NUMA node belonging to the NIC VPP is using. This allows to achieve optimal performance in terms of memory access latency.

The hICN traffic is forwarded using the VPP forwarder augmented with the hICN plugin described in Section 4.5, and running inside both containers. The goal of this experiment is to measure the performance of our hICN transport, by forwarding traffic between the two containers. To this end, we wrote a simple application that sits on top of our transport library and provides statistics regarding the transport itself, just like iperf [16] does for TCP/UDP. To compare hICN transport layer performance with respect to TCP, we use default TCP in the Linux kernel[2] (TCP Cubic) and in VPP (TCP New Reno). We use the iperf3 tool [16] to test the performance of TCP[16]. The reliable transport service used in this set of experiments is based on [43], which implements delay based flow and congestion control.

In the following, we report a summary of an extensive experimentation campaign to benchmark the performance of our transport services. In particular, we first study the computational cost of the crypto operations required to compute and verify signatures, then we evaluate their impact on transport services. To this end, we consider different scenarios:

(i) distribution of static content that the producer publishes asynchronously, namely *Asynchronous publication*

(ii) communications in which the content is requests as soon as it is available in the application, namely *Synchronous publication*

In *(i)* the producer segments, names and signs every requested content a priori, i.e., upon receipt of each interest, the corresponding Data Packet is already available in the *Data Packet Output Buffer*. In *(ii)* the producer segments, names and signs the application content upon the receiving of the first interest for it. In all our experiments, the producer publishes (and the consumer retrieves) a content with size 200MB. Moreover, we use RSA

[2]Kernel version 4.4.0104

and ECDSA as signing algorithm, choosing a key size of 1024 and 192bits respectively [3]. Finally, we used SHA256 as cryptographic hash algorithm. The crypto library used for the crypto operations is openssl 1.0.2o [22]. All statistics reported below are obtained from 30 independent experiments and mean values are reported with a t-student confidence interval with 99% significance.

4.6.2 Results

Type of test	Average	99% CI
hICN Asynchronous Publication		
Manifest RSA-1024	928Mbps	[919 936]
Packet-wise RSA-1024	290Mbps	[283 297]
Manifest ECDSA-192	531Mbps	[523 538]
Packet-wise ECDSA-192	28Mbps	[27 28]
hICN Synchronous Publication		
Manifest RSA-1024	525Mbps	[518 532]
Packet-wise RSA-1024	26Mbps	[26 27]
Manifest ECDSA-192	530Mbps	[522 537]
Packet-wise ECDSA-192	28Mbps	[28 29]
hICN Crypto Operations disabled		
No signature	6.52Gbps	[2.43 2.46]
TCP - Iperf		
Linux TCP (w/ TSO)	9.19Gbps	[9.09 9.30]
Linux TCP (w/o TSO)	5.00Gbps	[4.88 5.12]
VPP TCP stack	9.24Gbps	[9.22 9.26]

Table 4.1: Average goodput of (a) hICN - Asynchronous publication (b) hICN - Synchronous publication (c) hICN - Asynchronous publication w/o crypto operations (d) TCP - Iperf3.

Table 4.1 shows the goodput of our transport. In both the Synchronous and Asynchronous publication scenarios, the crypto operations have a considerable impact on the goodput performance. Obviously, the Asynchronous publication offers better performance than the Synchronous publication as the signature is computed offline and it has no impact on the goodput. If we compare the per-packet verification with the manifest approach, the latter is able to provide a higher throughput. This is because, signing only manifests reduces the number of signature verification that a consumer has to perform (about 93%

[3]RSA 1024 and ECDSA 192 are considered to offer the same level of security.

reduction). In this case, the throughput is limited by the per-packet hash computation (about 140000 hash calculation - 1.3s), and the cost for the signature verification (about 4000): using RSA-1024 the latter is around 0.208s, whereas using ECDSA-192 is about 1.6s (cfr. Table 4.2). We want to stress that we are not exploiting any dedicated hardware acceleration to speedup the crypto operations or the segmentation operations. Following this approach will significantly improve the performance.

In any case, the crypto operations put a boundary on the goodput of the application: with our software implementation, in case of verification with manifest, RSA-1024 takes around 1.508s for verifying 140000 Packets of 1500 bytes, while ECDSA-192 takes 2.9s. This turns in a maximum goodput of approximately 1.1 Gbps for RSA-1024 and 580 Mbps for ECDSA-192.

The approach without manifests requires producers and consumers to sign/verify every data packet: in the case of asynchronous publication, verifying with RSA-1024 allows to reach an acceptable goodput of 290 Mbps; on the other hand, verifying with packet-wise ECDSA-192 drops the throughput to 28 Mbps. Synchronous publication with per-packet signature drops the performance both with RSA-1024 and ECDSA-192 to 28 and 26 Mbps respectively. These goodputs are still acceptable for real time applications, where the rate is low and using manifest could increase the latency.

At last, we compare our performance with the one obtained by TCP in the same condition. We highlight that the hICN transport services should not be directly compared with TCP, as the latter does not provide neither integrity nor data-origin authentication. Therefore, to have an idea of the gap between our prototype and TCP, we disable integrity and data-origin authentication and we report the goodput our hICN transport achieves. In this case, the kernel implementation of TCP, with hardware accelerations disabled, achieves a goodput equal to 5Gbps while our transport is able to reach 6.52Gbps. We stress that, at the time of writing, none of the existing hardware acceleration for TCP (e.g., TSO, GSO, GRO) is compatible with our hICN transport layer. Therefore, we believe it would be unfair to exploit hardware acceleration for TCP. Still, we report the performance of TCP exploiting the hardware acceleration for the sake of completeness.

Table 4.2 shows the time required to compute the crypto operations. We report the measurements considering two different cases:

(i) crypto-operations performed inside loop on a vector of packets

(ii) crypto-operations performed per packet with a frequency of 1s

In the first case, the cost of the crypto operation is lower than in the second case. The explanation for this behavior is the following: performing crypto-operations on a vector of packets requires the CPU to spend more time to complete this operations and therefore it is less likely that the kernel runs a different process on the same CPU. Therefore, L1 and L2 cache locality is improved and the number of cache miss is drastically reduced improving the performance on computing the crypto operations. On the other hand, processing one packet per second does not allow to exploit the CPU optimization described before, and this turns in a performance worsening. We also show how using jumbo frame helps in reducing the cost of computing crypto operation. The adoption of jumbo frames reduces the number of packets generated during the segmentation operation, thus lower the computational cost per byte of the hash calculation (from 0.006us with MTU=1.5kB, to 0.003us with MTU=9KB).

	Vector of packets		Single packet	
Consumer: Signature verification				
RSA-1024	52.2us	[51.5 52.9]	140us	[132 149]
ECDSA-192	412us	[406 417]	757us	[697 817]
Producer: Signature computation				
RSA-1024	440us	[437 443]	775us	[733 818]
ECDSA-192	380us	[377 383]	701us	[661 740]
SHA-256 hash computation on MTU packet				
1.5kB	9.44us	[9.38 9.50]	28.62us	[31.03 32.08]
9kB	31.55us	[31.03 32.08]	68.26us	[63.63 72.89]

Table 4.2: Cryto operations cost in case of vector of packets and single packet computation.

Table 4.3 reports the impact of the signature computation and verification on the end to end latency. Upon the interest reception, the producer creates a new 1500 Bytes Data Packet, signs it and sends it back to the consumer that verifies the signature. This is a typical communication pattern of real time application where the rate is low and the

application data can fit in one MTU packet (e.g. RTP). In this scenario, using a manifest is not the best choice, since the packets need to be forwarded as soon as the corresponding interests are received.

The delay introduced by the real time signature/verification of each packet is significant: with respect to the RTT measured without any crypto operation (145 us), the delay is one order of magnitude bigger (1173us for RSA and 1667us for ECDSA).

Type of test	Average	99% CI
No signature	145us	[136 155]
RSA-1024	1173us	[1142 1205]
ECDSA-192	1667us	[1621 1712]

Table 4.3: Average end-to-end latency growth in case of signature computation and verification.

4.7 Related Work

Most of the work on transport layer for ICN has focused on congestion control which is receiver-driven as opposed to the current sender-based TCP/IP model. Also ICN transport does not require connection instantiation and accommodates retrieval from possibly multiple dynamically discovered sources.

It builds upon the flow balance principle guaranteeing corresponding request-data flows on a hop-by-hop basis [7]. A large body of work has looked into ICN transport (surveyed in [129]), not only to propose rate and congestion control mechanisms [133, 166] – especially in the multipath case [43], [96], [142] – but also to highlight the interaction with in-network caching [42], the coupling with request routing [79, 43], and the new opportunities provided by in-network hop-by-hop rate/loss/congestion control [41, 159, 47] for a more reactive low latency response.

Other work on the description of a transport layer and a socket API has appeared in [61] and [105].

The current work builds upon [105] as it makes use of the consumer/producer API as a general framework for all transport services built on top of ICN. In addition to that we

create a fully fledged transport layer with optimizations and an API that can be easily inserted in today applications with or without name space optimizations. In addition, we evaluated the performance of such transport stack and the design trade offs that need to be considered for different applications.

Moreover, unlike [105], our socket API does not require to the application developer to implement signature calculation and verification, but offers them as a transport service. In the area of transport services and related API recent work at the IETF is considering the problem of providing and novel transport service API to replace the BSD like socket API. Relevant work is [118] and more generally the effort in the TAPS WG and the NEAT project [23]. [67] considers Hadoop on top of NDN by means of a Socket API that is very specific to this application, i.e. the ambition is not to provide a general transport service to any kind of application like in this thesis.

4.8 Discussion and Conclusion

This chapter has presented the design, implementation and benchmarking of a transport layer for ICN with a socket API for applications. The implementation is based on a fast transport stack in userspace with kernel bypass based on VPP and DPDK. The present contribution applies for hICN and potentially CCN/NDN, and is currently open sourced in the Linux Foundation project FD.io [6].

We believe that such a transport layer allows to easily insert ICN in today's applications such as web services and to develop new ones in a simpler way. The choice to build a transport stack in userspace allows to integrate new extension rapidly with little effort with no performance trade-off.

We have evaluated the transport layer in terms of performance and compared the implementation with the Linux TCP and the VPP TCP stack to have a baseline reference in terms of performance target for benchmark workloads. Experimental results in Section 4.6 show that the crypto operations have a significant impact on the overall transport performance, both in terms of application goodput and latency. This overhead can be

easily reduced by offloading them to dedicated hardware such as Intel OAT acceleration available in the core itself or in external chips. Performing them in software is expensive and impairs the long term scalability of the transport stack.

Even using a recent CPU technology, the delay introduced by crypto operations significantly restricts the maximum achievable throughput, and also the signature/verification performance becomes really system dependent.

The availability of hardware offloading techniques for the transport layer tends to be appear when needed. As an example, TSO/LRO is available already in any end device with Gbps interfaces, including laptops. The large usage of TLS in the Internet is also going to make several cryptographic accelerations available in silicon through most used crypto library, both for client end devices, including mobiles, and server ends. In this paper we have shown that a pure software implementation can still provide decent performance with several limitations. This reality check allows to claim that many applications, such as video delivery and real time communications, are feasible with an ICN software stack that is portable into a large set of end devices. The current design and implementation has been already used to enable ICN transport in HTTP and WebRTC, as will be presented in the next two chapters.

While the transport layer presented in this chapter constitutes a fundamental component for integration of ICN into today's and future application, we recognize that named data provides a larger degree of freedom in the way application namespaces are mapped into network namespaces. Future work is needed to make further progress to define an additional session layer, where application namespaces are multiplexed into multiple network namespaces.

Chapter 5

HTTP over hICN

5.1 Introduction

As continuation of the previous chapter, which elaborated on the architecture of a modular and scalable transport layer, this chapter and the next one will focus on how the transport services can be exploited for the development of ICN applications. Request/Reply pattern of the ICN transport do not always fit into the applications needs, in particular in case of existing protocols which are already designed to work with a host-centric paradigm. These two chapters will show how it is possible to overcome such limitation, by providing two solutions allowing existing applications protocols to work on top of ICN. The implementation of the components used in the experiment here is based on hICN, although the design is general enough to work also with any other ICN implementation, such as CCN/NDN. This chapter will focus on HTTP over hICN.

Hypertext Transfer Protocol (HTTP) is an application-level protocol for distributed, collaborative, hypermedia information systems. It is a generic, stateless, protocol which can be used for many tasks beyond its use for hypertext, such as name servers, distributed object management systems and video distribution (MPEG/DASH).

The support of HTTP is crucial for hICN deployment and diffusion, due to the huge amount of application working on top it. Although semantic of the HTTP protocol itself is Request/Reply, single HTTP messages are pushed from client to server, since HTTP

assumes the underlying transport to be push based: messages are sent from the sender to the receiver. For this reason HTTP cannot be directly deployed on top of ICN. The contributions of this chapter can be mainly summarized in two points:

- We developed a new transport service able to provide push semantics on top of hICN, which can be exploited by applications, still keeping the well known properties of ICN/hICN.
- We evaluated at scale HTTP on top of this new transport service, developed within the framework described in the previous chapter.

In the fist part we describe the architecture of the transport service providing push semantics to hICN, and we see how to map HTTP messages on top of it. Then we will evaluate the benefits of our design and implementation with respect to a classic web service deployed on top of TCP/IP.

5.2 Reverse pull transport service

The design of the reverse pull transport service is inspired to [104], which shows how HTTP semantics can be achieved on top of ICN. In particular the *reverse pull transport service* makes use of a routable prefix both for the HTTP client and server. Its purpose is to allow protocols built on top of a transport providing push-based semantics (e.g. TCP) to work with the pull-based hICN transport.

The main idea is that a client (e.g. HTTP client) signals to a specific server (or to a set of servers) the fact that it is publishing data with a certain name. In this way the remote server can retrieve the content just published by initiating a reverse pull triggered by this signalization (fig. 5.1).

This requires server and client to be consumer and producer at the same time: more specifically, it means that the reverse pull transport service must instantiate two sockets: one consumer socket and one producer socket. The steps required for performing

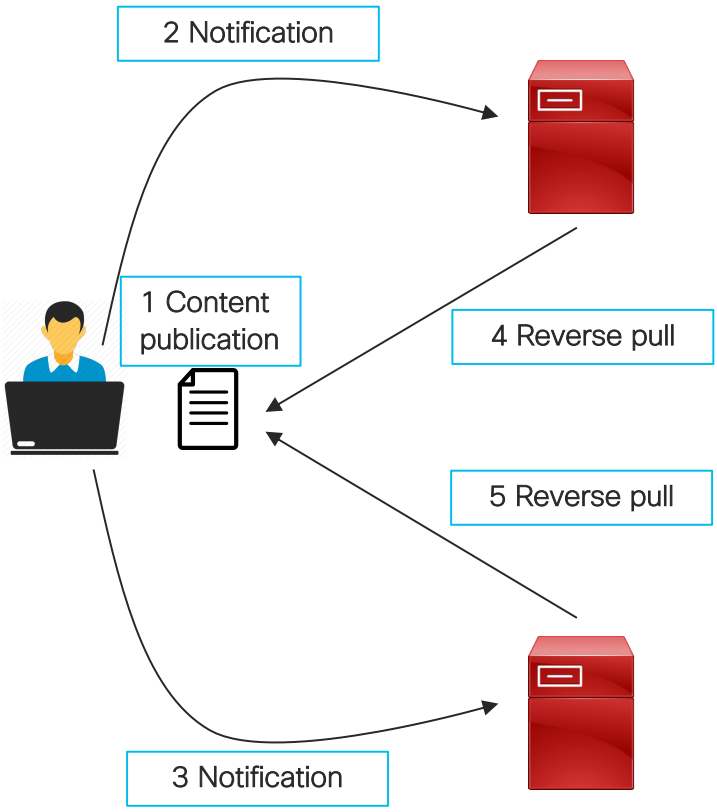


Figure 5.1: Example of reverse pull from one client to two servers.

a generic reverse pull operations are described in the following section. Section 5.3 will then elaborate on how optimize it for HTTP. The transport framework opensourced at [6] and [149] provides an implementation of the optimized protocol for HTTP, as well as the generic one.

5.2.1 Initialization

Servers are listening on a certain prefix used for receiving *reverse pull notification* from the clients. A client can use this prefix for initiating a connection with the server, eventually randomizing the hICN suffix for avoiding interest aggregation in the intermediate routers. The connection initialization consists in two messages: the **Initialization Message** and its **Acknowledge** (fig. 5.2).

The initialization message consists in a *signed* interest containing in its payload the following parameters:

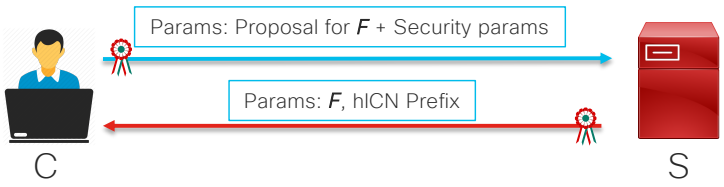


Figure 5.2: Reverse pull: Initialization phase.

- The locator of the key used for signing the interest
- A proposal for a function to be used for generating an identifier of the content the client will produce.

The server, upon the reception on this message, performs the following operations:

- It checks whether the interest is originated by a trusted client.
- It checks if the function proposed by the client can be used or not, and if not checks for an alternative function to propose.

If the interest was issued by a trusted client, the server responds with an ack, containing the hash function and the hICN prefix to use for generating the names of the next Interests. As an example, the prefix can be `b001::/64` and the function may be the concatenation of the prefix and the cryptographic hash of the data to be pushed to the server. The analysis of what function is the best to use is not part of this thesis and it is left for future work.

5.2.2 Naming

Before a client sends a Publish Notification to the server, it needs to choose a name for the content to produce. This name can be any name provisioned by the network service described in section 3.2.2.

5.2.3 Publish Notification

After the content publication, the next step of the reverse pull is called Publish Notification. Its purpose is to signal to one (or potentially many) servers the fact that a new

content to be sent to them is available. In particular, as shown in fig. 5.3, a given client sends to a server one Notification Interest, which is a normal interest carrying in its payload the hICN manifest indexing the content the client would like to push to the server. The control Interest is signed with the same key the client used for publishing the data to be pulled by the server, and its name is generated by using the function F that Client and Server agreed on during the initialization, applied over the prefix sent by the server and the data to be pushed itself. Notice that this function can be re-negotiated at any time during the Client-Server communication.

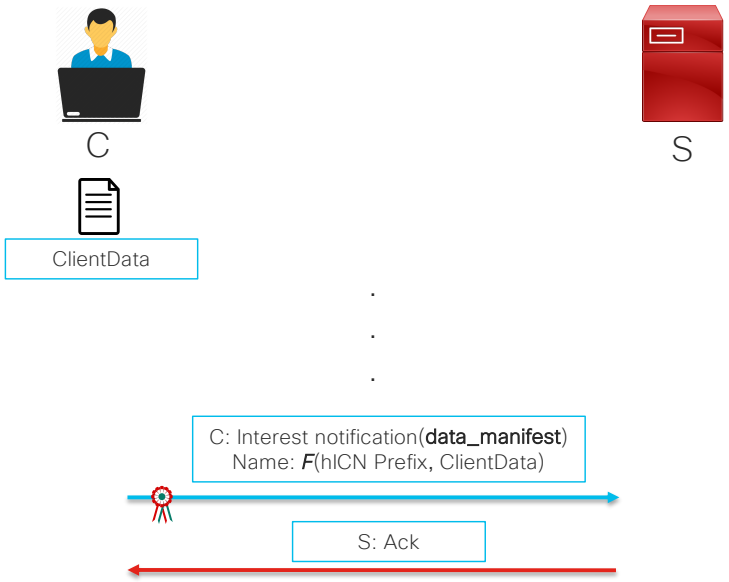


Figure 5.3: Reverse pull: Notification phase.

Once the server receives the control interest, it replies with a single hICN Data Packet, which acknowledges the interest reception.

The client continue to push data to the server by publishing them in the local producer socket cache (cf. section 4.5.4) and then notifying the server about the availability of new Application Data Unit (ADU), which will be then retrieved by reverse pull. The new ADU availability notification is performed as before, using a Notification Interest carrying the manifest indexing the frame just published.

5.2.4 Additional considerations

The paper [104] discourages the use of this approach because of its impact on the consumer mobility and because it may make hICN services vulnerable to reflection attacks. However, [30] proposes a really effective solution for the producer mobility and at the same time, reflection attacks can be avoided with the client signature on the Initialization/Notification Interests.

5.3 Optimized Reverse pull for HTTP

The general concept of reverse pull addressed in the previous chapter can be further enhanced in presence of request-reply protocols such as HTTP, where often different clients issue requests for the same web object. This can be done by conveying the semantic of the HTTP request into the hICN name used by Clients during the Notification phase. This would allow, when possible, to directly map HTTP Requests to HTTP Responses through the hICN name itself, and may prevent the server from pulling the request for the same object all the times from every client, as it currently happens with TCP/IP.

For achieving that, the server enforces a specific function F to the clients, which takes into consideration *only* specific parts of the http request. As a simple example we can consider the hash of the URL (without the query string) to be common across all the clients requesting the same resource. Unlike, we can consider the User-Agent header as something which is client specific. By tuning the part of the request the function F must consider, the Server can force many clients requesting the same resource to use the same hICN name, by mapping (for instance) the URL to the hICN name itself.

The first time the Server receives a Notification Interest with a given name from a client, it pulls the request through the reverse pull and uses the same name of the Notification Interest for publishing the manifest of the corresponding response. This name has been forced by the server itself through the function F sent during the Initialization phase, fig. 5.2. As an example, function F may concatenate the server prefix with the hash of

the URL contained in the HTTP request.

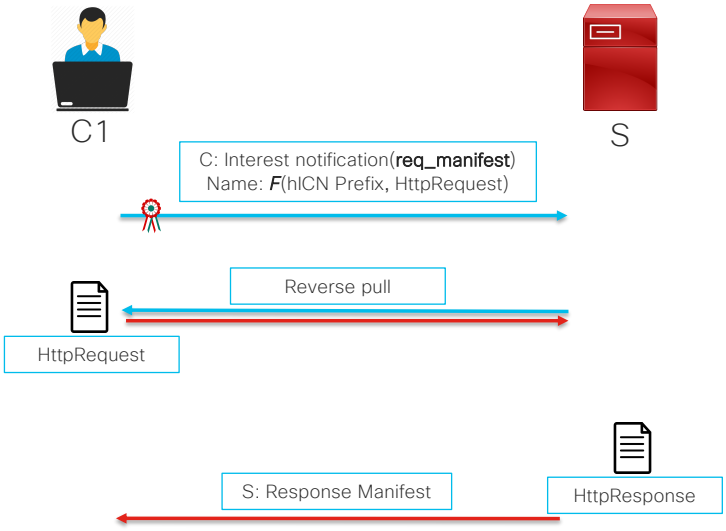


Figure 5.4: Reverse pull: Notification phase for HTTP.

Subsequently, a second client performs another Initialization to the server, which forces the same function and the same prefix to it. As soon as the second client tries to get the same web object, it will use the same function enforced by the server, in combination with the same prefix and the same URL. The name of the Notification Interest will be then the same used by the first client.

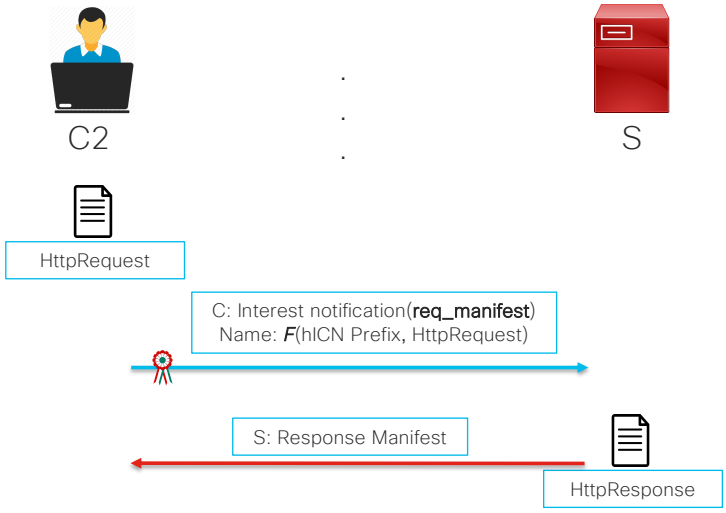


Figure 5.5: Reverse pull: Retrieval of already published response.

When the Notification Interest sent by the second client will reach the server, it will hit the HTTP Response previously produced for the first client, which has been cached in the Producer Socket output buffer (section 4.5). The hICN transport will then reply directly

to the interest using the previous response, without any additional computation required by the HTTP Server for producing a new Response.

This improvement works really well in the case of content distribution, where a single HTTP server is serving a given population of clients which are requesting the same web objects. An important example is DASH linear video broadcasting, that will be also used in section 5.5 for assessing the scalability gain of this approach.

5.4 Multipoint-To-Multipoint communication

The technique presented before can be further extended in order to achieve a multipoint-to-multipoint communication pattern. In the previous section we showed the fact that a single server can serve many clients in a *multicast* manner, by publishing an ADU once and using it for serving all the next additional requests.

In the same way, a single HTTP client can push one single HTTP request to several servers, *by publishing it once*. In fact the client can send a Notification Interest to *several servers*, which will reverse pull the client's request in a multicast fashion. The application itself will copy the content *just once* from the application to the producer socket. Instead, a normal TCP/IP server would require to copy the same HTTP request to each socket connected to a different server.

This technique can be exploited in several applications, such as multi-write distributed databases, in which a number of entities write simultaneously to many replica. More in general, the reverse pull mechanism plays a fundamental role in each application patterns that do not directly fit with the request-reply ICN communication and require a push mechanism, e.g., REST-based and pub-sub applications.

5.5 Evaluation of HTTP over hICN: Multicast and Server Load

This section will evaluate the design of HTTP over hICN in the case of Linear Video Distribution, which is a challenging test case as it requires provable QoE in terms of video quality and application responsiveness, and is also supposed to scale to a very large number of watchers.

Nowadays, one of the main limitations of the linear video distribution system are the server endpoints: the fact that a server needs to maintain a stateful TCP session per user does not scale when the number of users increases exponentially (think about popular sport events followed by millions of user). To cope with this, content providers use to load balance the clients requests among several instances of the same server for dealing with the problem of keeping a large number of TCP sessions open.

As described in the previous section, the hICN sockets in combination with the reverse pull transport service allows to serve/retrieve data to/from multiple destinations/sources. This allows to significantly reduce the server load, which can produce data once, instead of sending it to each client in unicast. Client's requests can be directly served by the hICN transport at the server once the data is made available after production. This section shows two key benefits of hICN for linear video distribution at scale:

- using hICN, **the server load scales with the number of channels**, rather than with the number of connected users as in TCP/IP;
- hICN deployment at the endponts and at the network edge yields to **traffic offload** not only at the server, but also in IP network core.

We consider the topology in Figure 5.6, where a cluster of video clients is connected to a video server distributing live video through MPEG/DASH. We use Linux Containers [18] with Ubuntu 16.04 for running client and server applications. The client application is the VIPER video player used for the experiment in section 3.4, which is available in the

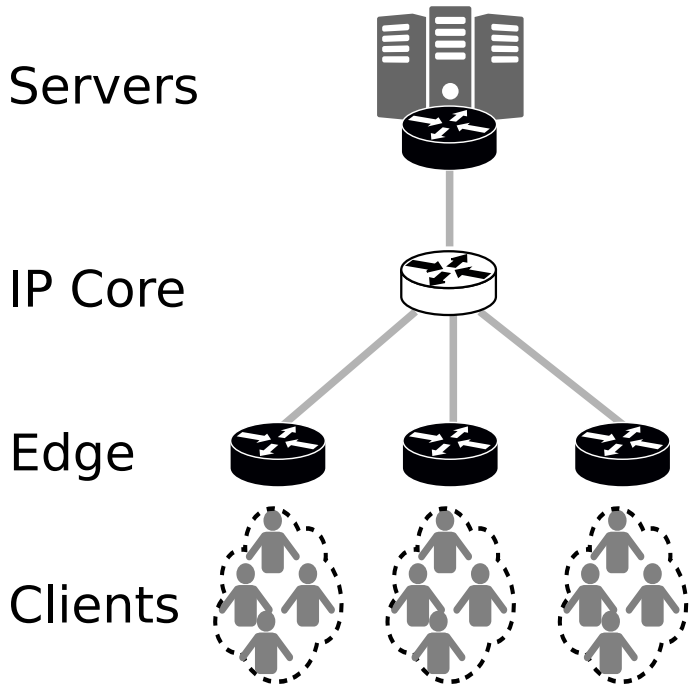


Figure 5.6: Video distribution network

Fast Data project [149]. Every client is connected through a 50Mb/s link to the edge routers, while the other links in the topology are 10Gb/s links. The video server is an Apache Traffic Server (ATS) [13], configured as an HTTP reverse proxy with a 2GB cache (1GB of RAM cache and 1GB of raw device cache). The content source consists in a live video feed sent by the Open Broadcaster Software (OBS) [21] sending a RTMP stream to a *nginx* [14] server providing HLS streams through the `nginx-rtmp` [20] module. ATS proxifies requests of the clients towards the nginx server, and stores in the HTTP cache the corresponding responses.

We implemented a plug-in that uses hICN sockets to interface ATS with the hICN VPP forwarder (cfr. section 4.5.1). Every hICN VPP forwarder in the topology has 750MB packet cache size. We run 2 hours experiments with TCP/IP and with hICN under different client population (specifically 100, 200 or 300 clients). Each client requests one of 48 available channels (encoded in a single video quality) and switches to a different channel every 10 minutes. Channel selection follows a Zipf distribution with $\alpha = 1.4$. The experiment has been deployed using vICN, described in chapter 7.

In the experiments, we considered three different scenarios:

(i) called *TCP*, in which every router and endpoint uses TCP/IP;

(ii) called *hICN endpoints*, in which hICN is deployed only at the server and clients;

(iii) called *hICN endpoints + edge*, in which hICN is deployed both at the endpoints and in the edge routers;

Table 5.1 compares the load at ATS in scenarii (i)-(ii) under the three client populations.

metric	N=**100** [C=**22**]		N=**200** [C=**30**]		N=**300** [C=**35**]	
	IP/TCP	**hICN**	**IP/TCP**	**hICN**	**IP/TCP**	**hICN**
r	743.4	215.7	1447.5	312.8	1963.0	358.7
hit	294.8	0	614.4	0	768.6	0
miss	448.5	215.7	832.2	312.8	1192.1	358.7
mem	1313.7	47.8	1484.6	47.1	1522.6	58.2
$\overline{\textbf{cpu}}$	9.6	6.1	16.7	6.1	21.4	6.4

r: #requests ($.10^3$), *hit*: cache hits ($.10^3$), *miss*: cache misses ($.10^3$), mem: total memory (MB) and $\overline{\text{cpu}}$: avg cpu usage (%)

Table 5.1: ATS performance metrics with N clients. C is the average number of channels being watched.

Results show that using TCP/IP, the total number of requests handled by ATS grows linearly with the amount of clients, while using hICN it grows linearly with the number of channels being watched. The reason for such different behavior lies in the content awareness brought by ICN at network layer and in the content-based rather than host-based communication model: requests for the same channel are directly satisfied by the hICN forwarder or the transport layer, reducing the load on the application. This is also confirmed by the absence of hits in the ATS cache using hICN, since duplicated requests do not reach the application. Considering that in a real deployment the difference between the number of actively watched channels and the number of total watchers can differ of many orders of magnitude, hICN scalability benefit may be significant. Moreover, hICN considerably reduces memory and CPU consumption in the system. Quantitative results are reported in Table 5.1.

Figure 5.7 compares the memory consumed by TCP and by hICN sockets for handling a single video channel with different number of watchers. The considered scenarios are (i)-

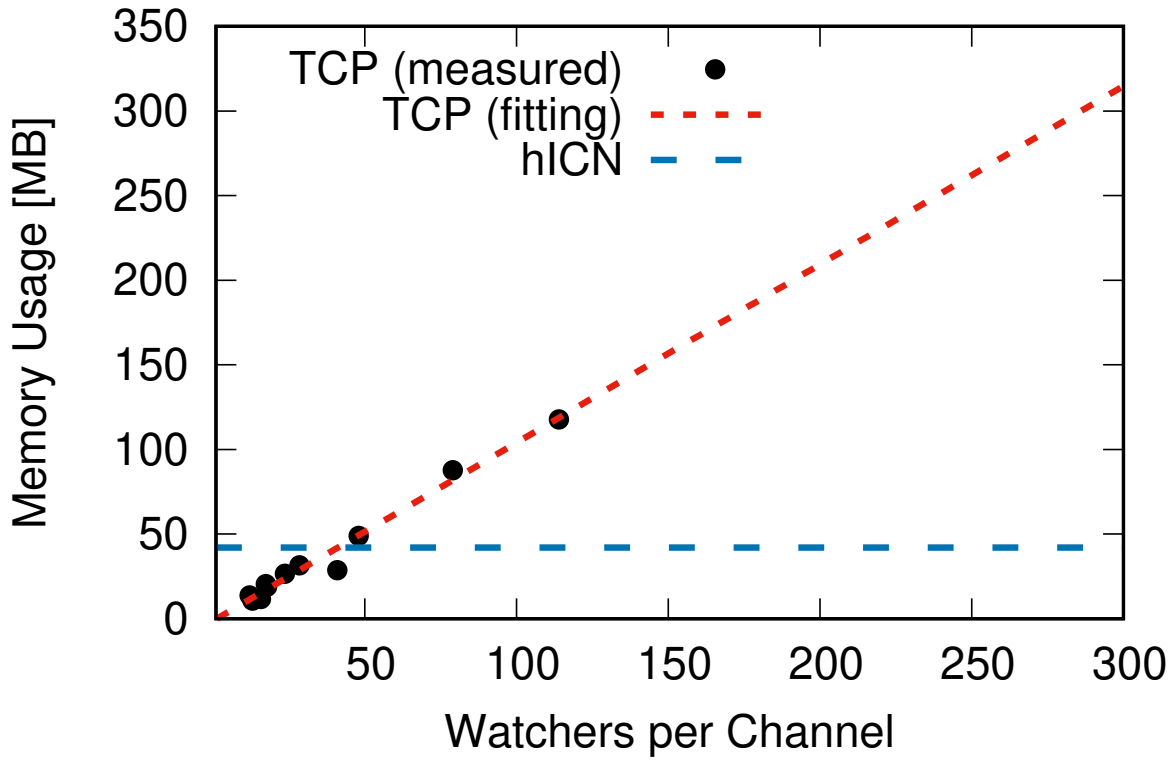


Figure 5.7: hICN and TCP memory used by a single channel increasing the number of watchers.

(ii), as for Table 5.1. Each dot reports the memory consumed by the kernel to handle TCP sockets opened by each client. We measure the memory used by the sockets exploiting Linux proc filesystem (/proc/net/sockstats). The red dashed line shows the expected memory consumption of TCP when increasing the number of watchers. Such line is obtained fitting the TCP memory consumption we measured in our tests. The blue dashed line reports the memory cost of hICN socket to handle one video channel. This value corresponds to the amount of memory reserved in the hICN producer socket output buffer (section 4.5.4), for each channel. As expected, results show that the memory required by TCP increases with the number of clients, while the memory required by the hICN socket remains constant. In fact, hICN socket does not maintain per-consumer connections, while TCP requires one socket for each connected client, so increasing the memory requirement.

Figure 5.8 shows the additional improvement in terms of IP core traffic offload provided by hICN enablement at edge routers. The test considers 300 clients. The figure reports

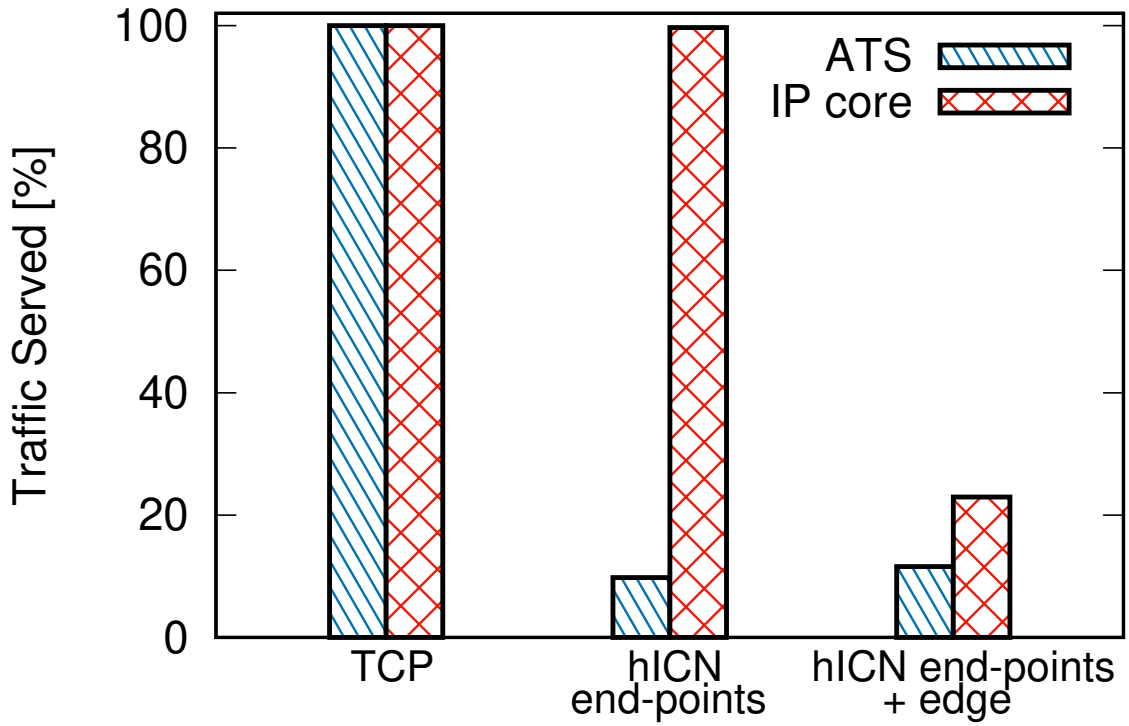


Figure 5.8: Percentage of the traffic served by each component in the network.

the percentage of total traffic received by clients that is served by ATS (in red) or by IP core network (in blue), respectively. The plot confirms what observed in Table 5.1: the request aggregation feature of hICN allows to reduce the amount of traffic served by ATS in scenario (ii) (*hICN end-points* case) w.r.t. the traffic served by ATS in scenario (i) (TCP/IP case). Deploying hICN also at the edge routers (scenario (iii) - *hICN end-points + edge*) reduces network traffic in the IP core network. It is worth noticing that IP core traffic is higher than the traffic received at ATS because of the considered network topology. The three hICN routers aggregate client requests and receive traffic independently from the server, possibly introducing multiple copies of the same interest/data packets.

5.6 Related Work

In the past, there has been some efforts for deploying HTTP on top of ICN.

[158] proposes a first attempt on adapting the HTTP semantics on top of ICN semantics. The solution described here requires the deployment of HTTP-ICN proxies to translate

HTTP messages to ICN interest-data and it is not compatible with hICN since it maps almost directly the HTTP URL to an URL-based ICN name. The work at [98] addresses the problem in a similar way, through IP-NDN proxies. We believe that a solution based on proxies is the best choice as intermediate step, but it does not represent a final solution, in particular in case of encrypted traffic. In case of HTTPS, with end-to-end encryption, the proxy has no way to decrypt the content.

[104] examines multiple diverse approaches to run modern Web-like applications over ICN, discussing advantages and drawbacks of each of the proposed approaches. As already discussed at the beginning, this work served as starting point for developing the solution presented in this chapter. However, this paper does not present an experimental evaluation of the proposed solutions.

Chapter 6

RTC over hICN

6.1 Introduction

The solution presented in the previous section works well in case of live linear video distribution, which is not limited by stringent low-latency requirements. In case of real time applications, with extremely low latency budget, the reverse pull exploited for MPEG/DASH video distribution is not a valid solution anymore. For addressing such requirements, we need a new transport protocol tailored for real time applications, as well as a new architecture design able to exploit ICN for scaling. More specifically, for evaluating RTC over ICN we will use hICN.

Traditionally used in collaboration systems, WebRTC is increasingly adopted by media companies for real-time services such as live eventing, gaming, betting or auctioning. 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, which are bounded by the chunk granularity at which they operate and thus struggling with low-latency objectives. However, unlike modern CDNs, WebRTC distribution model is not designed for large scale. To improve scalability, the architecture of traditional multiparty conferencing tools has moved from P2P, to centralized Multi-point Control Unit (MCU)-based architecture and, more recently, toward a lighter and possibly decentralized Selective Forwarding Unit (SFU)-based architecture. 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, the use of Simulcast [73] is becoming increasingly popular.

This chapter explores the potential benefits resulting from the use of WebRTC over hICN for real time content distribution, in terms of scalability and efficiency benefits. ICN appears suitable to support RTC application, as partially confirmed by initial work within the ICN community [50, 84, 170].

The contributions can be enumerated in two points:

(i) We design and implement hICN-RTC, an integrated WebRTC over hICN system.

(ii) We propose the hICN-RTC synchronization protocol as a new transport service, allowing the use of the hICN pull-based transport without introducing additional latency to the content distribution.

Preliminary results shows that hICN-RTC scales with the number of active speakers rather than the total number of users in the conference, allowing calls with 10 times more participants with respect to WebRTC, using 118 times less resources.

The rest of this chapter is structured as follow: Section 6.2 describes hICN-RTC and the benefits of the proposed architecture. Section 6.3 presents the hICN-RTC synchronization protocol. In Section 6.4 we show the results of our experiments. Finally, Section 6.5 describes the related work and we conclude in Section 6.6.

6.2 hICN-RTC Architecture

hICN-RTC builds upon the standard WebRTC SFU-based architecture and adapts it to hICN by integrating hICN both at clients and at SFU. In hICN-RTC, clients and SFU, referred to as *Hybrid Forwarding Unit* or HFU, are composed by:

(i) the WebRTC application logic (e.g., encoding/decoding audio and video flows at the clients and forwarding streams without any transcoding operation at the HFU)

(ii) the hICN transport layer to carry flows between clients and SFU in hICN packets

(iii) the hICN forwarder operating on hICN packets.

6.2.1 hICN-RTC Communication Model

In a SFU/HFU-based architecture, each client can communicate only with the central node and not directly with the other participants. In this scenario, we define the terms *contribution* and *distribution*. Contribution is the connection between the clients and the HFU, where the HFU collects the streams from the participants, while distribution is the connection from the HFU to the clients, where the HFU distributes all the streams of the call to the participants. In hICN-RTC we use two different naming schemes for the contribution and the distribution. In this way we reflect this separation also at the network level, where the requests coming from the participants will be routed to the HFU, while the requests from the HFU will reach the right participant.

On the contribution, the HFU asks the streams from the users using a participant specific prefix, e.g. */participant-1/video* or */participant-1/audio*. We refer to these participant specific prefixes as *contribution prefixes*. Instead, each participant uses the *distribution prefixes* to retrieve the content from the HFU. The distribution prefixes are in the form */call-m/active-speaker-1/video*, . . . , */call-m/active-speaker-n/video*, and they indicate the streams coming from the N most active speakers in the call. We identify the "speech activity" of a participant using the audio level information in the RTP packet [80] and we rank them using the last-N algorithm [72]. The prefixes are exchanged when a new participant joins the call: the new comer sends the contribution prefixes to the HFU while the HFU sends the distribution prefixes.

A prefix in hICN-RTC identifies a flow (video or audio) composed by a sequence of RTP packets generated by the application. A single RTP packet is carried as a payload of a hICN Data packet and it is identified by a full name, such as */participant-1/video/seg=50*.

Figures 6.1 and 6.2 show how the nodes in hICN-RTC exchange the flows using the described naming scheme. Because audio and video works in a similar way, we describe only the video streams. Figure 6.1 shows the contribution in a call with three participants

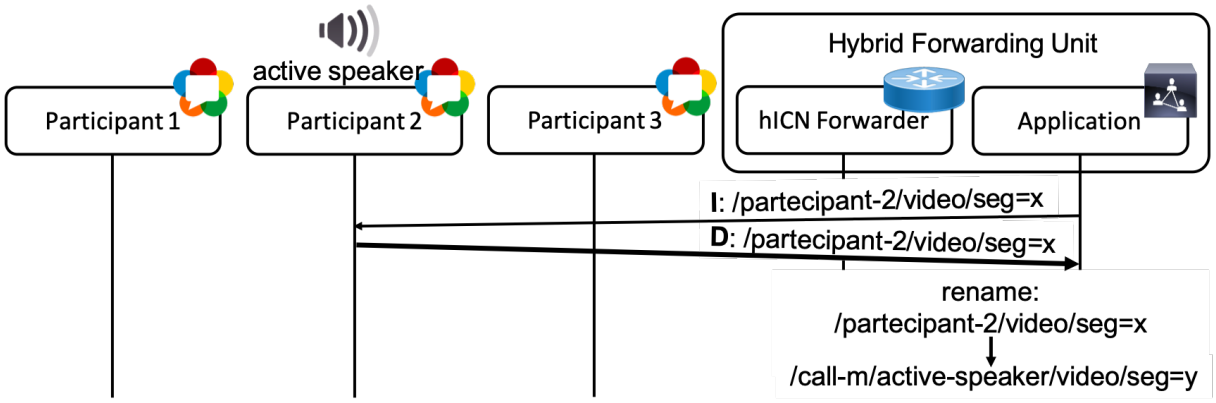


Figure 6.1: Contribution content exchange

and one active speaker (Participant 2). The active speaker is the only one that will be displayed in this example call. The HFU needs to retrieve the video from Participant 2 in order to distribute it to all the other users. To request the video the HFU uses the contribution prefix that identify the Participant 2. In the figure we see that the HFU requires the content sending Interests with name */participant-2/video/seg=x*.

When the HFU receives the packets from Participant 2, it renames the packets using a distribution prefixes, e.g */call-m/active-speaker/video/seg=y*, as shown in Figure 6.1. The HFU publishes the packet with the new name on the distribution link. The renamed packets are downloaded by all the participants using the distribution prefixes (see Figure 6.2).

The renaming of the Data packets at the HFU is a core component of hICN-RTC. Unfortunately this action invalidates the signature in the Data packet, breaking the Data integrity properties guaranteed by hICN. A simple solution would imply the HFU to re-sign each Data, but this would be too costly. To solve this problem, we introduced the concept of *mapping manifest*. A mapping manifest is a Data packet that describes the changes done by the HFU on the packets, in particular it maps the new name to the old one. When a participant receives a packet that it is not able to verify, it asks the mapping manifest to the HFU: using the mapping manifest the consumer is able to replace the name in the packet with the original one and it can verify the signature.

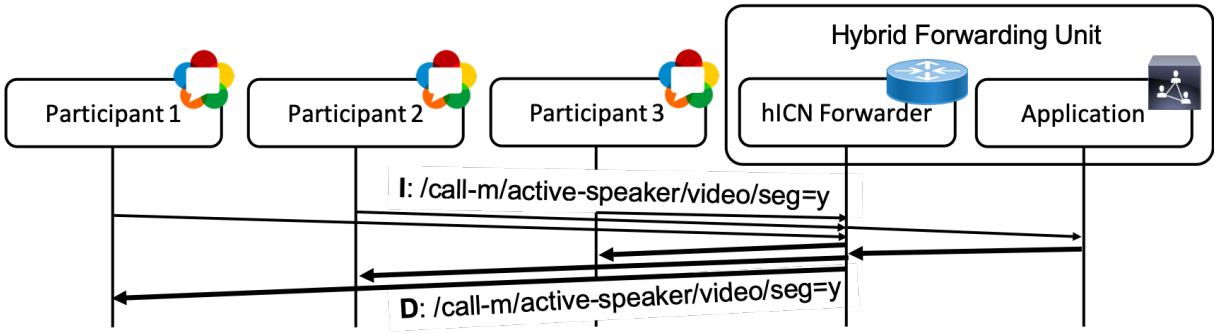


Figure 6.2: Distribution content exchange

6.2.2 hICN-RTC Advantages

In hICN-RTC the HFU pulls only the streams from the active speakers, which are those to be distribute in the call. This is visible in Figure 6.1 where the HFU pulls the video only from Participant 2. In contrast, a standard SFU usually retrieves the streams from all participants and drops those belonging to the non-active speakers. By pulling only useful streams, hICN-RTC reduces the bandwidth used in the contribution.

On the distribution, all the participants use the same distribution prefixes to download the content from the HFU. When the list of active nodes changes the HFU does not need to inform the participants, that will keep using the same names to get the content of the call. Using this approach the participant does not need to re-synchronize with the HFU and can always display a video without interruptions. We discuss about the synchronization problem between consumer and producer in the next Section. In addition, the requests from the participants can be aggregated and satisfied directly by the hICN forwarder in the HFU, as shown in Figure 6.2 where only one of the three requests reach the application: thanks to request aggregation, adding participants to the call the load on the application remains constant. Requests aggregation is the real advantage given by hICN that allows hICN-RTC to scale with the number of active-speakers instead that the number of participants.

6.3 Synchronization Protocol

hICN adopts a pull-based transport model which may potentially lead to additional latency suffered by consumers due to the fact that a full round-trip time (RTT) elapses between the emission of an Interest and the reception of the corresponding Data packet. To overcome such issue, the consumer keeps a window of pending Interest (Interests sent and not already satisfied) for Data yet to be generated by the producer. Using this strategy, as soon as a new Data is available it can be delivered to the consumer with no additional delay. In addition, a consumer needs to know which Data to request, namely the segment number of the newly Data packet generated by the producer. In this section we describe the hICN-RTC Synchronization Protocol that decides which Data packet to ask and how many pending Interests are needed for a client in order to get always fresh Data in a timely fashion.

Real Time Producer Socket: The producer side of the protocol has two main functions: (i) to encapsulate RTP packets received by the application inside hICN Data packets and to send them to the forwarder; (ii) to provide the consumer with the information needed to generate Interests for upcoming Data. Algorithm 6.3 details the actions executed by the producer.

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

Figure 6.3: Producer Side

Upon reception of an Interest packet, the producer executes the `OnInterest` function. The producer extracts the segment from the Interest name, as well as the Interest lifetime

and computes the maxSeg, which is the largest segment number that will be produced before the expiration of the received Interest. If the Interest segment number is smaller than currentSeg (namely the segment number to use in the name of the next Data packet) or larger than maxSeg, then the Interest refers respectively to a Data packet produced in the past or to be produced too far ahead in the future. If so, the producer replies with a negative acknowledgment (nack). A nack is a Data packet that contains the currentSeg and the current production rate of the producer (prodRate).

When the application generates a new RTP packet, the producer runs the `OnRTPData` function: 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 passed to the hICN forwarder to satisfy corresponding pending Interests.

Real Time Consumer Socket: The consumer side of the hICN-RTC Synchronization Protocol has two objectives: (i) to learn the current segment number used by the producer, and (ii) to adjust the window of pending Interests in order to avoid additional latency in media retrieval.

A consumer can be in two states: CATCH_UP or IN_SYNC. In the CATCH_UP state, the consumer tries to quickly estimate the number of pending Interests to send to the producer to match its production rate. Once consumer/producer are synchronized, the consumer switches to the IN_SYNC state, where tries to adapts the Interest sending rate according to the network variation, remaining in sync with the producer. At the beginning the consumer is in CATCH_UP state and starts requesting segment number 0; currentWin, that indicates maximum number of pending Interest allowed, and pendingInt, namely the actual number of pending Interests, are set to 1. The consumer is described in Algorithm 6.4.

`OnNack` function is executed upon reception of a nack. First of all, the consumer decrements pendingInt to free one space in the window that can be used to send another Interest. Then it extracts from the nack name its segment number, and the inProduction segment, that is stored in the nack payload. The reception of a nack indicates that

```
Function OnNack(nack)
    pendingInt -;
    segment = getSegment(nack);
    inProduction = getSegmentInProduction(nack);
    estimatedProdRate = getProductionRate(nack);
    nextSegment = inProduction ;
    if inProduction > segment then
        state = CATCH_UP; currentWin ++;
    else
        currentWin *2/3;
    end if
    scheduleInterest();
end
Function OnData(data)
    pendingInt -;
    sendContentToApp(getRTP(data));
    if state == CATCH_UP then
        currentWin ++;
    end if
    scheduleInterest();
end
Function scheduleInterest()
    while pendingInt < currentWin do
        sendInterest(prefix, nextSegment);
        pendingInt ++; nextSegment ++;
    end while
end
Function OnNewRound()
    if no Nacks in the last 3 rounds then
        state = IN_SYNC ;
    end if
    if state == IN_SYNC then
        currentWin = estimatedProdRate * estimatedMinRtt ;
    end if
end
```

Figure 6.4: Consumer Side

the consumer is out of sync with the producer. To quickly re-synchronize, the consumer sets nextSegment (the segment number used for the next Interest) equal to inProduction. If inProduction is larger than segment, the consumer is asking for old Data. If so, the consumer switches to the CATCH_UP phase and increases the window. In this phase the consumer increases currentWin exponentially (+1 at each packet reception), in order to quickly catch-up with the producer. Otherwise, the consumer is asking for Data that will be generated too far in the future. This means that the pending Interest window is too large and the consumer decreases it (by a factor of 2/3).

When a Data packet is received, the consumer socket executes the `OnData` function. As in the case of nack reception, the consumer reduces the value of pendingInt. Then, the consumer removes the hICN header from the Data packet, and sends the RTP packet to the application. If in CATCH_UP state, the consumer also increases the window size. An RTP packet is sent to the application even when the consumer is in CATCH_UP state. In fact, valid Data packets can be received and displayed even during this phase. For every packet received (Data or nack) the consumer also executes the `scheduleInterest` function that sends all the possible Interests allowed by the current window size.

The `OnNewRound` function is called periodically by the consumer, once at every round (200ms in our implementation). This function determines the state switch: if the consumer did not receive nacks in the last 3 rounds, it is considered to be synchronized with the producer. When the consumer is IN_SYNC, it also adjust currentWin, setting it equal to the estimated production rate of the producer (estimatedProdRate) times the estimated minimum RTT (estimatedMinRtt). This allows the consumer to adjust the window according to the network changes.

6.3.1 Protocol Discussion

The algorithm proposed in this section lacks of a congestion control mechanism: the consumer is forced to download at the same rate at which the producer is sending packets. This is not always possible and depends on the network conditions. To tackle this problem

we are currently studying a congestion control that combines Simulcast [73] and receiver-based adaptive bitrate selection [137]. Using these mechanisms the consumer will be able to switch between different bitrates and select the best one for the current network conditions.

In hICN the consumer is also responsible for recover packet losses. The hICN transport can perform fast-recovery without the need to rely on the RTCP Generic NACK [114] generated by the decoder. In additions hICN allows for specific in-network loss recovery mechanisms [47] that may reduce the need for more complex protocols such as FEC.

The synchronization protocol uses nack packets. This expose the producer to vulnerabilities, since each nack is a Data packet and must be signed: an attacker can send Interests that always generate nacks, forcing the producer to spend most of the time signing them. This problem was studied in [51] and we adopt a similar solution to mitigate this attack.

6.4 hICN-RTC Evaluation

To implement hICN-RTC we modified the code provided by the open source project Jitsi [3] and we implemented hICN-RTC clients as web applications, customizing WebRTC library inside Chromium [69]. The main modification inside WebRTC library is the introduction of the hICN sockets (cfr. section 4.5) in replacement of standard ones. hICN sockets, as well as the synchronization protocol are developed starting from open source code published in FD.io hICN project [6]. The HFU runs a slightly modified version of the Jitsi video-bridge: most of the application logic is reused, notably the speaker ranking mechanism [72] that has a central role in our architecture. In addition, we implemented the Data packet renaming described in Section 6.2 and we used the hICN sockets. As underling hICN forwarder, the HFU uses VPP [152]. In all the tests, the participants produce a single video stream (Simulcast is disabled) with an average bit rate of 500Kbps at 25 frames per seconds.

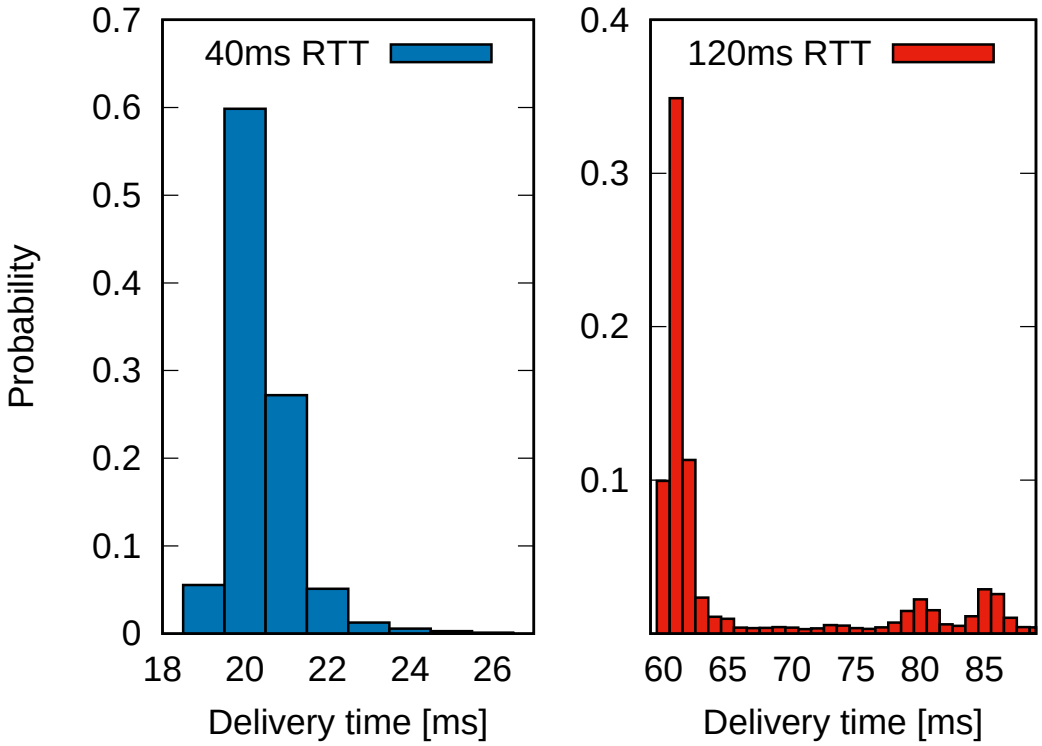


Figure 6.5: Packet delivery delay

Synchronization Protocol Benchmark

In this section we evaluate the hICN-RTC synchronization protocol proposed in Section 6.3. In our test we use a simple topology where four users are connected to the conference and we measure the time that elapses between the creation of a Data packet at producer and its reception at consumer side. We vary the propagation delay of each link in the topology using the netem qdisc [60]. The total RTT between two clients, passing through the HFU, is set to 40ms and 120ms. We collect data from a conference of 2 minutes. Figure 6.5 depicts the results. As expected, the majority of the packets are received within 1/2RTT. In the case of 120ms RTT some packets are received with a small addition delay. These are the packets taken from the HFU cache. A participant may retrieve content from the cache if it goes temporarily out of sync with the producer: the cache helps to mask this border effect and, even in this case, the application at the client is able to display the video with no interruption.

We also measure how much time it takes for a new participant to start receiving valid video packets from the conference: we measure the time elapsed between the sending of

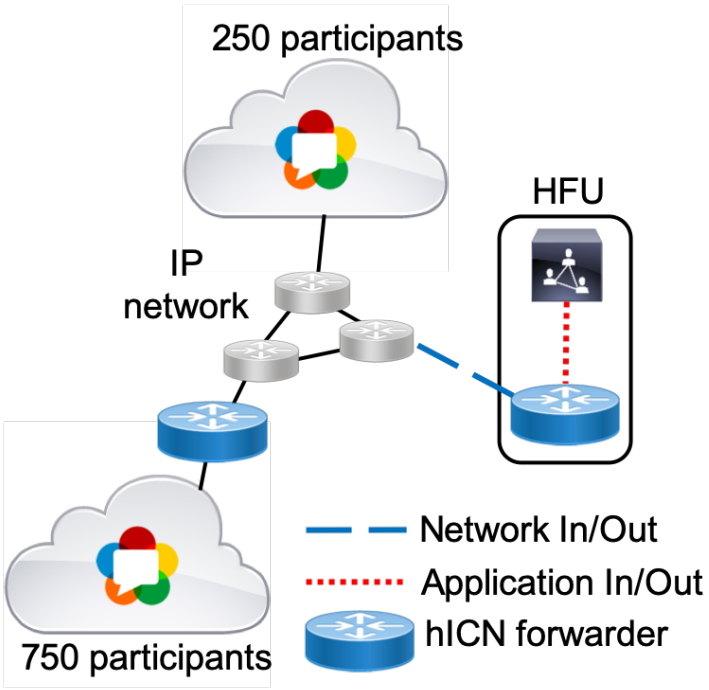


Figure 6.6: Topology

the first Interest and the reception of the first valid RTP packet. We notice that even increasing the total RTT to 200ms, the first packet is received in about 2RTTs, showing that the synchronization protocol is robust to the network delay and that it is able to quickly synchronize producer and consumer.

6.4.1 Scalability

We consider the topology in Figure 6.6. The upper cloud is composed by users (up to 250) that are connected directly to the HFU (no addition hICN router in the middle). The lower cloud (750 users) instead is connected to the HFU through additional hICN nodes. In all experiments, we progressively add participants to the conference, setting the number of active speakers equal to 3.

The workload for WebRTC experiments is generated using Selenium Grid [11]. We run multiple LXC [18] containers, each one containing an instance of Chrome and a Silenium node. Selenium nodes are controlled by a centralized orchestrator that runs Jitsi Torture [4]. The generation of the workload in this setting is quite heavy and we were not able to scale it to more then 100 users due to the amount of resources required. We used 5

servers with 36 Xeon E5-2695 v4 CPUs each, for a total of 360 cores using hyper-threading (about 3 cores for each Chrome instance) and a 6th server dedicated to the Jisti video-bridge. For the hICN-RTC workload we used a simple test application running over hICN, one for each participant. This application does not perform encode/decode operations, allowing us to run up to 1000 clients with 2 servers.

We generates two different workloads for WebRTC. In the first one, indicated as WebRTC in the plots, each participant sends its video to the SFU, the SFU discards the streams that are not needed and forwards the other. This is the standard way an SFU-based WebRTC architecture works today. In the second one, denoted as ideal-WebRTC, only the active speakers send their video to the SFU that replicates and forwards them and to all the other participants. To obtain this behavior in the ideal-WebRTC only 3 clients have the camera on. We use this second workload to show that the benefits of hICN-RTC are coming from the hICN requests aggregation, and not by the fact that we pull only the streams from the active participants.

The results of the experiments are shown in Figure 6.7, 6.8 and 6.9. Notice that all the plots have the y-axis in log scale. All the plots show statistics both at the application and at the network level. The values for the application are reported with solid bars, while the values related to the network are indicated with different patterns. Finally, the gray area on the right side of the plots highlights the results for the 750 participants connected to the HFU through an hICN network.

Figures 6.7 and 6.8 show the 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 with a blue dashed line in Figure 6.6. The application traffic is the portion of network traffic effectively received and sent by the application at the HFU, measured in the link highlighted with a dotted red line in Figure 6.6. This portion of traffic is indicated with a solid bar. For WebRTC this distinction is meaningless, since the application needs to process all the received traffic.

Distribution traffic, meaning the traffic from the clients to the HFU, is reported in Fig-

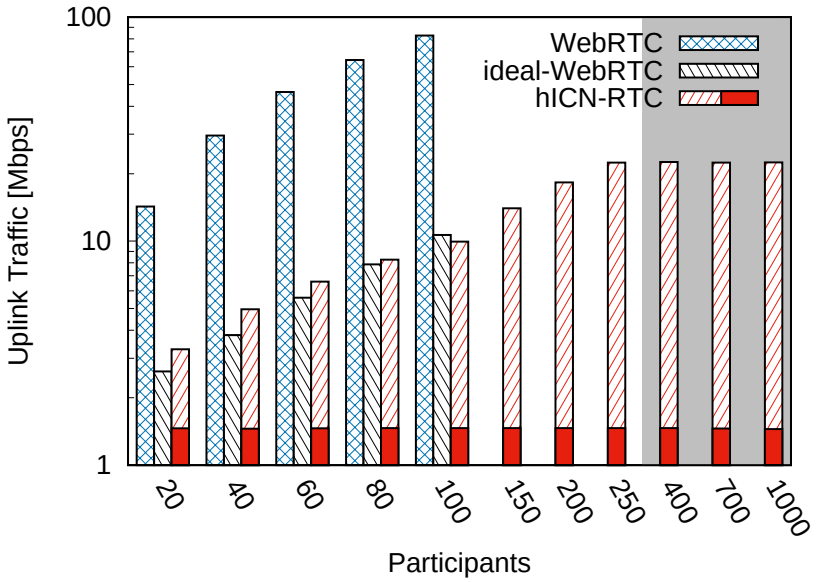


Figure 6.7: Contribution traffic.

ure 6.7. The figure shows that traffic generated by WebRTC is much higher than traffic in the other two cases, due to the fact that each participant sends its contribution to the SFU. For ideal-WebRTC and hICN-RTC the amount of data sent increases with the number of participants. This is due to control traffic (RTCP) in case of ideal-WebRTC and to the increasing number of Interests in case of hICN-RTC. It is worth noting that traffic generated by ideal-WebRTC is smaller than hICN-RTC in a conference call with few participants, whereas it becomes higher when we add more participants. This means that the overhead introduced by RTCP is higher than the overhead due to the Interests when the call is big enough. For hICN-RTC traffic reaching the application is always constant, regardless of the number of participants. The plot also shows that adding hICN capabilities to the network helps improving scalability even further: increasing the number of participants, not only application traffic, but also incoming network traffic remains constant, in virtue of requests aggregation performed by the intermediate hICN nodes.

Figure 6.8 shows the distribution traffic. From the figure we can see that, even if the three settings should generate more or less the same traffic, both WebRTC and ideal-WebRTC generate more traffic due to RTCP messages, which are not present in hICN-RTC. Notice also that the traffic sent by the application in hICN-RTC is always constant.

Figure 6.9 reports the CPU used in the central node by the three workloads. To estimate

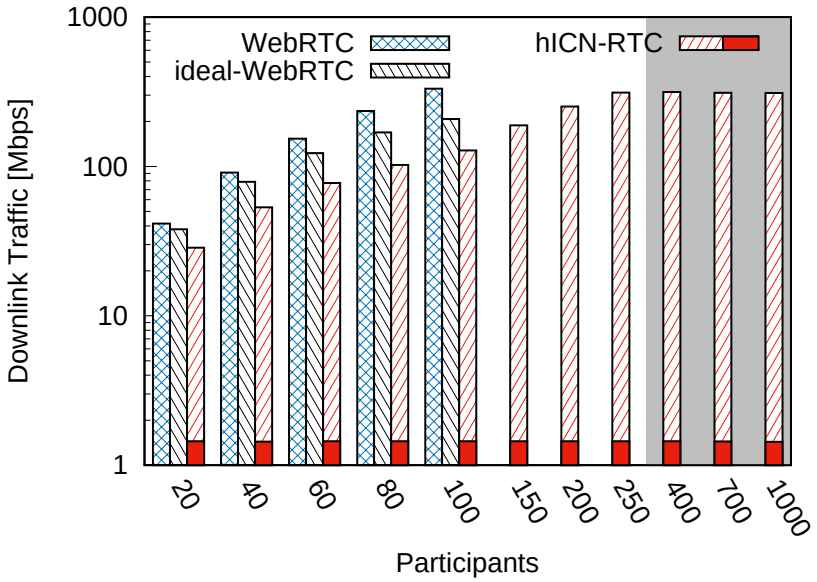


Figure 6.8: Distribution traffic.

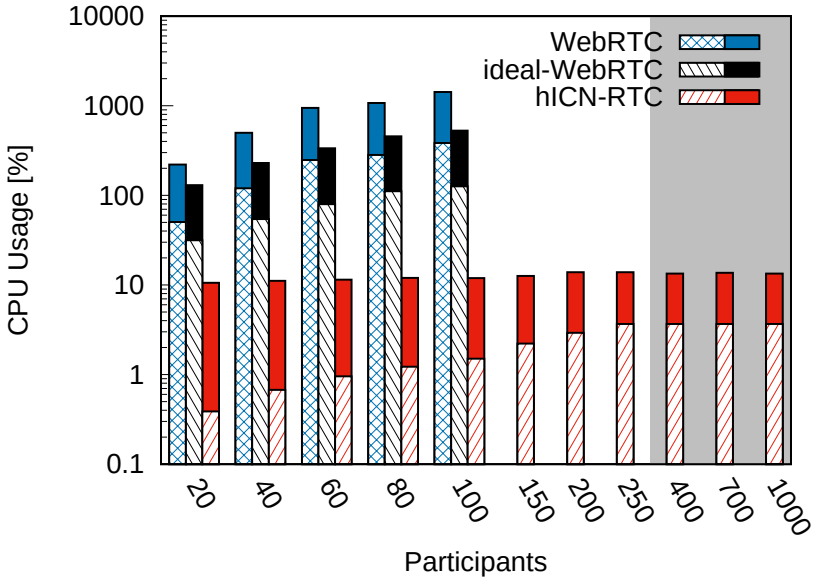


Figure 6.9: CPU usage.

the CPU required by the networking operations in case of (ideal-)WebRTC we estimated the CPU used by the Linux Kernel by monitoring the SYS percentage in the TOP command. The results in the plots are the average values measured during the 2 minutes experiments, from where we discounted the value measured at rest (no traffic). In case of hICN-RTC instead we estimated the CPU used by VPP. VPP always uses one full CPU core in order to prevent context switching. Values reported in the plot are taken measuring the number of CPU cycles required to process one packet: VPP requires on

average about 1300 CPU cycles to process a packet (Interest or Data) . We multiply this value by the number of packets processed per seconds to get the total cost. The percentage reported in the chart is the percentage of cycles used by VPP to process the traffic with respect to the number of CPU cycles available in a second. hICN-RTC outperforms the other architecture: it uses in total 12% of a CPU for 100 participants, 43 times less than ideal-WebRTC and 118 times less than standard WebRTC. This is due to the fact that the application in case of hICN-RTC always receives the streams only from the active speakers and send only one copy of them to the network, minimizing the number of send/receive packets. The network forwarder, VPP in our implementation, is in charge to replicate packets for all participants. In ideal-WebRTC instead the application needs to replicate streams, increasing the number of system calls having an impact both on the application load and on the kernel. In the standard WebRTC this problem is exacerbate by the fact that the application receives the streams from all the participants. Notice in addition that the CPU load remains almost constant in hICN-RTC. This shows that hICN-RTC is insensitive to the number of participants, whereas it scales with the number of active-speakers.

6.5 Related work

There have been some efforts in the ICN community to address real-time communication over ICN. VoCCN [81] and ACT [173] show the feasibility of real-time communications using ICN considering respectively two-party and multi-party audio conferencing. While they are both limited to audio conferencing, NDN-RTC [74] addresses directly WebRTC by making the underlying networking stack NDN-based. NDN-RTC also provides a first method to minimize latency using a pull-based communication framework. RDR [99] improves this mechanism introducing metadata packets, that inform the consumer about the current production state, in a similar way to the nacks in hICN-RTC. All the architectures described so far are based on the basic P2P model of WebRTC and so suffer for the same limitations.

Other work replaced the WebRTC-based approach with a fully ICN architecture trying to leverage monitoring at client-level [50, 84] and at network-level [170] to decide upon rate adaptation. In [50], 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. However, the proposed approach still remains host-only driven. In [84], 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.

6.6 Conclusion

We investigate scalability benefits of hICN-RTC, an integration of ICN in standard WebRTC. hICN-RTC builds upon a media-switching control architecture that exploits the underlying ICN stack for an effective media distribution at scale. A first evaluation of hICN-RTC confirms hICN suitability for support of real-time communications and shows significant improvements on scalability: hICN-RTC is insensitive to the number of users connected to the conference and scales with number of data sources. Requests aggregation and caching provided by additional hICN-enabled nodes in the network brings scalability even further. We leave for future work the design of a tailored receiver-based congestion control protocol, where users download media streams selecting among different bitrates according to the network condition. This will maintain the current scalability of hICN-RTC, while improving the QoE of the end users.

Chapter 7

Virtualized ICN

7.1 Introduction

The last part of this thesis aims to complement the work done in terms of deployability of ICN, by proposing a solution for programmatically deploying, configuring and managing ICN networks. In fact, the proliferation of ICN depends also on the capabilities of automating its deployment, from the network to the application layer.

Automating the ICN deployability has also the advantage of fostering testing and experimentation at scale and in real-world environment, helping ICN to evolve from a promising network architecture to a feasible deployment-ready solution. The ICN community has largely stated the importance of a pragmatic experimental and application-driven research approach since its inception (see e.g. [169], [130]).

Multiple tools and testbeds have been developed for simulation and emulation (CCNx, NDN software and testbed, CCNlite, MiniCCNx [40], MiniNDN). Most of them have been designed to assist research, specifically on design and evaluation of aspects of ICN architecture (e.g., caching, forwarding or routing). They operate in dedicated fully-ICN network environments, trading-off abstraction of network characteristics for scale and offering limited flexibility to modify core ICN features, network topology and settings, or application APIs.

In this chapter, we aim to complement existing tools with *vICN (virtualized ICN)*, a flexible unified framework for ICN network configuration, management and control that is able to satisfy a number of important deployment and experimentation use cases:

(i) automate the deployment, configuration and management of large ICN networks;
(ii) conduct large-scale and fine-controlled experiments over generic testbeds;
(iii) instantiate reliable ICN network with real applications in proofs of concept.

Clearly, requirements are different: research experimentation needs fine-grained control and monitoring of the network as well as reproducibility of the experiments. Prototypes for demonstration require a high level of programmability and flexibility to combine emulated and real network components or traffic sources. More than in previous cases, reliability and resource isolation is a critical property for deployments in ISP networks.

The operations required for the deployment of an ICN network include to install, configure, monitor a new network stack in forwarding nodes or the socket API used by applications at the end-points. If loading a network stack from an application store into general purpose hardware is easy to realize, a network stack has different requirements compared to a cloud based micro-service: ultra-reliability, high-speed and predictability, to cite a few. vICN shares the same high-level goals as SDN/NFV architectures, but with additional ICN-specific capabilities not typically required by IP services. Overall, we identify three main challenges that vICN addresses and that differentiate it w.r.t. state of the art:
(i) *Programmability*, i.e., the need to expose a simple and unified API, intuitive enough to facilitate bootstrap, expressive enough to accommodate both resource configuration and monitoring and flexible enough to allow the user to decide about level of control granularity. Existing software like OpenStack, which is built as a collection of independent components, each one following different design patterns, does not offer a satisfactory level of programmability.
(ii) *Scalability*: vICN aims at combining high-speed packet processing, network slicing and virtualization, and highly parallel and latency minimal task scheduling. Current systems are based on a layered architecture that prevent fine-grained optimization, thus limiting scalability on the long term.

(iii) *Reliability*: a fundamental property of vICN lies in its ability to maintain the state of deployment, recover from failures and perform automatic troubleshooting. This requires the overall software to be able to accommodate programmable function monitoring and debugging. In existing designs, each component has independent implementations to achieve that.

The remainder of the chapter is organized as it follows. Section 7.2 summarizes the state of the art, before introducing vICN architecture in section 7.3 and its implementation in section 7.4. Section 7.5 provide concrete examples of vICN in action. Section 7.6 concludes the chapter.

7.2 Related work

Salsano et al. [135] proposed to introduce network virtualization for ICN networks through the use of Openflow. In [126], the authors address a similar issue and propose an architecture to perform network slicing. However, [135] does not consider any network slicing technology, and [126] misses the aspects of network management and control. Mininet [102] makes a step in that direction and sits closer to our objectives as it enables the creation of virtual networks based on containers and virtual switches. Application performance can be tested in emulated network conditions by setting parameters such as link delays and capacity, node CPU share, etc. However, it does not propose any slicing mechanism, and lacks support for wireless or any control on applications or workload.

Interesting tools have also emerged from the testbed community. Emulab is a network experimentation framework joining emulation facilities with physical testbeds, but it lacks support for wireless topologies and offers no control over the network resources. NEPI [88] is maybe one of the most polyvalent tools. It hides all the complexity under a uniform programming interface. NEPI however lacks some control granularity and specializes in the management of resources provided by testbeds, assuming tasks such as slicing are already performed.

The Cloud computing community has made important efforts to facilitate the use of data-

center resources. Cloud Operating Systems have been proposed, such as OpenStack [59], designed to manage and monitor large-scale deployments, providing access to network, compute and storage resources through a set of homogeneous APIs and sub-projects. Available tools are generally oriented towards applications being deployed in a global pool of resources. Container-specific tools such as Kubernetes [113] present some interesting aspects in that they expose a unique consistent API for simplicity, with however limited control granularity our purpose. Automation is ensured by third party tools layered on top of these standard APIs, like Chef [151], which are intrinsically limited by their procedural language design, where the user must make every step of the deployment explicit, manually adapt to the current state and handle errors. Other tools (e.g., Puppet [112]) use a descriptive language, where the user only needs to describe his/her needs and leave the rest to the tool. Despite the integration effort in Cloud computing, the silos around functionalities and the proliferation of APIs appear limiting for our purpose. The system does not enable simple setup and control for users, nor to build applications on top.

The now joint SDN and NFV communities are maybe the closest to our needs, at least from an architectural point of view. In [78] for instance, the authors describe a set of design principles for the Management and Orchestration (MANO) of virtualized network functions (VNF). Their approach is based on a modular architecture, clearly identifying the fundamental function such as user and VNF description, orchestration, etc. They also point the need to ensure reliable management by considering the lifecycle of the resource they manage. OpenDayLight is a promising candidate framework for building NFV capabilities, as it relies on a model-driven abstraction layer that fits with our requirements. However, this aspect is mainly used from a software engineering point of view, and not to offer programmability of the resource (called "micro-service"). Moreover, orchestration is layered on top of other modules that behave as silos.

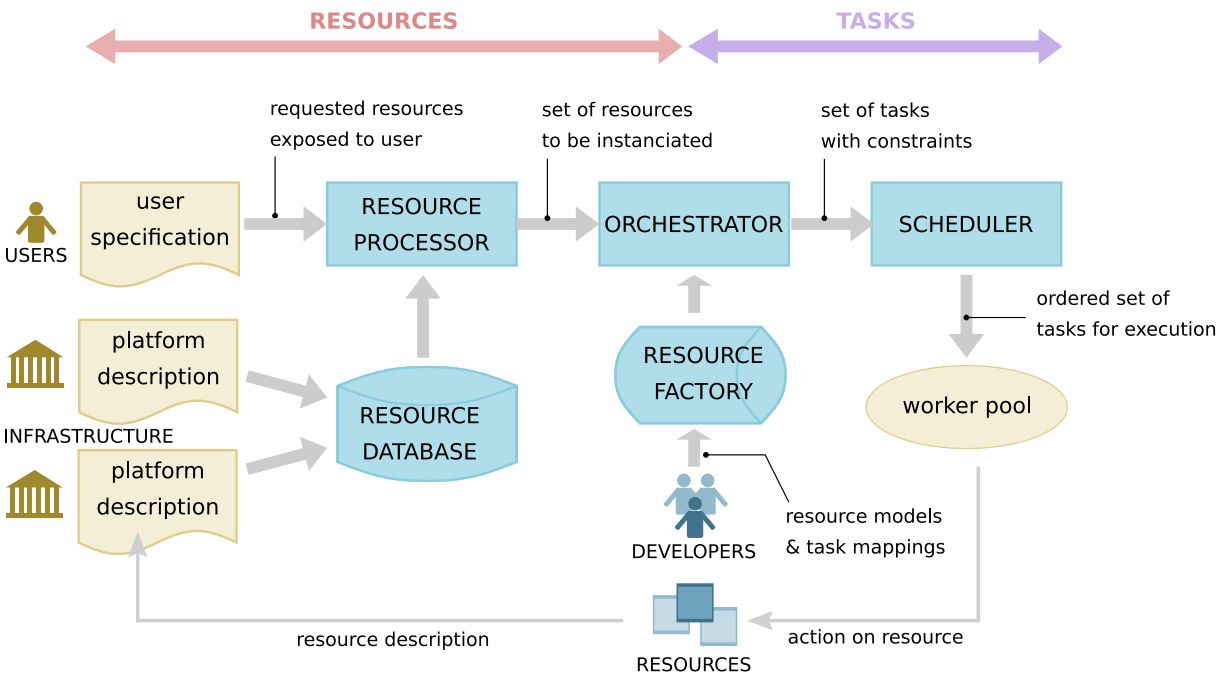


Figure 7.1: vICN functional architecture

7.3 The vICN framework

The high-level architecture of vICN, presented in fig. 7.1 reminds of NFV proposals such as MANO. Our contribution is in applying the underlying resource end-to-end and globally, and the properties this guarantees.

7.3.1 Functional architecture

The Resource is the basic unit of information in vICN. It consists of an abstract model that a set of mappers use to translate actions on the Resource into a series of task to be executed. Resources are stored in the **Resource factory**, and can be very diverse, (e.g., a specific Virtual Machine, an application or an IP route). They can be combined or extended to form other Resources. The vICN architecture differentiates the role of users, developers, and infrastructure providers. Developers integrate tools by creating new Resources or extending the old ones. Both users and infrastructure providers use this base set of Resources to describe what they respectively require (users) or make available (infrastructure providers).

Resources are specified according to different degrees of detail: e.g., the infrastructure is

described precisely to form a **Resource database** that serves as a base for deployment. On the other hand, user specifications might be general or abstract, and only mention Resources of interest for the user. The role of the **Resource processor** is to turn an abstract and incomplete description into a set of Resources mapped on the infrastructure leading to a consistent deployment. Once Resources are selected, the **orchestrator** translates them into a set of actions to be performed, based on the current state of deployment and on constraints due to task synchronization or sequentiality. The resulting actions are processed by a **scheduler**. It outputs an execution plan and dispatches parallel tasks to a worker pool with the objective of minimizing the deployment time.

To summarize, vICN is based on two main abstractions: *Resources*, which are external units of information exposed to users, developers and infrastructure providers, and *tasks*, which are internal units of information, defined by developers, to translate Resource requests into changes of network deployment state.

7.3.2 Resource model

In vICN, the internal Resource model is exposed directly to users and developers through a query language, in the spirit of SQL or SPARQL, which builds on and extend an *object relational model* [52]. The model defines a base *object* as a set of typed attributes and methods, where types refer to standard integers, strings, etc., or to newly defined object themselves. The query language built on top of it is used to create, destroy and manipulate those objects, either for Resource setup or to retrieve monitoring information. This model benefits from the power and expressiveness of the relational algebra [117] and of some key concepts of Object-Oriented Programming, namely composition and inheritance. It serves as an integrated interface based on both human- and machine-readable semantics (as in YANG [38]).

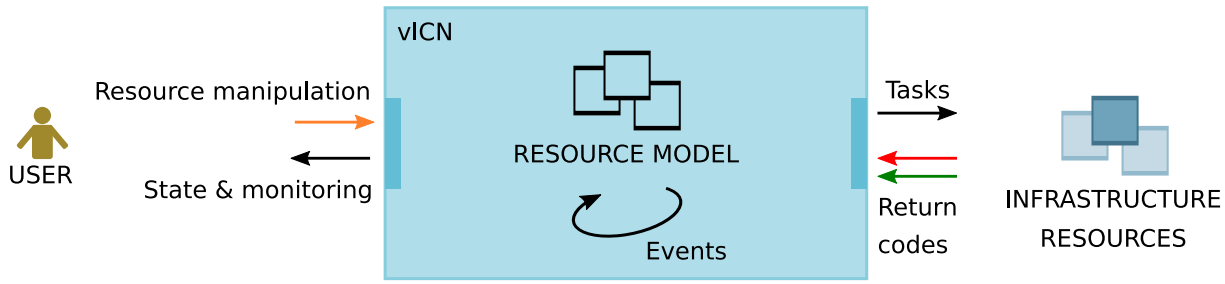


Figure 7.2: Flow of information in vICN

Resources

Resources in vICN are logical representations of physical and/or remote elements whose state has to be kept synchronized. The state of a Resource can be affected by user queries, by events involving Resources, or through monitoring queries issued to the remote Resource. For instance a routing component might recompute routes when notified about a change in the set of nodes, interfaces or links. The information flow is shown in fig. 7.2.

Resource state

The state of a Resource is tracked, for reliability and consistency, by a Finite State Machine (FSM) presented in the left part of fig. 7.3. The FSM models the possible states of a Resource (rectangles, raising events) and the pending operations, or tasks, being executed (round shapes). The transitions are dictated by user actions or internal events and follow the typical lifecycle of an object: *INITIALIZE* is called when the shadow Resource and its object are being created for internal setup; *CREATE* and *DELETE* are the respective constructor and destructor: they can create or destroy the remote Resource and eventually set some attributes; *GET* retrieves the current state of a Resource, as well as the state of some of its attributes; *UPDATE* proceeds to attribute update, and in fact runs parallel instances of attribute-FSM, as shown on the right-hand side of the figure.

Resource mapper

For each transition between states, a developer can associate commands to be executed thanks to Resource mappers. These commands are handled by vICN through the task

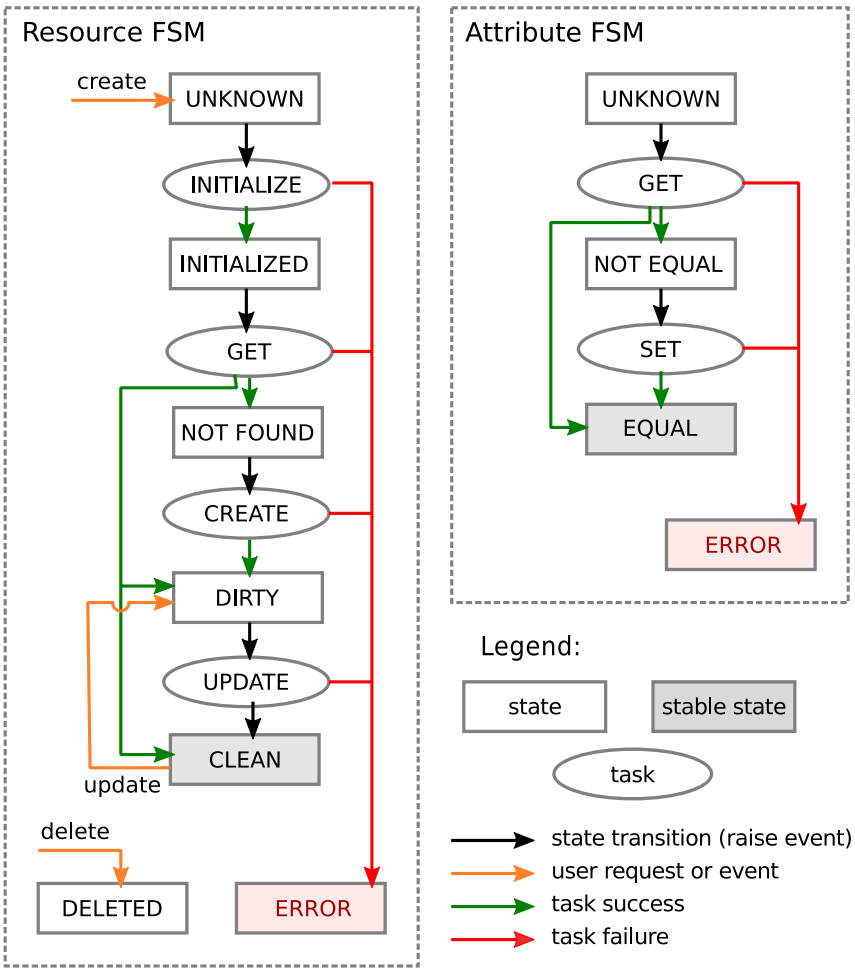


Figure 7.3: vICN Finite State Machine

abstraction, which also inherits from the base object. They are specialized to cope with multiple southbound interfaces such as NETCONF/YANG, SSH/Bash or LXD REST calls (similarly to Object-Relational Mappers such as SQLAlchemy [39]).

New tasks can be created through inheritance or composition, using algebraic operators to inform about their parallel or sequential execution. Resource objects are equipped with similar operators, so that inheritance and composition produce a similar composition of tasks. Both resources and tasks define an algebra, and the scheduler will be able to use this property to perform calculations and optimize the execution plan. A subset of Resources defined in vICN is represented in fig. 7.4, showing in particular the four base abstractions of Node, Interface, Channel and Application from which most Resources inherit, similarly to the model defined in [111].

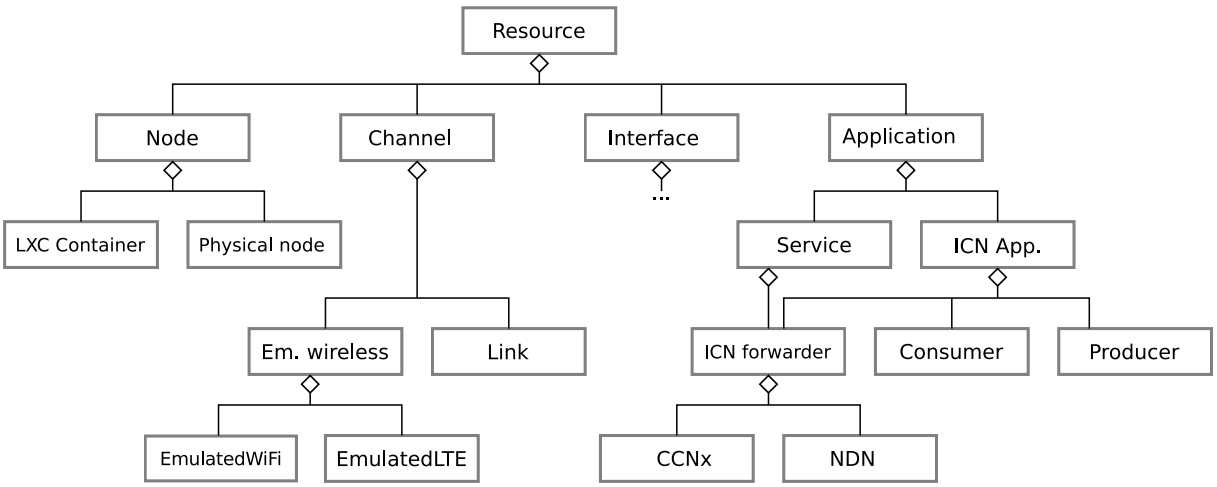


Figure 7.4: vICN partial Resource Hierarchy

7.3.3 Resource processor

The resource processor plays the central role of adapting the user requests to the platform policies and the available Resources. For instance, an abstract Resource `Node` can be implemented either as a `LXCContainer` or a `VM`. This choice (*specialization* step) can be either explicitly dictated by user preferences or *inferred* by the tool itself depending on the context. As another example, when deploying an ICN forwarder on a node that runs `hicn-ping`, vICN might prefer a hICN forwarder instance, and do the same for all nodes of the same experiment.

The Resource processor is also in charge of *mapping* the Resources to deploy onto available Physical servers, by verifying all the constraints/policies required by the user, the developer or by the infrastructure provider. Such assignment can be assimilated to an (NP-Hard) Constraint-Satisfaction Problem (CSP) [95] as the system has to accommodate several Resources in a finite capacity system in terms of networking, compute and memory. Both user-specified attributes and optional infrastructure provider policies are taken into account in the CSP as additional constraints. The output is a mapping from specification to implementation, which is also used to expose back monitoring to the user in a consistent way.

7.3.4 Orchestrator and Scheduler

The role of the orchestrator is to maintain one FSM per Resource, and ensure they reach the state requested by the user. Its outcome is a task dependency graph, which is shared with the scheduler. Task dependencies are derived from Resources dependencies, structure of the FSM, as well as from inheritance and composition constraints related to both the Resources and the mappers.

The scheduler ensures the scalability of the deployment by scheduling the parallel execution of tasks over a pool of worker threads. Given the dependency graph presented before, this corresponds to a classical DAG scheduling problem, which has been studied in the community [143]. Figure 7.5 presents a toy-scenario underlining the need for not naive scheduling algorithm. In that small example, a greedy selection of the task with higher distance to destination is a sufficient heuristic to get an optimal solution and save one execution round. We remark that user interactions can cause the graph of tasks to evolve in time and require a recomputation. Heuristics [134] might then be preferred to optimal solutions because of their faster execution time, while providing satisfactory performance.

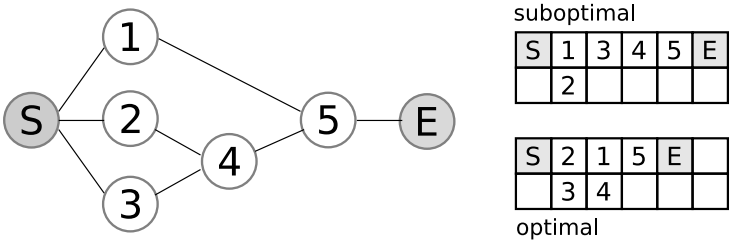


Figure 7.5: Toy scenario - vICN scheduler

Because of its centralized architecture, vICN performance is also impacted by the network transmission time[1]. We alleviate this issue by enabling task batching, when two consecutive tasks target the same node interface. The algebraic structure of tasks also makes it possible to reorganize the graph structure to better optimize execution, or to increase the ability to batch tasks.

[1]network round-trip-time and, for instance with TLS, session establishment time

7.4 Implementation

The flexibility of vICN lies in its modular architecture organized around its Resource model. Various Resources can be developed to cover a wide range of underlying infrastructure, and bring missing functionalities such as slicing or topology management. We here describe the current release and its set of base Resources covering the whole ICN stack. They build on and reuse available technologies, selected with scalability and reliability in mind.

7.4.1 vICN codebase

A first version of vICN has been open-sourced within the Community ICN (CICN) project [149], as part of the Linux Foundation's Fast Data I/O effort. The code, written in Python, is released under the Apache v2.0 license. This release implements all the building blocks described in fig. 7.1 and is mature enough to launch complex ICN deployments. Alongside, we distribute a prepackaged LXD image containing the full CICN/hICN suites (including forwarders, the ICN stack, and useful applications), so that a full ICN network can be bootstrapped in tens of seconds or minutes.

7.4.2 Slicing

In addition to bare-metal deployments, vICN is able to slice nodes and links offered by the infrastructure through a set of technologies that we describe here. This is crucial for proper experiment isolation, and to realize separate control and management planes.

Virtual nodes can be implemented either as containers or as virtual machines. We chose containers as the core technology (via the use of LXD) because they are more lightweight and efficient (thanks to zero-copy mechanisms, ZFS filesystem and simplified access to the physical resources). Increased security concerns and limitations such as sharing the same kernel are not limiting since most ICN functions are implemented in userland.

Network is shared at layer 2 via OpenVSwitch [150], which provides advanced functionalities, such as VLAN and OpenFlow rules, required by our wireless emulators and to bridge external real devices to the virtual environment. vICN fully isolates the deployment's network from the outside world by creating a single and isolated bridge per deployment, using iptables as a NAT to provide external connectivity. On top of that, we reduce the load of the bridge and isolate control traffic from the data plane. Indeed, we directly link connected containers, through pairs of Virtual Ethernet interfaces (veth), thus bypassing the bridge. Connected containers that are spawned across different servers in a cluster are transparently connected through a GRE tunnel.

Finally, vICN has to arbitrate for shared resources on the physical host, be it container or interface names (with constraints such as the 16-character limit on Linux), VLAN IDs, and even MAC or IP address depending on the level of required network isolation. It is important to do such "naming" properly not only for correctness, but also to simplify debugging and troubleshooting. vICN further enforces consistent names that uniquely identify a Resource, which allow for faster detection and recovery when the tool restarts or has to redeploy the same experiment.

7.4.3 IP and hICN topologies

Using the mechanisms described previously, it is possible to build a layer-2 graph on top of which vICN can set up IP and hICN connectivity. A centralized *IP Allocation* Resource is in charge of allocating IP prefixes and addresses to the different network segments of the graph, as well as hICN names to content producers. Global Connectivity is then ensured by computing the routes to be installed on the nodes. vICN provides a generic *routing module* implementing various algorithms (such as Dijkstra or Maximum-Flow) taking as an input a graph (layer-2) and a set of prefix origins (allocated IP addresses). It outputs a set of routes, encoded as vICN Resources. Route setup is then driven by a *Routing Table* Resource, from which the Linux and VPP routing tables inherit.

7.4.4 Link emulation

A feature that is missing from most tools is the ability to measure the performance of applications running on top of virtual networks with specific bandwidth or propagation delays. vICN offers Resource attributes for the Linux Traffic Control layer (tc) in order to shape link bandwidth and emulate constrained networks.

A complementary aspect is the ability to use emulated radio Resources as an alternative to real hardware in a transparent fashion. Two types of radio channel are currently supported, WiFi and LTE, both based on real-time simulation features of the NS-3 simulator. The vICN radio channel Resource is implemented as a drop-in replacement of a regular radio link Resource. It connects stations and access point (or UEs and Base Station) through a configurable radio channel, and hides the internal wiring from the user. The emulation then takes care of all relevant wireless features such as *beaconing, radio frequency interference, channel contention, rate adaptation and mobility*. Real-time emulation scales by using multiple instances orchestrated by an overarching mobility management Resource in vICN, communicating in real time with the different emulators. This process can collect relevant information from the simulation, and expose it to the internal model and thus monitoring.

7.4.5 Monitoring capabilities

Monitoring is natively implemented as part of vICN as a transversal functionality, building on the object model introduced in section 7.3. The query language offered by vICN allows to query any object attribute, including annotations made by the Resource processor and orchestrator about the host or the deployment state of the Resource. This is the same interface that is used by the orchestrator to query the current state of a remote Resource, to communicate with the emulators, or for the user to interact with vICN in order to change an attribute or create a new Resource at runtime. Its syntax closely matches SQL syntax. More precisely a query object contains the following elements: the object name, a query type (create, get..), a set of filters and attributes, eventually completed by attribute

values to be set.

For periodic measurements such as link utilization, vICN provides a daemon that can be installed on the nodes and exposes information via a similar interface. Communication between the components is ensured using the IP underlay setup by vICN.

7.5 Examples

We now illustrate some characteristics of vICN using a particular use case: mobile video delivery. This section is not meant to be exhaustive, but to illustrate how the design of vICN helped us solve practical challenges, and to emphasize general properties of the design that are relevant to other use-cases.

7.5.1 Use case description

The recent years have seen drastic changes in the video consumption patterns that put much pressure on delivery networks: the shifts in video quality (up to 4K), from broadcast to on-demand and from fixed to wireless and mobile networks. Our objective was to show that ICN addresses these challenges, using mechanisms like caching or multihoming over heterogeneous networks. Figure 7.6 represents an example of such a video delivery network. It consists of four parts: an heterogeneous WiFi/LTE access network with multihomed video clients; a backhaul network aggregating the resulting traffic with workload from emulated clients; a core network composed of two nodes; and producers serving 4K video. All nodes have a fully-featured ICN-stack. The core nodes use a VPP-based high-speed forwarder, the others a socket-based one. Overall, the deployment consists of 22 LXC containers, 3 real devices connected to the virtual network, 22 emulated links (including WiFi and LTE channels), and one physical link between DPDK-enabled network cards (the core). We use a pre-packaged container image containing all the necessary software to reduce the bootstrap time.

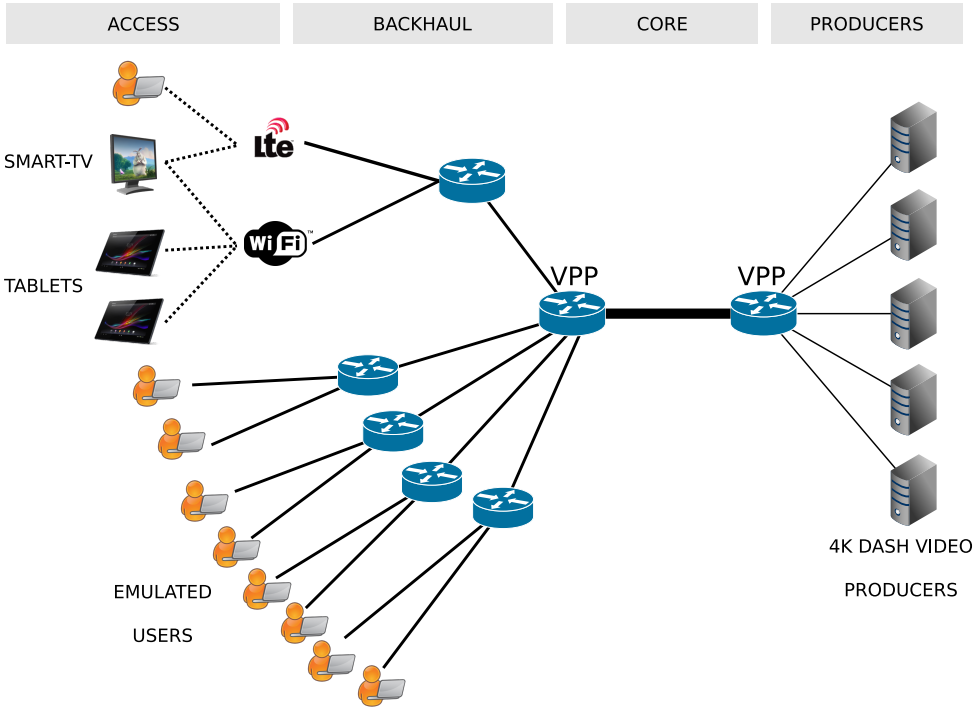


Figure 7.6: Mobile World Congress topology

7.5.2 Scalability

The simplification offered by vICN is illustrated the following numbers: during the deployment, vICN created about 800 Resources compared to the 104 declared in the topology file, a reduction in complexity of 85-90%. More than 1500 bash commands were executed, either directly on physical machines, or on LXC containers. This even underestimates the number of Bash commands an operator would type to deploy an equivalent topology, as some are batched for efficiency reasons (e.g., we insert all IP routes for a given node in a single command).

Figure 7.7 then shows the time taken by vICN to deploy the topology as a function of the number of dedicated threads. We deployed this topology on a Cisco UCS-C with 72 cores clocked at 2.1 Ghz. We first note that multi-threading provides a sevenfold reduction in bootstrap time, and that the topology can be deployed in about two minutes. The observed gains are due to the I/O-intensive nature of tasks, which spend most of their lifetime waiting for return values. This reduction is specific to our implementation and our simplistic scheduling heuristic. The shape of the curve remains nonetheless interesting, with a performance bound appearing. This is due to the underlying task graph, whose

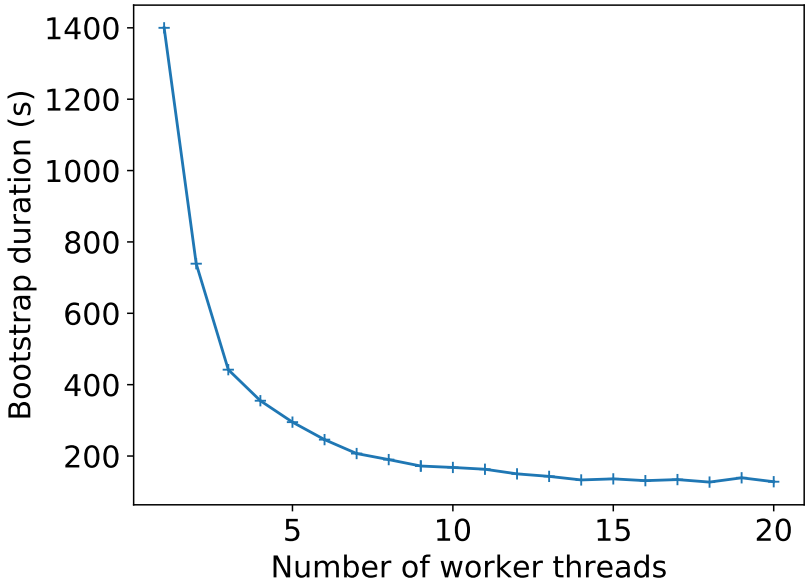


Figure 7.7: vICN bootstrap time vs number of worker threads

breadth intrinsically limits the number of tasks that can be run in parallel.

7.5.3 Programmability

One advantage of our Resource model (see section 7.3.2) is the use of inheritance. It allows the user to choose his level of granularity depending on his or her needs and expertise. In particular, the user can remain oblivious to the underlying technology used to deploy Resources. We used that feature to scale the demonstration on a cluster of servers connected through a switch instead of a single powerful server. In that configuration, linking containers on different hosts requires to connect them to virtual bridges on their respective hosts, and to link these bridges through a L2-tunnel. The two deployments, shown in fig. 7.8, require different Resources and tasks. However, they can be realized with the same vICN specifications, thanks to the Link abstract Resource. Here, vICN completely abstracts the implementation complexity and enables painless switching from one deployment to the other.

The deployment of containers running VPP is another example of vICN's ability to shield a user from implementation and configuration details thanks to its Resource model. Indeed, VPP uses DMA access to contiguous memory areas named hugepages. Both the host and

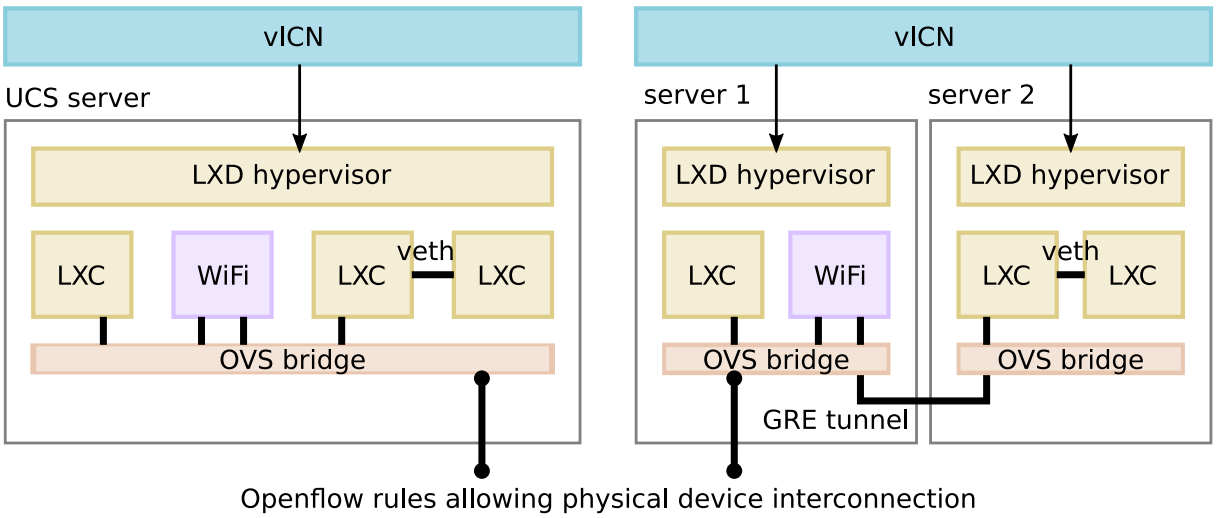


Figure 7.8: Alternative vICN topology deployments on single server and a cluster.

the containers have to be configured to allocate and share enough of these hugepages. On top of starting and setting up the application on the container, VPP thus requires to execute commands on the physical node and to change the container's configuration before its creation. In vICN, simply linking VPP to a container is enough to perform the bootstrap. The tool is then able to change the other Resources (e.g., use a VPP-enabled container instead of the standard one) and to run all the necessary commands.

The flexibility of the framework also allowed us to switch Resources in many occasions. During our tests, we replaced real tablets by emulated nodes to generate test workloads. During the demonstrations, we could also seamlessly use a real LTE mobile core instead of an emulated one. It only required to change one Resource in the specification and did not affect the rest of the scenario.

7.5.4 Monitoring and Reliability

We conclude by highlighting how the Resource model enables monitoring and debugging. As described in section 7.4.5, vICN exposes a query language based on its underlying model for monitoring. This language can be used to collect information about the network status at different time scales: link utilization, radio status, cache status etc. vICN thus integrates all information about the deployment in a consistent and query-able representation, building on the model introduced in section 7.3. In the same way vICN

provides an API to navigate through structured logs that may assist the whole process of software development.

7.6 Conclusion

In this last part of the thesis, we introduce *vICN (virtualized ICN)*, a flexible unified framework for ICN network configuration, management, and control to complement existing tools, especially for large scale and operational networks deployment. vICN is an object-oriented programming framework rooted in recent advances in SDN/NFV research that provides higher flexibility than existing virtualization solutions. It is specifically tailored to ICN, but its modular design allows for extensions to other technologies. While most of current software is developed in silos, with significant limitations in terms of optimization, vICN offers the capability to optimize each component of the virtual network to provide carrier-grade service guarantees in terms of programmability, scalability and reliability.

The vICN design, which we illustrate briefly through a concrete example, comes with a free software implementation available in the Linux Foundation for the community with multiple objectives: demonstrations, research and field trials. We leave for future work the detailed presentation of vICN characteristics by means of benchmarking in different use cases.

Chapter 8

Conclusions and Future Work

8.1 Summary

This thesis proposed solutions for incrementally deploying ICN into the current internet, considering novel designs and architectures allowing clean ICN insertion at the network, transport and application level. In addition, due to the shift to cloud computing and virtualization, and the need of being able to automate and program the deployment of networks and applications, the thesis proposed a flexible and unified framework for automated ICN network deployment, configuration, management and control.

Clean slate insertion of ICN at the network layer has been treated in chapter 3. The proposed solution, hICN, envisions the insertion of ICN inside the current Internet Protocol, by mapping application level names into IP addresses and reusing the TCP/IP headers for conveying ICN semantics inside IP. The architecture of hICN is a promising solution in the mid-term for operational networks in a variety of segments: residential, enterprise and data center. Results are promising, and they show that:

(i) hICN does not trade off any of the ICN principles presented in chapter 2.

(ii) hICN packets are already able to natively traverse the current IPv6 internet, with some exceptions related to company security appliances. This restriction is anyway limited to the first hop between the company network and the first hICN router, and can be easily solved with a tunneling protocol.

Another fundamental contribution of this work is the open source implementation of hICN [6], whose codebase can be already used for deploying hICN as part of the current internet.

Future work in this direction envisions an analysis on whether hICN can exploit IP control plane for intermediate nodes: although the IP control plane can potentially already be reused for the routing, it still requires a novel control-plane service to securely provision name prefixes at the producer, similarly to what DHCP provides to provision locators to network interfaces.

As second main topic, the thesis focused on the design, implementation and benchmarking of a transport layer for ICN, with a socket API for exposing ICN transport services. The architecture of the transport is based on a set of modular transport services designed to provide the ICN feature to applications without complexifying them. Applications access ICN transport services through a set of API based on the standard BSD Socket API.

The implementation of the transport framework is available at [6] and it is based on high speed technologies such as VPP and DPDK. An extensive evaluation of this transport at the end of chapter 4:

(i) confirms that such implementation is able to perform well for the segmentation and reassembly transport services, exploiting zero copy operations thanks to shared-memory between transport layer and ICN forwarder.

(ii) it underlines the importance of using a transport manifest, due to the cost of crypto operations such as signing, verifying and hashing.

(iii) it points out the impact of crypto operations on the overall performance and suggests the need of implementing such operations in hardware for keeping a high speed implementation able to guarantee the basic security features of ICN: authentication and integrity.

As a future work, the ICN transport design would still require additional integration to security features in the end-points such as TLS, DTLS and also the novel MLS [35] protocols to support as many applications as possible. Furthermore, the design of a

session layer mapping and multiplexing application namespaces into multiple network namespaces is a required work that complements the discussion initiated in this thesis.

Chapter 5 and chapter 6 presented two relevant examples of application protocols built on top of the ICN transport. The former presented a new transport service able to provide push semantics on top of the pull-based ICN transport. Although many kind of applications can take advantage of push semantics, this chapter gave a special focus on HTTP, being HTTP one of the most diffused protocols today, in particular for delivering media content over the internet. Its support appears fundamental for ICN deployment and adoption. Not supporting HTTP would force any application to be rewritten on purpose on top of the request-reply ICN transport. Chapter 5 proposes a method for mapping HTTP requests to ICN names, allowing applications built on top of HTTP to take advantage of the ICN features. Experiments at the end of chapter showed the benefits of placing HTTP on top of the ICN transport, in particular in case of linear video distribution realized through live Dynamic Adaptive Streaming over HTTP (DASH): the content distribution scales with the number of the channel streamed rather than with the number of active user, allowing servers to scale much more since the number of active users can be orders of magnitude bigger than the number of actual channel broadcasted.

Chapter 6 presented the design and the architecture of hICN-RTC, a new distribution protocol tailored for real time applications. The chapter in particular contributed in the following:

(i) It presented the hICN-RTC synchronization protocol, a new data retrieval transport service able to overcome the limitation of the pull-based transport of ICN in case of real time content distribution.

(ii) It presented a new distribution architecture exploiting the ICN request aggregation and multicast, where the media bridge connecting all the participant to a given real time session retrieves the real time media only from the active producers and redistribute it to the other participants.

The benefits of this novel architecture are shown at the end of chapter 6. Results showed that the new synchronization protocol tailored for RTC allows to get a per-packet delivery

delay within 1/2 RTT for the majority of the packets. Furthermore, hICN-RTC allowed the whole system to scale up to 1000 conference participant, thanks to the fact that the media bridge does not retrieve the media from all the clients but it rather receive just the media of the active ones. The rest of the traffic is served directly by the underlying ICN network.

At last, chapter 7 presented vICN, a novel flexible and unified framework for ICN network configuration, management and control, providing programmability for developers and scalability / reliability for users. vICN simplifies the deployment of ICN networks, by mapping declarative user specifications to corresponding resources and deploying them on top of the available infrastructure. It allows to strictly control all the level of the corresponding deployment, to combine network and applications needs, and to satisfy ICN requirement such as name provisioning or multipath routing. The examples at the end of chapter 7 showed how vICN can significantly automate the deployment of ICN topologies, by translating simple user requirements to complex network deployment-configuration operations (104 resources deployed through more than 1500 Bash commands). In addition, it provides managing and monitoring of the deployed network, by querying the state of the resources and reporting it to the user.

vICN has been extensively used for deploying and configuring all the ICN networks used during the demonstrations. It always provided a sufficient level of reliability and scalability, by allowing at the same time to easily integrate new resources thanks to the programmable and modular framework, thought for continuously integrating new components.

8.2 List of contributions

The work presented in this thesis would not have been possible without the help of advisors and fellow research workers. This section states the part played in every chapter by the author with respect to the contributions of the other members of the team.

With respect to the Hybrid ICN solution presented in chapter 3, the design of the hICN protocol was a joint work among all the authors, while the main contribution was mainly

in the generation/analysis of data through extensive experimentation. Experiments also led to reconsider some design choices we took initially, by redefining the semantic of the header fields or their disposition, in particular for what regards manifest, signature and fields in the L4 header.

The work presented in chapter 4 and chapter 5, which includes the design, implementation and evaluation of the transport framework, the socket API and its application with respect to HTTP, has been a complete contribution of the author of the thesis.

Chapter 6 is a partial contribution of the author. While no particular contribution has been done with respect the design of the synchronization protocol itself, a big contribution consisted mainly in the design of the system architecture, in setting up the experiments for the evaluation and in analysing the data.

Finally, chapter 7 has been a work started by the author of the thesis which then has been integrated with additional contributions of the other authors, in particular with improvements to the design of vICN.

Bibliography

[1] EU FP7, Pursuit project. http://www.fp7-pursuit.eu/PursuitWeb/.

[2] IPv6 CIDR REPOR. http://www.cidr-report.org/v6/as2.0/.

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

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

[5] libparistraceroute. https://github.com/libparistraceroute/.

[6] Linux Foundation FD.io Hybrid ICN project. https://wiki.fd.io/view/HICN.

[7] NDN project, Named-Data Networking Principles. https://named-data.net/project/ndn-design-principles/.

[8] NSF/Intel, Partnership on Information-Centric Networking in Wireless Edge Networks (ICN-WEN). https://www.nsf.gov/pubs/2016/nsf16586/nsf16586.htm.

[9] PeeringDB. https://www.peeringdb.com/.

[10] Route Views Project. http://www.routeviews.org/.

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

[12] Under the hood: Broadcasting live video to millions. http://goo.gl/gsFyqo.

[13] Apache Traffic Server, 2018.

[14] *nginx*. https://nginx.org/en/, 2018.

[15] Internet AS-level Topology Archive. http://irl.cs.ucla.edu/topology, 2018.

[16] Iperf. https://iperf.fr, 2018.

[17] Live encoder settings, bitrates, and resolutions. http://goo.gl/tDtc1i, 2018.

[18] *lxc*. https://linuxcontainers.org/, 2018.

[19] MGEN. https://www.nrl.navy.mil/itd/ncs/products/mgen, 2018.

[20] *nginx* RTMP module. https://nginx.org/en/, 2018.

[21] Open Broadcaster Software (OBS). https://obsproject.com/, 2018.

[22] *openSSL*. https://www.openssl.org/, 2018.

[23] The NEAT project, 2018.

[24] Team Cymru, IP to ASN mapping. http://www.team-cymru.org/Services/ip-to-asn.html, Accessed Dec. 2017.

[25] 13, I.-T. S. G. Data aware networking (information centric networking) – Requirements and capabilities . https://www.itu.int/rec/T-REC-Y.3071-201703-P, Dec 2016.

[26] AbdAllah, E. G., Hassanein, H. S., and Zulkernine, M. A Survey of Security Attacks in Information-Centric Networking. *IEEE Communications Surveys Tutorials 17*, 3 (thirdquarter 2015), 1441–1454.

[27] Adhatarao, S. S., Chen, J., Arumaithurai, M., Fu, X., and Ramakrishnan, K. K. Comparison of naming schema in ICN. In *'16* (June 2016).

[28] Atkinson, R., and Bhatti, S. Identifier-Locator Network Protocol (ILNP) Architectural Description. RFC 6740, Nov 2012.

[29] Augé, J., Carofiglio, G., Enguehard, M., Muscariello, L., and Sardara, M. Simple and Efficient ICN Network Virtualization with vICN. In *Proceedings of the 4th ACM Conference on Information-Centric Networking* (New York, NY, USA, 2017), ICN '17, ACM, pp. 216–217.

[30] Augé, J., Carofiglio, G., Grassi, G., Muscariello, L., Pau, G., and Zeng, X. MAP-Me: Managing Anchor-less Producer Mobility in Information-Centric Networks.

[31] Auge, J., Carofiglio, G., Muscariello, L., and Papalini, M. Anchorless mobility management through hICN (hICN-AMM): Deployment options. Internet-Draft draft-auge-dmm-hicn-mobility-deployment-options-01, Internet Engineering Task Force, Dec 2018. Work in Progress.

[32] Auge, J., Carofiglio, G., Muscariello, L., and Papalini, M. Anchorless mobility through hICN. Internet-Draft draft-auge-dmm-hicn-mobility-01, Internet Engineering Task Force, Dec 2018. Work in Progress.

[33] Barford, P., Bestavros, A., Byers, J., and Crovella, M. On the marginal utility of network topology measurements. In *'01* (2001).

[34] Bari, M. F., Chowdhury, S. R., Ahmed, R., Boutaba, R., and Mathieu, B. A survey of naming and routing in information-centric networks. *IEEE Communications Magazine 50*, 12 (December 2012), 44–53.

[35] Barnes, R., Millican, J., Omara, E., Cohn-Gordon, K., and Robert, R. The Messaging Layer Security (MLS) Protocol. Internet-Draft draft-ietf-mls-protocol-04, Internet Engineering Task Force, Mar 2019. Work in Progress.

[36] Baugher, M., Davie, B., Narayanan, A., and Oran, D. Self-verifying names for read-only named data. In *Proc. of IEEE INFOCOM 2012 Workshops* (Orlando, FL, USA, March 2012), pp. 274–279.

[37] Bhat, D., Wang, C., Rizk, A., and Zink, M. A load balancing approach for adaptive bitrate streaming in Information Centric networks. In *2015 IEEE International Conference on Multimedia Expo Workshops (ICMEW)* (June 2015), pp. 1–6.

[38] Bjorklund, M. YANG - A Data Modeling Language for the Network Configuration Protocol (NETCONF). RFC 6020, Oct 2010.

[39] Brown, A. *The architecture of open source applications (SQLAlchemy)*, vol. 2. Kristian Hermansen, 2012.

[40] Cabral, C., Rothenberg, C. E., and Magalhães, M. F. Mini-CCNx: Fast prototyping for named data networking. In *Proceedings of the 3rd ACM SIGCOMM workshop on Information-centric networking* (2013), ACM, pp. 33–34.

[41] Carofiglio, G., Gallo, M., and Muscariello, L. Joint Hop-by-hop and Receiver-driven Interest Control Protocol for Content-centric Networks. In *'12* (New York, NY, USA, 2012), pp. 37–42.

[42] Carofiglio, G., Gallo, M., and Muscariello, L. On the Performance of Bandwidth and Storage Sharing in Information-centric Networks. *Computer Networks 57*, 17 (Dec 2013), 3743–3758.

[43] Carofiglio, G., Gallo, M., and Muscariello, L. Optimal multipath congestion control and request forwarding in information-centric networks: Protocol design and experimentation. *Computer Networks 110* (2016), 104–117.

[44] Carofiglio, G., Gallo, M., Muscariello, L., and Perino, D. Pending Interest Table Sizing in Named Data Networking. In *Proceedings of the 2Nd ACM Conference on Information-Centric Networking* (2015), ACM-ICN '15, pp. 49–58.

[45] Carofiglio, G., Morabito, G., Muscariello, L., Solis, I., and Varvello, M. From Content Delivery Today to Information Centric Networking. *Comput. Netw. 57*, 16 (Nov 2013), 3116–3127.

[46] Carofiglio, G., Muscariello, L., Augé, J., Papalini, M., Sardara, M., and Compagno, A. Enabling ICN in the Internet Protocol: Analysis and Evaluation of the Hybrid-ICN Architecture. In *Proc. of ACM SIGCOMM ICN '19* (2019).

[47] Carofiglio, G., Muscariello, L., Papalini, M., Rozhnova, N., and Zeng, X. Leveraging ICN In-network Control for Loss Detection and Recovery in Wireless Mobile Networks. In *'16* (New York, NY, USA, 2016), pp. 50–59.

[48] Carzaniga, A., and Wolf, A. Forwarding in a Content-based Network. In *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications* (New York, NY, USA, 2003), SIGCOMM '03, ACM, pp. 163–174.

[49] Chai, W., He, D., Psaras, I., and Pavlou, G. Cache "Less for More" in Information-centric Networks. In *'12* (Berlin, Heidelberg, 2012), pp. 27–40.

[50] Chakraborti, A., Amin, S., Zhao, B., Azgin, A., Ravindran, R., and Wang, G. ICN Based Scalable Audio-Video Conferencing on Virtualized Service Edge Router (VSER) Platform. In *Proceedings of the ACM Conference on Information-Centric Networking* (2015), ICN '15, ACM.

[51] Compagno, A., Conti, M., Ghali, C., and Tsudik, G. To NACK or not to NACK? negative acknowledgments in information-centric networking. In *2015 24th International Conference on Computer Communication and Networks (ICCCN)* (2015), IEEE, pp. 1–10.

[52] Date, C. J., and Darwen, H. *Foundation for object/relational databases: the third manifesto.* Addison-Wesley Professional, 1998.

[53] Davie, B., and Gross, J. A Stateless Transport Tunneling Protocol for Network Virtualization (STT). Internet-Draft draft-davie-stt-08, Internet Engineering Task Force, Apr 2016.

[54] Detti, A., Salsano, S., and Blefari-Melazzi, N. IP protocol suite extensions to support CONET Information Centric Networking. Internet-Draft draft-detti-conet-ip-option-05, Internet Engineering Task Force, Jun 2013. Work in Progress.

[55] Farinacci, D., Fuller, V., Meyer, D., and Lewis, D. The Locator/ID Separation Protocol (LISP). RFC 6830, Jan 2013.

[56] Fayazbakhsh, S., Lin, Y., Tootoonchian, A., Ghodsi, A., Koponen, T., Maggs, B., Ng, K., Sekar, V., and Shenker, S. Less Pain, Most of the Gain:

Incrementally Deployable ICN. In *Proc. of the ACM SIGCOMM 2013* (New York, NY, USA, 2013), SIGCOMM '13, pp. 147–158.

[57] Feng, B., Zhou, H., and Xu, Q. Mobility support in Named Data Networking: a survey.

[58] for Standardization (ISO), I. O. ISO 26324:2012 Information and documentation – Digital object identifier system, 2012.

[59] Foundation, T. O. `https://www.openstack.org/`, 2017.

[60] Fundation, T. L. Linux Fundation Wiki - Netem. `https://wiki.linuxfoundation.org/networking/netem`.

[61] Gallo, M., Gu, L., Perino, D., and Varvello, M. NaNET: Socket API and Protocol Stack for Process-to-content Network Communication. In *Proc. of the 1st ACM SIGCOMM ICN Conference* (New York, NY, USA, 2014), '14, pp. 185–186.

[62] Garcia-Luna-Aceves, J. J., Martinez-Castillo, J. E., and Menchaca-Mendez, R. Routing to Multi-Instantiated Destinations: Principles, Practice and Applications. 1–1.

[63] Ghali, C., Narayanan, A., Oran, D., Tsudik, G., and Wood, C. A. Secure Fragmentation for Content-Centric Networks. In *2015 IEEE 14th International Symposium on Network Computing and Applications* (Sept 2015), pp. 47–56.

[64] Ghali, C., Tsudik, G., and Wood, C. (The Futility of) Data Privacy in Content-Centric Networking. In *Proceedings of the 2016 ACM on Workshop on Privacy in the Electronic Society* (2016), pp. 143–152.

[65] Ghali, C., Tsudik, G., and Wood, C. A. Network Names in Content-Centric Networking. In *Proc. of the 3rd ACM SIGCOMM ICN* (New York, NY, USA, 2016), '16, pp. 132–141.

[66] Ghodsi, A., Koponen, T., Rajahalme, J., Sarolahti, P., and Shenker, S. Naming in Content-oriented Architectures. In *'11* (New York, NY, USA, 2011), pp. 1–6.

[67] Gibbens, M., Gniady, C., Ye, L., and Zhang, B. Hadoop on Named Data Networking: Experience and Results. *Proc. ACM Meas. Anal. Comput. Syst. 1*, 1 (Jun 2017), 2:1–2:21.

[68] Gilligan, R., McCann, J., Bound, J., and Thomson, S. Basic Socket Interface Extensions for IPv6. Tech. Rep. 3493, Mar 2003.

[69] Git, G. WebRTC library. `https://webrtc.googlesource.com/`.

[70] Grandl, R., Su, K., and Westphal, C. On the interaction of adaptive video streaming with content-centric networking. In *Proc. of IEEE Int. Packet Video Workshop* (Dec. 2013).

[71] Gritter, M., and Cheriton, D. R. An Architecture for Content Routing Support in the Internet. In *Proc. of the 3rd USENIX Symposium on Internet Technologies and Systems* (Berkeley, CA, USA, 2001), USITS'01, pp. 4–4.

[72] Grozev, B., Marinov, L.and Singh, V., and Ivov, E. 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* (2015), NOSSDAV '15, ACM.

[73] Grozev, B., Politis, G., Ivov, E., Noel, T., and Singh, V. Experimental Evaluation of Simulcast for WebRTC. *IEEE Communications Standards Magazine* (2017).

[74] Gusev, P., and Burke, J. NDN-RTC: Real-Time Videoconferencing over Named Data Networking. In *Proceedings of the ACM Conference on Information-Centric Networking* (2015), ICN '15, ACM.

[75] Heath, L., Owen, H., Beyah, R., and State, R. CLIP: Content labeling in IPv6, a layer 3 protocol for information centric networking. In *Proc. of ICC 2013* (June 2013), pp. 3732–3737.

[76] Hesmans, B., Duchene, F., Paasch, C., Detal, G., and Bonaventure, O. Are TCP Extensions Middlebox-proof? In *Proc. of the 2013 HotMiddlebox Workshop* (New York, NY, USA, 2013), HotMiddlebox '13, pp. 37–42.

[77] Honda, M., Nishida, Y., Raiciu, C., Greenhalgh, A., Handley, M., and Tokuda, H. Is It Still Possible to Extend TCP? In *Proc. of ACM SIGCOMM IMC 2011* (New York, NY, USA, 2011), IMC '11, pp. 181–194.

[78] Institute, E. T. S. Network Functions Virtualisation (NFV); Management and Orchestration. Tech. Rep. GS NFV-MAN 001, European Telecommunications Standards Institute (ETSI), 2014.

[79] Ioannidis, S., and Yeh, E. Jointly optimal routing and caching for arbitrary network topologies. In *Proc. of the 4th ACM SIGCOMM ICN 2017* (2017), pp. 77–87.

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

[81] Jacobson, V., Smetters, D. K., Briggs, N. H., Plass, M. F., Stewart, P., Thornton, J. D., and Braynard, R. L. VoCCN: Voice-over Content-centric Networks. In *Proc. of the 2009 Workshop on Re-architecting the Internet* (2009), ReArch '09, ACM.

[82] Jacobson, V., Smetters, D. K., Thornton, J. D., Plass, M. F., Briggs, N. H., and Braynard, R. L. Networking Named Content. In *Proc. of the 5th ACM CoNEXT* (New York, NY, USA, 2009), CoNEXT '09, pp. 1–12.

[83] James, C., Halepovic, E., Wang, M., Jana, R., and Shankaranarayanan, N. K. 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) (Sept 2016), pp. 331–336.

[84] Jangam, A., Ravindran, R., Chakraborti, A., Wan, X., and Wang, G. Realtime multi-party video conferencing service over information centric network. In *2015 IEEE International Conference on Multimedia and Expo Workshops (ICMEW)* (June 2015).

[85] Jiang, X., Bi, J., and Wang, Y. What benefits does NDN have in supporting mobility. In *Proc. of IEEE Symposium on Computers and Communications (ISCC)* (June 2014), pp. 1–6.

[86] Koch, P. A DNS RR Type for Lists of Address Prefixes (APL RR). RFC 3123, Jun 2001.

[87] Koponen, T., Chawla, M., Chun, B., Ermolinskiy, A., Kim, K., Shenker, S., and Stoica, I. A Data-oriented (and Beyond) Network Architecture. *ACM SIGCOMM Comput. Commun. Rev. 37*, 4 (Aug 2007), 181–192.

[88] Lacage, M., Ferrari, M., Hansen, M., Turletti, T., and Dabbous, W. NEPI: using independent simulators, emulators, and testbeds for easy experimentation. *SIGOPS Operating Syst. Rev. 43*, 4 (2010), 60–65.

[89] Langley, A., et al. The QUIC Transport Protocol: Design and Internet-Scale Deployment. In *Proceedings of the Conference of the ACM Special Interest Group on Data Communication* (New York, NY, USA, 2017), SIGCOMM '17, ACM, pp. 183–196.

[90] Lederer, S., et al. Adaptive streaming over content centric networks in mobile networks using multiple links. In *Proc. of IEEE ICC* (2013).

[91] Lederer, S., Mueller, C., Rainer, B., Timmerer, C., and Hellwagner, H. An experimental analysis of Dynamic Adaptive Streaming over HTTP in Content Centric Networks. In *2013 IEEE International Conference on Multimedia and Expo (ICME)* (July 2013), pp. 1–6.

[92] Linguaglossa, Leonardo and Rossi, Dario and Pontarelli, Salvatore and Barach, Dave and Marjon, Damjan and Pfister, Pierre. High-speed Software Data Plane via Vectorized Packet Processing. `https://perso.telecom-paristech.fr/drossi/paper/vpp-bench-techrep.pdf`, 2018.

[93] Liu, X., Trappe, W., and Zhang, Y. Secure Name Resolution for Identifier-to-Locator Mappings in the Global Internet. In *2013 22nd International Conference on Computer Communication and Networks (ICCCN)* (July 2013), pp. 1–7.

[94] Luigi Rizzo, U. o. P. Netmap - the fast packet I/O framework. `http://info.iet.unipi.it/~luigi/netmap/`, 2018.

[95] Mackworth, A. K. Constraint satisfaction problems. *Encyclopedia of AI 285* (1992), 293.

[96] Mahdian, M., Arianfar, S., Gibson, J., and Oran, D. MIRCC: Multipath-aware ICN Rate-based Congestion Control. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking* (New York, NY, USA, 2016), ACM-ICN '16, ACM, pp. 1–10.

[97] Majeed, M., Ahmed, S., Muhammad, S., Song, H., and Rawat, D. Multimedia streaming in information-centric networking: A survey and future perspectives. *Computer Networks 125* (2017), 103 – 121.

[98] Marchal, X., Aoun, M. E., Mathieu, B., Cholez, T., Doyen, G., Mallouli, W., and Festor, O. Leveraging NFV for the deployment of NDN: Application to HTTP traffic transport. In *NOMS 2018 - 2018 IEEE/IFIP Network Operations and Management Symposium* (April 2018), pp. 1–5.

[99] Mastorakis, S., Gusev, P., Afanasyev, A., and Zhang, L. Real-Time Data Retrieval in Named Data Networking. In *1st IEEE International Conference on Hot Information-Centric Networking (HotICN 2018)* (2018).

[100] Mealling, M. H. A URN Namespace of Object Identifiers. RFC 3061, Feb 2001.

[101] Mealling, M. H. The Network Solutions Personal Internet Name (PIN): A URN Namespace for People and Organizations. RFC 3043, Jan 2001.

[102] Mininet. http://mininet.org/, 2017.

[103] Misra, S., Tourani, R., and Majd, N. Secure Content Delivery in Information-centric Networks: Design, Implementation, and Analyses. In *'13* (New York, NY, USA, 2013), pp. 73–78.

[104] Moiseenko, I., Stapp, M., and Oran, D. Communication Patterns for Web Interaction in Named Data Networking. In *Proceedings of the 1st ACM Conference on Information-Centric Networking* (New York, NY, USA, 2014), ACM-ICN '14, ACM, pp. 87–96.

[105] Moiseenko, I., Wang, L., and Zhang, L. Consumer / Producer Communication with Application Level Framing in Named Data Networking. In *Proc. of the 2nd ACM SIGCOMM ICN Conference* (New York, NY, USA, 2015), '15, pp. 99–108.

[106] Mosko, M., Solis, I., and Wood, C. CCNx Semantics. Internet-Draft draft-irtf-icnrg-ccnxsemantics-06, Oct 2017.

[107] Mosko, M., Solis, I., and Wood, C. A. CCNx Messages in TLV Format. Internet-Draft draft-irtf-icnrg-ccnxmessages-09, Internet Engineering Task Force, Jan 2019. Work in Progress.

[108] Mosko, M., Solis, I., and Wood, C. A. CCNx Semantics. Internet-Draft draft-irtf-icnrg-ccnxsemantics-10, Internet Engineering Task Force, Jan 2019. Work in Progress.

[109] Mosko, M., and Wood, C. A. Secure Fragmentation for Content Centric Networking. In *2015 IEEE 12th International Conference on Mobile Ad Hoc and Sensor Systems* (Oct 2015), pp. 506–512.

[110] Ngai, E., Ohlman, B., Tsudik, G., Uzun, E., Wahlisch, M., and Wood, C. A. Can We Make a Cake and Eat It Too? A Discussion of ICN Security and Privacy. *SIGCOMM Comput. Commun. Rev. 47*, 1 (Jan 2017), 49–54.

[111] NS3. The Network Simulator 3. `https://www.nsnam.org/`, 2017.

[112] OpenStack, P. `https://wiki.openstack.org/wiki/Puppet`, 2017.

[113] Orchestration, P.-G. C. `https://kubernetes.io/`, 2017.

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

[115] Pandit, R. Reliable Datagram Sockets (RDS). In *OpenIB Developers Workshop, Feb* (2006).

[116] Paper, G. A. W. Understanding Information-Centric Networking and Mobile Edge Computing. `http://www.5gamericas.org/files/1214/8175/3330/Understanding_Information_Centric_Networking_and_Mobile_Edge_Computing.pdf`, Dec 2016.

[117] Paredaens, J. On the expressive power of the relational algebra. *Information Processing Letters 7*, 2 (1978), 107 – 111.

[118] Pauly, T., Trammell, B., Brunstrom, A., Fairhurst, G., Perkins, C., Tiesel, P. S., and Wood, C. A. An Architecture for Transport Services. Internet-Draft draft-pauly-taps-arch-00, Internet Engineering Task Force, Feb 2018. Work in Progress.

[119] Perino, D., Varvello, M., Linguaglossa, L., Laufer, R., and Boislaigue, R. Caesar: A content router for high-speed forwarding on content names. In *2014 ACM/IEEE ANCS Symposium* (Oct 2014), pp. 137–147.

[120] Petrangeli, S., Bouten, N., Claeys, M., and Turck, F. D. Towards SVC-based Adaptive Streaming in information centric networks. In *2015 IEEE International Conference on Multimedia Expo Workshops (ICMEW)* (June 2015), pp. 1–6.

[121] Popa, L., Ghodsi, A., and Stoica, I. HTTP As the Narrow Waist of the Future Internet. In *Proc. of the 9th ACM SIGCOMM Hotnets Workshop* (New York, NY, USA, 2010), Hotnets-IX, pp. 6:1–6:6.

[122] Posch, D., Kreuzberger, C., Rainer, B., and Hellwagner, H. 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* (New York, NY, USA, 2014), VideoNext '14, ACM, pp. 25–30.

[123] Rahman, A., Trossen, D., Kutscher, D., and Ravindran, R. Deployment Considerations for Information-Centric Networking (ICN). Internet-Draft draft-rahman-icnrg-deployment-guidelines-05, Internet Engineering Task Force, Jan 2018. Work in Progress.

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

[125] Ravindran, R., Chakraborti, A., Amin, S., Azgin, A., and Wang, G. 5G-ICN: Delivering ICN Services over 5G Using Network Slicing. *IEEE Communications Magazine 55*, 5 (May 2017), 101–107.

[126] Ravindran, R., Chakraborti, A., Amin, S. O., Azgin, A., and Wang, G. 5G-ICN : Delivering ICN Services over 5G using Network Slicing, 2016.

[127] Ravindran, R., Lo, S., Zhang, X., and Wang, G. Supporting seamless mobility in named data networking. In *Proc. of IEEE ICC 2012* (June 2012), pp. 5854–5869.

[128] Ravindran, R., Suthar, P., and Wang, G. Enabling ICN in 3GPP's 5G NextGen Core Architecture. Internet-Draft draft-ravi-icnrg-5gc-icn-00, Internet Engineering Task Force, Oct 2017.

[129] Ren, Y., Li, J., Shi, S., Li, L., Wang, G., and Zhang, B. Congestion Control in Named Data Networking - A Survey. *Comput. Commun. 86*, C (Jul 2016), 1–11.

[130] Research, F. I., and Experimentation. `https://www.ict-fire.eu`, 2017.

[131] Rossini, G., and Rossi, D. A dive into the caching performance of Content Centric Networking. In *IEEE 17th Workshop on Computer Aided Modeling and Design of Communication Links and Networks (CAMAD)* (Sept 2012), pp. 105–109.

[132] Rossini, G., and Rossi, D. Evaluating CCN Multi-path Interest Forwarding Strategies. *Comput. Commun. 36*, 7 (Apr 2013), 771–778.

[133] Saino, L., Cocora, C., and Pavlou, G. CCTCP: A scalable receiver-driven congestion control protocol for content centric networking. In *Proc. of IEEE ICC 2013* (June 2013), pp. 3775–3780.

[134] Sakellariou, R., and Zhao, H. A hybrid heuristic for DAG scheduling on heterogeneous systems. In *Parallel and Distributed Processing Symposium, 2004. Proceedings. 18th International* (2004), IEEE, IEEE, p. 111.

[135] Salsano, S., Blefari-Melazzi, N., Detti, A., Morabito, G., and Veltri, L. Information Centric Networking over SDN and OpenFlow: Architectural Aspects and Experiments on the OFELIA Testbed. *Comput. Netw. 57*, 16 (Nov 2013), 3207–3221.

[136] Samain, J., Augé, J., Carofiglio, G., Muscariello, L., Papalini, M., and Sardara, M. Enhancing Mobile Video Delivery over an Heterogeneous Network Access with Information-Centric Networking. In *Proceedings of the SIGCOMM Posters and Demos* (New York, NY, USA, 2017), SIGCOMM Posters and Demos '17, ACM, pp. 22–24.

[137] Samain, J., Carofiglio, G., Muscariello, L., Papalini, M., Sardara, M., Tortelli, M., and Rossi, D. Dynamic Adaptive Video Streaming: Towards a Systematic Comparison of ICN and TCP/IP. *IEEE Transactions on Multimedia 19*, 10 (Oct 2017), 2166–2181.

[138] Sardara, M., Muscariello, L., Augé, J., Enguehard, M., Compagno, A., and Carofiglio, G. Virtualized ICN (vICN): Towards a Unified Network Virtualization Framework for ICN Experimentation. In *Proc. of the 4th ACM SIGCOMM ICN* (New York, NY, USA, 2017), ICN '17, ACM, pp. 109–115.

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

[140] Sardara, M., Muscariello, L., and Compagno, A. Efficient Transport Layer and Socket API for ICN. In *Proceedings of the 5th ACM Conference on Information-Centric Networking* (New York, NY, USA, 2018), ICN '18, ACM, pp. 206–207.

[141] Sardara, M., Samain, J., Augé, J., and Carofiglio, G. Application-specific policy-driven 5G Transport with Hybrid ICN. *2019 IEEE International Symposium on Local and Metropolitan Area Networks (LANMAN)* (2019).

[142] Schneider, K., Yi, C., Zhang, B., and Zhang, L. A Practical Congestion Control Scheme for Named Data Networking. In *Proceedings of the 3rd ACM Conference on Information-Centric Networking* (New York, NY, USA, 2016), ACM-ICN '16, ACM, pp. 21–30.

[143] Sinnen, O. *Task scheduling for parallel systems*, vol. 60. John Wiley & Sons, 2007.

[144] Smetters, D., and Jacobson, V. Securing network content. Tech. rep., 2009.

[145] So, W., Narayanan, A., and Oran, D. Named data networking on a router: Fast and DoS-resistant forwarding with hash tables. In *2013 ACM/IEEE ANCS Symposium* (Oct 2013), pp. 215–225.

[146] Srinivasan, S., Rimac, I., Hilt, V., Steiner, M., and Schulzrinne, H. Unveiling the Content-Centric Features of TCP. In *Proc. of IEEE ICC 2011* (June 2011), pp. 1–5.

[147] Srinivasan, S., and Schulzrinne, H. IPv6 Addresses as Content Names in Information-Centric Networking. In *USENIX ATC 2011 - Poster session* (June 2011).

[148] Suthar, P., Stolic, M., Jangam, A., and Trossen, D. Native Deployment of ICN in LTE, 4G Mobile Networks. Internet-Draft draft-suthar-icnrg-icn-lte-4g-04, Internet Engineering Task Force, Nov 2017. Work in Progress.

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

[150] The Linux Foundation. Open vSwitch. `http://openvswitch.org/`, 2017.

[151] The Linux Foundation. OpenStack Chef. `https://wiki.openstack.org/wiki/Chef`, 2017.

[152] The Linux Foundation. Vector Packet Processing - Fast Data I/O. `https://wiki.fd.io/view/VPP/`, 2017.

[153] The Linux Foundation Projects. Data Plane Development Kit. `https://dpdk.org`, 2018.

[154] Trossen, D., Reed, M. J., Riihijärvi, J., Georgiades, M., Fotiou, N., and Xylomenos, G. IP over ICN - The better IP? In *2015 European Conference on Networks and Communications (EuCNC)* (Jun 2015), pp. 413–417.

[155] Tyson, G., Sastry, N., Rimac, I., Cuevas, R., and Mauthe, A. A Survey of Mobility in Information-centric Networks: Challenges and Research Directions. In *Proc. of the 1st ACM NoM Workshop 2012* (New York, NY, USA, 2012), NoM '12, pp. 1–6.

[156] Vahlenkamp, M., Schneider, F., Kutscher, D., and Seedorf, J. Enabling ICN in IP networks using SDN. In *ICNP* (2013), pp. 1–2.

[157] van Adrichem, N. L. M., and Kuipers, F. NDNFlow: Software-defined Named Data Networking. In *NetSoft* (2015), pp. 1–5.

[158] Wang, S., Bi, J., Wu, J., Yang, X., and Fan, L. On Adapting HTTP Protocol to Content Centric Networking. In *Proc. of the 7th International Conference on Future Internet Technologies* (New York, NY, USA, 2012), CFI '12, pp. 1–6.

[159] Wang, Y., Rozhnova, N., Narayanan, A., Oran, D., and Rhee, I. An Improved Hop-by-hop Interest Shaper for Congestion Control in Named Data Networking. In *'13* (New York, NY, USA, 2013), pp. 55–60.

[160] Westphal, C., et al. Adaptive Video Streaming over Information-Centric Networking (ICN). RFC 7933, Aug 2016.

[161] White, G., and Rutz, G. Content delivery in Content-Centric Networks. `http://www.cablelabs.com/wp-content/uploads/2016/02/Content-Delivery-with-Content-Centric-Networking-Feb-2016.pdf`, 2016.

[162] Yi, C., Afanasyev, A., Moiseenko, I., Wang, L., Zhang, B., and Zhang, L. A Case for Stateful Forwarding Plane. *Comput. Commun. 36*, 7 (Apr 2013), 779–791.

[163] Yi, C., Afanasyev, A., Wang, L., Zhang, B., and Zhang, L. Adaptive Forwarding in Named Data Networking. *SIGCOMM Comput. Commun. Rev. 42*, 3 (Jun 2012), 62–67.

[164] Yu, Y., Afanasyev, A., Clark, D., claffy, k., Jacobson, V., and Zhang, L. Schematizing Trust in Named Data Networking. In *Proc. of the 2Nd ACM SIGCOMM ICN* (New York, NY, USA, 2015), '15, pp. 177–186.

[165] Yuan, H., Crowley, P., and Song, T. Enhancing Scalable Name-Based Forwarding. In *ANCS* (2017), pp. 60–69.

[166] Zhang, F., Zhang, Y., Reznik, A., Liu, H., Qian, C., and Xu, C. A transport protocol for content-centric networking with explicit congestion control. In *Proc. of 23rd ICCCN 2014* (Aug 2014), pp. 1–8.

[167] Zhang, G., Li, Y., and Lin, T. Caching in information centric networking: A survey. *Computer Networks 57*, 16 (2013), 3128 – 3141.

[168] Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., claffy, k., Crowley, P., Papadopoulos, C., Wang, L., and Zhang, B. Named Data Networking. *ACM SIGCOMM Comput. Commun. Rev. 44*, 3 (Jul 2014), 66–73.

[169] Zhang, L., Afanasyev, A., Burke, J., Jacobson, V., claffy, k., Crowley, P., Papadopoulos, C., Wang, L., and Zhang, B. Named Data Networking. *SIGCOMM Comput. Commun. Rev. 44*, 3 (Jul 2014), 66–73.

[170] Zhang, L., Amin, S., and Westphal, C. Demo: VR Video Conferencing over Named Data Networks. In *Proceedings of the 4th ACM Conference on Information-Centric Networking* (2017), ICN '17, ACM.

[171] Zhang, M., Luo, H., and Zhang, H. A Survey of Caching Mechanisms in Information-Centric Networking. *IEEE Communications Surveys Tutorials 17*, 3 (2015), 1473–1499.

[172] Zhang, Y., Afanasyev, A., Burke, J., and Zhang, L. A survey of mobility support in Named Data Networking. In *IEEE INFOCOM WKSHPS 2016* (April 2016), pp. 83–88.

[173] Zhu, Z., Wang, S., Yang, X., Jacobson, V., and Zhang, L. ACT: Audio Conference Tool over Named Data Networking. In *Proceedings of the ACM SIGCOMM Workshop on Information-centric Networking* (2011), ICN '11, ACM.

[174] Zuraniewski, P., van Adrichem, N., Ravesteijn, D., IJntema, W., Papadopoulos, C., and Fan, C. Facilitating ICN Deployment with an Extended Openflow Protocol. In *Proc. of the 4th ACM SIGCOMM ICN* (New York, NY, USA, 2017), ICN '17, pp. 123–133.

pdfpages

Appendix A

Résumé en Français

Information Centric Networking

Information Centric Networking (ICN) désigne une architecture de réseau basée sur le nommage de données, plutôt que sur l'emplacement de leur hôte, pour une communication utilisateur-contenu. L'idée de la mise en réseau de données nommées n'est pas nouvelle. Elle a inspiré de nombreuses propositions, telles que TRIAD [71], CBN [48], DONA [87], avant de recevoir une attention significative au sein de la communauté de chercheurs avec les architectures CCN [82] et NDN [168] et avec des projets basés sur Pub-Sub (par ex. PSIRP/PURSUIT [1]).

Malgré d'importantes différences architecturales, une idée commune caractérise ICN: des schémas de communication évolutifs et indépendants de l'emplacement de leurs hôtes, avec une sécurité ancrée sur les données elles-mêmes. Il en résulte un support natif pour mobilité, le stockage et la sécurité en tant que fonctionnalités de réseau, qui sont intégrées dans l'architecture par conception, plutôt que comme une réflexion après coup. Plusieurs années de recherches et d'expérimentation en laboratoire ont fait progresser la conception architecturale et contribué à montrer le potentiel d'ICN. Cependant, l'idée divise encore la communauté en raison du compromis «coût/bénéfice» [56] lié à son introduction dans les réseaux IP existants.

Récemment, un regain d'intérêt industriel et universitaire pour ICN a été généré par la

nécessité de conceptions de réseaux capables de faire face aux futurs défis du réseau 5G. La prochaine génération de réseaux radiomobiles a en effet l'ambition de desservir un large nombre de cas d'utilisation sur plusieurs marchés verticaux et ICN a été identifié comme un candidat prometteur pour satisfaire leurs besoins réseaux, en matière de performances, d'évolutivité et de coût [25, 116, 8]. Les discussions architecturales de la 5G ont également relancé le débat sur les étapes de déploiement et le coût de l'introduction d'ICN dans les réseaux opérationnels.

Même si la virtualisation et la segmentation du réseau centré sur les applications en 5G permettent l'utilisation de nouveaux plans de données comme ICN [128, 125], le scepticisme persiste quant à son insertion à court terme.

Résumé de la thèse

Le but de cette thèse est de proposer une solution efficace pour insérer ICN dans l'Internet actuel, en considérant trois aspects cruciaux:

- **Le déploiement d'ICN dans l'Internet actuel**
- **La fourniture de services de transport ICN aux noeuds finaux**
- **L'amélioration et l'automatisation de la configuration, de la gestion et du contrôle d'un réseau ICN**

Premièrement, cette thèse propose une solution efficace pour **déployer incrémentalement ICN dans l'Internet actuel**. ICN est généralement déployé en exploitant une approche de superposition sur IP, en raison de la difficulté de changer la couche réseau (Internet Ossification) et l'effort requis pour normaliser une nouvelle couche protocolaire de niveau 3. De plus, tout le matériel construit au cours des trois dernières décennies a été optimisé pour le transfert IP et la conception et le déploiement de nouveaux appareils optimisés pour ICN est coûteux pour les opérateurs. Le chapitre 3 propose Hybrid ICN (hICN), une solution élégant au problème du déploiement de l'ICN. hICN est une

nouvelle architecture ICN construite sur l'utilisation des adresses IP comme noms ICN et la mise en correspondance la sémantique ICN à celle des en-têtes IP et TCP. Cela permet aux paquets ICN de traverser l'Internet de manière transparente, car ils apparaissent exactement comme un paquet IP normal. En même temps, ces paquets peuvent être traités par quelques routeurs hICN dans le réseau, qui peuvent être stratégiquement placés à la périphérie comme extension de routeurs IP standard, par exemple sous forme de fonction réseau virtuelle. Avec une vaste campagne expérimentale, le chapitre montre que les paquets hICN peuvent traverser l'Internet actuel, sans que l'architecture hICN ne fasse de compromis sur les principes de l'ICN, qui sont présentés dans chapitre 2.

Deuxièmement, la thèse se concentre sur le problème de **déployer ICN dans les points de terminaison réseau**, autrement dit sur l'hôte final. Cela se fait généralement par bibliothèques logicielles fournissant une API aux applications. Cependant, la plupart du temps ces bibliothèques permettent uniquement de développer des applications au-dessus de la couche réseau ICN: bien que cette approche donne un contrôle au niveau des paquets de la communication, la plupart du temps, il en résulte un défi trop complexe pour les développeurs d'applications, forcé de faire face à des problèmes typiques de couche 4 tels que le contrôle de la congestion et opérations de segmentation/remontage. Ce qu'un développeur d'applications aimerait est d'échanger de façon simple les données d'application (ADU), sans traiter les problèmes de réseau tels que les retransmissions, la gigue, le MTU découverte, etc. Ce qu'il faut, c'est une couche de transport offrant un environnement propre et une API simple qui permet aux développeurs d'accéder aux services de transport, comme ils les accèdent avec l'API Socket sur tous les systèmes d'exploitation modernes. Un motif du succès de TCP/IP s'explique par le fait que les applications peuvent l'utiliser via API simples permettant aux développeurs de communiquer via le réseau comme s'ils écrivaient simplement dans un fichier. Le chapitre 4 présente les fonctionnalités de la couche de transport et un ensemble d'interfaces (API) permettant aux développeurs d'applications de facilement accéder aux services de transport ICN. Plus particulièrement, ce chapitre décrit son architecture et sa mise en oeuvre, en mettant l'accent sur des technologies récentes telles que VPP [152], permettant de

fournir performance et efficacité aux applications. Une analyse comparative approfondie à la fin du chapitre présente une étude de performances, et dans quelle mesure les services de transport spécifiques d'ICN peuvent affecter les applications.

Par la suite, les chapitres 5 et 6 évaluent les avantages que les services de réseau et de transport hICN peuvent apporter aux applications. Ils considèrent notamment deux cas d'utilisation représentatif de la diversité des applications réseaux: HTTP et WebRTC. Bien que la sémantique de HTTP est Demande/Réponse comme ICN, les messages HTTP simples sont envoyés depuis le client vers serveur et vice-versa, car HTTP suppose que le transport sous-jacent se comporte comme TCP: les messages sont envoyés de l'expéditeur au destinataire. Pour cette raison HTTP ne peut pas être déployé directement sur ICN. Un service de transport supplémentaire est ainsi requis, placé comme middleware entre l'application elle-même et la couche de transport: cela permet un equivalent entre la sémantique HTTP et celle des noms ICN, ajoutant les avantages de ce dernier au modèle de communication client-serveur unicast de HTTP. Le chapitre 5 évalue ensuite les propriétés d'évolutivité de HTTP sur ICN, en particulier en ce qui concerne le côté serveur, qui doit notamment servir un grand nombre de clients. Il compare ensuite les différents comportements du transport ICN par rapport au modèle TCP / IP standard. Les résultats montrent que HTTP sur ICN évolue mieux que les solutions HTTP sur TCP actuelles, notamment dans le cas de la diffusion de vidéo linéaire.

La solution présentée pour HTTP n'est pas valable pour le cas d'une application avec un budget de latence extrêmement faible, comme les jeux vidéo ou la vidéoconférence. Le protocole WebRTC et son architecture sont de plus en plus adoptés par les entreprises de médias pour les services temps réel. Les exigences de WebRTC en matière de latence et d'évolutivité ne sont pas possible en utilisant la solution adoptée pour HTTP. Pour répondre à ces exigences, le chapitre 6 propose un nouveau protocole de synchronisation adaptée aux applications en temps réel, ainsi qu'une nouvelle conception d'architecture capable de suivre la croissance du volume de contenus (par exemple, le nombre de flux multimédias actifs dans une vidéoconférence) plutôt que le nombre de participants à la session en temps réel. Le chapitre 6 présentera la conception de hICN-RTC, une version

de WebRTC intégrée sur hICN, ainsi que son protocole de synchronisation, permettant d'utiliser le transport hICN sans introduire de latence supplémentaire dans la distribution de contenu. Les résultats d'expérimentation au sein de ce chapitre confirment les choix de conception effectués et valident les propriétés d'évolutivité de la solution conçue.

Enfin, le chapitre 7 propose une solution pour **déployer, configurer et gérer des réseaux et applications ICN par programmation**: virtual ICN (*vICN*). Dans ICN, la liaison entre l'application et la couche réseau limite le choix d'outils d'automatisation de déploiement de réseaux, qui doivent donc être repensés en conséquence. La plupart du temps, ces outils ne fournissent pas ce type de communication "intercouche" (par exemple, la fourniture d'adresses IP pour les noms hICN). vICN est un travail préliminaire qui tente de combler cette lacune, en particulier dans le cas de hICN, où une collaboration étroite entre l'application et la couche réseau est requise, étant donné que la sémantique ICN est directement intégrée dans la couche IP.

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

Titre : Vers un déploiement incrémental programmable et évolutif d'ICN dans le monde réel

Mots clés : ICN, Déploiement Incrémentiel, Transport, Socket API, HTTP, WebRTC, Gestion Réseau, Configuration Réseau, Contrôle Réseau

Résumé : Réseau centré sur l'information (ICN) englobe une ensemble d'architectures réseaux repensant les principes de communication Internet autour des données nommées. Après plusieurs années de recherche et l'émergence de quelques propositions populaires, l'idée de remplacer TCP / IP par un réseau centré sur les données reste objet de débat. Les avantages du ICN ont été préconisés dans le contexte des réseaux 5G pour la prise en charge de modèles de communication multi-accès / source, à latence minimale et avec plusieurs utilisateurs mobile. Toutefois, des tests à grande échelle et une insertion dans des réseaux opérationnels doivent encore être réalisés, probablement en raison de l'absence d'une stratégie de déploiement incrémental claire. L'objectif de cette thèse est de proposer et d'évaluer des solutions efficaces pour le déploiement de l'ICN.
Tout d'abord, nous proposons Hybrid-ICN (hICN), une intégration ICN dans IP (plutôt que sur / sous / à la place de) qui a pour ambition de ne pas échanger les principes architecturaux de ICN. En réutilisant les formats de paquets existants, hICN introduit de l'innovation au sein de la pile IP, nécessitant un minimum de mises à niveau de logicielles et garantissant une interconnexion transparente avec les réseaux IP existants. Deuxièmement, la thèse est centrée sur le problème du déploiement de l'ICN aux extrémités du réseau, notamment l'hôte final, en concevant une infrastructure de transport et une API de socket pouvant être utilisées dans plusieurs architectures ICN telles que NDN, CCN et hICN. Le cadre favorise les technologies de pointe visant à fournir des performances et une efficacité aux applications. Une analyse comparative détaillée à la fin du chapitre présentera les performances du cadre de transport.
Ensuite, les avantages que les services de transport et de réseau hICN peuvent apporter aux applications seront évalués en considérant deux principaux cas d'utilisation: HTTP et WebRTC. Le premier représente le protocole de facto du Web, tandis que le second est une nouvelle technologie émergente de plus en plus adoptée pour les services en temps réel.
Enfin, la thèse propose une solution pour déployer par programmation, configurer et gérer des réseaux et des applications ICN: Virtualized ICN (vICN), un cadre unifié et programmable pour la configuration et la gestion de réseaux, qui utilise les progrès récents en matière d'isolement des ressources et de techniques de virtualisation. Il offre une plate-forme unique, flexible et évolutive pour répondre à différents objectifs, en particulier les déploiements réels de l'ICN dans les réseaux IP existants.

Title : Towards a scalable and programmable incremental deployment of ICN in the real world

Keywords : ICN, Incremental Deployement, Transport, Socket API, HTTP, WebRTC, Network Management, Network Configuration, Network Control

Abstract : Information-Centric Networking (ICN) embraces a family of network architectures re-thinking Internet communication principles around named-data. After several years of research and the emergence of a few popular proposals, the idea to replace TCP/IP with data-centric networking remains a subject of debate. ICN advantages have been advocated in the context of 5G networks for the support of highly mobile, multi-access/source and latency minimal patterns of communications. However, large-scale testing and insertion in operational networks are yet to happen, likely due to the lack of a clear incremental deployment strategy. The aim of this thesis is to propose and evaluate effective solutions for deploying ICN.

Firstly, we propose Hybrid-ICN (hICN), an ICN integration inside IP (rather that over/under/ in place of) that has the ambition to trade-off no ICN architectural principles. By reusing existing packet formats, hICN brings innovation inside the IP stack, requiring minimal software upgrades and guaranteeing transparent interconnection with existing IP networks.

Secondly, the thesis focuses on the problem of deploying ICN at the network endpoints, namely at the end host, by designing a transport framework and a socket API that can be used in several ICN architectures such as NDN, CCN and hICN. The framework fosters cutting-edge technologies aiming at providing performance and efficiency to applications. An extensive benchmarking at the end of the chapter will present the performance of the transport framework.

Subsequently, the benefits that hICN network and transport services can bring to applications will be assessed, by considering two main use cases: HTTP and WebRTC. The former represents the de-facto protocol of the Web, while the latter is a new emerging technology increasingly adopted for real time services.

At last, the thesis proposes a solution for programmatically deploying, configuring and managing ICN networks and applications: Virtualized ICN (vICN), a programmable unified framework for network configuration and management that uses recent progresses in resource isolation and virtualization techniques. It offers a single, flexible and scalable platform to serve different purposes, in particular the real deployments of ICN in existing IP networks.

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