Content-centric Networking

Master Thesis

of

Frederik Van Bogaert

September 6, 2014

Supervisors:
Dr. Maarten Wijnants (Coordinator)
Prof. dr. Wim Lamotte (Promotor)

Universiteit Hasselt

Eindwerk voorgedragen tot het behalen van de graad van master

in ICT

Abstract

The current internet model, which relies on packet switching, has served us well and led to a huge explosion in connectivity of both computers and people, and created a huge new industry. However, the way the internet is used today (which is mainly viewing different kinds of content) is increasingly at odds with its original intention of connecting computers with known addresses to each other. This thesis examines several technologies, collectively known as "Named Data Networking", which aim to fix this discrepancy by placing emphasis on the content being transferred, in order to create a more durable, secure and efficient internet for tomorrow.

In this thesis, several Named Data Networking technologies are examined, including their philosophies, advantages and disadvantages. Particular interest is given to "Content-Centric Networking" (CCN), a promising technology with a lot of current research. The theoretical and practical issues concerning implementation are also addressed, by giving a discussion of a Voice-over-IP replacement using CCN, as well as a detailed look at the real-world performance of the technology in the field of multimedia dissemination using Dynamic Adaptive Streaming.

In this examination, it is discovered that CCN faces some serious performance hurdles requiring significant fine-tuning, but that HTTP-like performance can be achieved in some cases. Given the obtained results, the viability of replacing the IP stack with CCN is discussed.

Preface

I would like to thank my coordinator, Maarten Wijnants, for providing me with the relevant data and for thoroughly checking draft versions of this document. Thanks also to my promoter, Wim Lamotte and to the EDM staff for giving me the resources necessary for my thesis. Finally I would like to thank my parents for their patience.

Contents

Part I

Literature study

Chapter 1

Introduction

The internet is commonly considered one of the greatest inventions of the 20th century; and like all great inventions, any possible improvement to it could prove immensely useful to society. Despite its rapid pace of development, there are still many ways to improve the security, efficiency and ease-of-use of the internet. One possible route for improvement is to access content on the internet by name instead of by address. This technology is alternatively named **Content-Centric Networking (CCN)** or **Named Data Networking (NDN)**. The Internet Research Task Force (IRTF), which is the longer-term focused parallel organization to the Internet specification producing Internet Engineering Task Force (IETF), has established a research group that works specifically on **Information-Centric Networking (ICN)**.

In this thesis, 'Named Data Networking' is used to refer generally to techniques to access content by name on the internet, while 'Content-Centric Networking' refers to one specific implementation of this philosophy.

This dissertation starts out by describing the reasons for upgrading the current internet stack with alternatives based on NDN, and the advantages such an approach can bring (chapter 1). After that the principles and issues surrounding NDN are described (chapter 2). Next CCN in particular is treated (chapter 3), followed by a study of a Voice-over-CCN implementation (chapter 4), a comparison of other NDN techniques (chapter 5) and a look at the main weaknesses of CCN (chapter 6). The main findings are then sum-

marized (chapter 7). This concludes the first part of the thesis. The second part is concerned with an in-depth study of the performance of CCN. Various research targets and approaches are discussed (chapter 8), followed by an examination of the performance of CCN for file transfer in general (chapters 9 and 10), and applied to adaptive streaming of multimedia content (chapter 11). A few other experiments round off the experimental phase (chapter 12). Finally, the findings are discussed (chapter 13).

1.1 Motivation

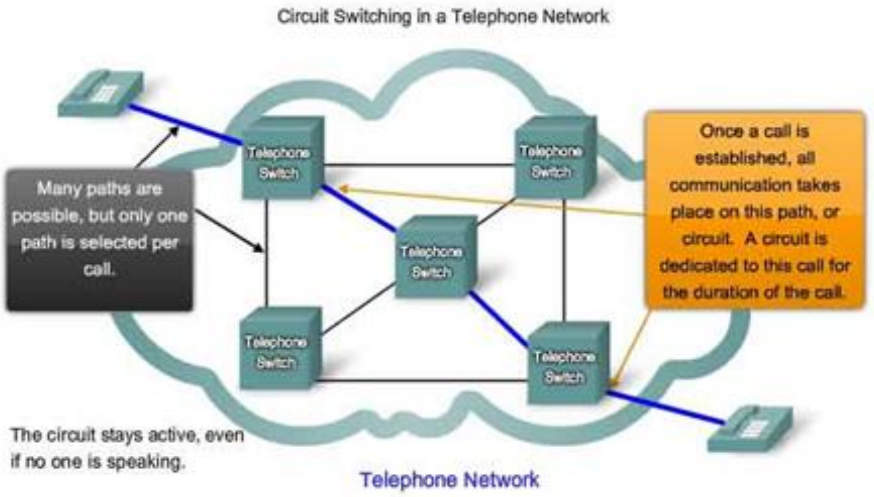
To understand the context of NDN and CCN, it is important to understand why people are looking for alternatives to traditional IP networking. In this section, both the reasons for introducing IP in the first place, and the reasons for moving beyond IP are discussed.

1.1.1 The origin of IP

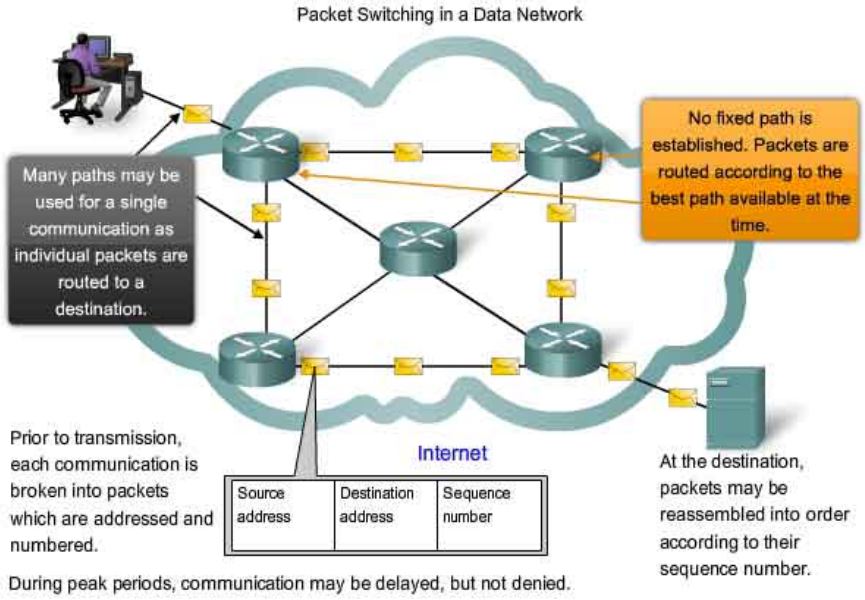
Before the introduction of packet switching, networking between two computers was universally done in the same way as the telephone network was implemented: the problem the networking community intended to solve was to connect two hosts together by first setting up a communication path between the involved parties. This approach is known under the term **circuit switching** (see figure 1.1(a)).

There are four main issues with the telephone model, however, that make it unsuitable for data networking [Jac06]:

Scaling It is difficult to successfully scale this approach. Because the telephone network is a hierarchical system, and because resources need to be reserved in all nodes on the path between the caller and the callee, the amount of bandwidth necessary in the links between the top nodes in the hierarchy is consider-



(a) Circuit switching



(b) Packet switching

Figure 1.1: Architectural comparison of circuit and packet switching. Source: [Int]

able. In data networking, a lot of bandwidth that was reserved would not be used, leading to an inefficient use of resources.

Stability The telephone system is a rigid central hierarchy. As a result, all the nodes (exchanges) in between the hosts, as well as all

the links (cables) between them need to be in perfect working order for a call to succeed between two hosts. Adding more infrastructure between callers decreases the total reliability of the telephone network because the system strength is determined by the weakest link.

Setup time The telephone network first sets up a path between the caller and the callee, a process that guarantees an uninterrupted connection. However, this connection setup takes time. If all you need is to transmit a small file (for example, 5 kB), the setup time is disproportionately large in comparison to the time it takes to send the file.

Efficiency A system which can only work in a fixed hierarchical fashion and which reserves resources all along the way is simply not an efficient system for distributing (short) messages.

Packet switching (see figure 1.1(b)) fixed these problems. In packet switching, the data is split into independent packets which each contain its source and destination address, so they can be routed individually, at each node, without any prior setup of the connection (see figure 1.1 for a comparison).

Scaling becomes less of a problem when no resources need to be reserved, the network only uses the amount of bandwidth that is absolutely necessary. Because each intermediate node decides how to forward a packet, the network can recover from link failures without manual intervention, and more links in the network increase the stability of the network as a whole, instead of decreasing it because each extra link provides an alternative path for packets to move along from source to destination. There is no longer a delay to set up a connection. By switching from a link-focused view to a conversation-focused view, networking between computers was significantly simplified, leading to big gains in efficiency, reliability and scalability.

1.1.2 Obsolescence of IP

However, today the conversation paradigm itself is obsolete. In the early days of networking, computers were big, expensive and static.

The main problem that the network architects were trying to solve was establishing usable links between two computers (hosts) on different networks. Now, however, we spend more time trying to find out where (on which host) content we wish to access (such as a web page, a paper or a radio live stream) is located, so that we can access it. This indirection is a direct consequence of the current internet design, and it is unnecessary. Just like the focus on the telephony model complicated the implementation of effective networking between computers, the enduring focus on the conversation paradigm[1] makes most tasks that are executed today more difficult. In particular, the following issues arise with the current, TCP/IP-based architecture [Jac06, MPZ10a, KCC+07, MA11]:

Mobility When packet switching first gained momentum, all computers were big, hulking monsters that were tightly attached to the ground; now mobile computing is on the rise. The fact that hosts can move about is something that was not anticipated when the IP stack was designed and now leads to awkward workarounds (see section 2.3). The explosion in popularity of smartphones makes this matter even more urgent.

Caching Often, caching and content delivery networks (**CDN**s) are used to shield hosts from the effects of high demand for a specific resource on the web and to reduce start-up delay, but there are situations where this is not possible, such as live streaming. A router may have thousands of different IP 'conversations' all trying to access the same resource, but the router itself cannot tell, and may be forced to send the same content up- and downstream thousands of times. For instance, if 10 users on the same network are watching the same live stream, the edge router of the network will send the same requests for content upstream 10 times, receive the same content 10 times, and distribute it to the 10 hosts. CDNs are these days also used to distribute live video streams to many users. They however rely on complicated DNS techniques that can potentially be replaced by CCN.

[1] The view that all data flowing over a network is a conversation between two hosts

Complexity The development of new protocols for accessing data is becoming increasingly complex (for instance: Gnutella, bittorrent, ...). The main reason that creating new protocols is and remains such a nuisance is that the IP network cares only about paths to hosts, not about data. To determine how to get the data, one must therefore first discover the location of that data. If the network itself moves the focus to data, protocols layered on top of it, as well as applications can be simplified because they no longer need to worry about where to get their content.

Name persistence If content is accessed through a certain name, that name ideally should not change as long as the content is still available. The fact that many websites are forced to occasionally rethink their layout leads to a great number of broken links and assorted difficulty. This would be even worse if content were habitually addressed by IP address (such as 192.168.2.2:80/index.html) because content providers often switch hosting (and thus move to new servers, with a different IP). Fortunately, DNS and URL rewriting provide partial solutions to this problem, but they still fail if the identifier for the content changes.

Data availability The perception users get from internet services depends on having high availability and low latency, but this is difficult to achieve if the requested data is provided by a single host (in other words, through a single point of failure). Dynamic DNS and content delivery networks like Akamai can resolve this problem by allowing the data to be accessible from several servers, but typically this is not a financially feasible option for small websites, which means that problems such as the Slashdot effect[2] are still fairly common on the internet.

Authenticity Most of the data on the internet is transferred without a guarantee of origin or integrity. Current methods, which rely on a Public Key Infrastructure (PKI), provide a form of authenticity, but they are not ideal because they typically secure the channel over

[2]The Slashdot effect: When content hosted on a small site suddenly becomes very popular, causing that server to be overloaded and eventually even crash due to the spike in demand. See image A.1 on page 135.

which the content is served, rather than to secure the data itself (or even the source). If you have a malevolent ISP, it could substitute the requested data with data of its own, even if the network connection was secured using methods like SSL (see, for instance, [Nig08]). In this model, the security of the data is inherently tied to the security of the server that hosts it; if the host is compromised, it can send malicious data over 'trusted' links. It also relies on having secure mechanisms to identify and locate hosts. Also, a piece of content that has been securely retrieved can still become corrupted after it has been downloaded, and before it is used (for instance, by malware).

1.1.3 Research goal

For the reasons discussed in the previous chapter, several projects have developed over the past decade to try to replace the IP stack with a new networking stack that doesn't suffer from these limitations. In the rest of this thesis, these replacements are discussed on a technical level, and one of them in particular (CCN) is discussed and dissected in more detail. This thesis therefore starts out as a *domain survey* of Named Data Networking technologies, and progresses into a **case study** of CCN, in the field of smart multimedia streaming.

The central research question addressed by this thesis is: **Can CCN provide a viable replacement for the TCP/IP stack?**

Chapter 2

Named Data Networking

Named Data Networking refers to any internet technology that places content ('named data') centrally, reducing the location of that content to a secondary role, to improve the efficiency of content distribution.

2.1 Concepts and terminology

These are some of the terms and concepts that are referred to in the rest of this thesis:

Public Key Encryption In traditional symmetric key encryption, the same key is used for encryption and decryption purposes. The disadvantage of this method is that a key needs to be agreed between the communicating parties in advance. This needs to happen in such a way that the key cannot be intercepted. Public key encryption solves the problem by using a different key for encryption and decryption. A person using public key cryptography will have a *private* key that they keep to themselves, and a *public* key that is available to everyone. Messages encrypted with the private key can only be decrypted using the public key. Because there is only one

person that holds the private key, only one person can encrypt a message that can be decrypted with the public key. Also, messages signed with the public key can only be decrypted using the private key, which means that messages signed with a public key can only be decrypted by the owner of the corresponding private key [RSA78].

Cryptographic signatures NDN technologies typically use signing to ensure the integrity of data. Before sending the data, the sender generates a cryptographic signature of the data. A cryptographic signature is a hash of the data, encrypted with the sender's private key (see figure 2.1). The signature of the data can be used by the recipient to verify the authenticity of the data, by decrypting the signature with the sender's public key and comparing the result against the hash of the data. If the decrypted signature matches the hash of the data, it means that the data came from the correct source and that it has not been altered.

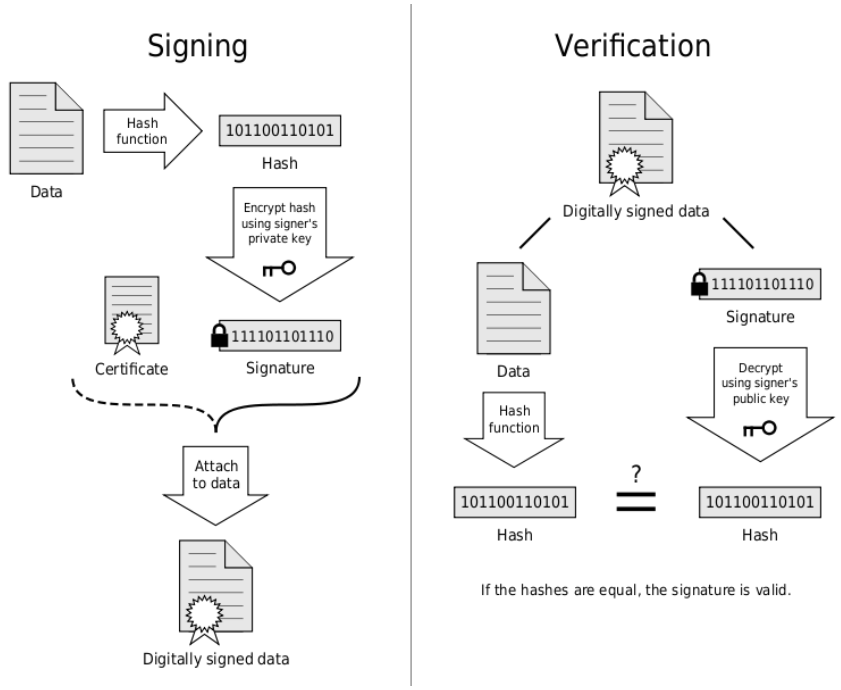


Figure 2.1: The use of digital signatures to perform content authenticity verification [Sig].

Nonce A pseudo-random number used to identify a cryptographic transaction to make replay attacks[1] impossible.

Autonomous System (AS) A collection of connected IP routing prefixes under the control of one or more network operators with one routing policy on the internet. Routing between multiple ASs currently happens using BGP (Border Gateway Protocol, see [BGP]).

Names A crucial concept in NDN is the **name** of the data. This is because NDN attempts to widen the gap between an *identifier* of data (the name) and a *locator* of data (the means of retrieval). In IP networking with DNS, the difference is marginal: an URL is both an identifier and a locator of data. NDN instead insists that a piece of data should only ever have a single identifier, even if it is available from multiple locations. Furthermore, names need to be **persistent** and **contention-free** (no disputes about the ownership of data). If data names are organized in a hierarchical fashion, for instance, should the data name be 'company.project.person.paper' or 'person.paper'? What if that person changes company? In other words, it is very hard to organize data names in a hierarchical fashion if they are not supposed to change. A different approach is to use a flat (non-hierarchical) namespace based on a cryptographic hash of its content. This has the added advantage that this kind of name can be **self-certifying** (see next paragraph). Both approaches are common in NDN technologies.

Self-certifying names A self-certifying name is a name that, by itself, can be used to verify the **integrity** and/or the **authenticity** of the data. A name can be used to verify the integrity of data if it includes a hash of the content it describes. In the same way, a name can be used to verify the content's authenticity if it includes, for instance, a hash of the public key of the entity (person or company) that authored (or published) the data.

[1] A replay attack involves listening in on a host who is authenticating itself to another host, capturing all the traffic between the hosts and simply retransmitting it to authenticate itself as well.

Distributed Hash Tables If a flat namespace is chosen, traditional hierarchical routing (like DNS) becomes impossible. Instead, Distributed Hash Tables (DHT) can be used for routing. A Distributed Hash Table is a routing system (often used in peer-to-peer networks) which is decentralized, scalable and largely self-organized. With N hosts in the network, it allows content to be routed in at most *O(log N)* steps with *O(log N)* states in the routing table, although a trade-off can be made to store more data in routing tables in return for a reduced number of steps and smaller latency [ADD+08].

Publish-subscribe NDN technologies can also integrate publish/-subscribe functionality [CAFR12, CAJ+11]. Usually on the Internet, a client/server architecture is used where a client requests a file or a feature, and a servers sends a reply. For many use cases however, it would be advantageous to instead use push messaging: the server notifies everyone interested when a resource they care about becomes available or is modified. For example, e-mail clients need to regularly poll their server for new e-mail, where it would be more efficient if the e-mail servers could simply push the e-mail to the client when it arrives. Publish/subscribe systems allow a client to *subscribe* to an event to register its interest in it; when an event happens, the server *publishes* the event to all subscribers. Systems for integrating publish/subscribe mechanisms into NDN are discussed in section 5.5.

Adaptive Streaming Adaptive Streaming (or Dynamic Adaptive Streaming over HTTP: DASH, see 2.4.2) is a relatively new phenomenon on the internet but is already being widely adopted in different forms. It refers to streaming multimedia content in such a way that the quality-of-experience[2] is adapted to the quality of the network link to the consumer. As an example, if a content consumer is traveling and watching a video on a smartphone, the server will stream high quality frames to the consumer when he or she has good coverage/Wi-Fi connectivity, and will fall back to lower quality content when switching to cellular internet connectivity or when coverage drops. Adaptive streaming delivers different quality encodings of the multimedia stream to each screen (customer) even

[2] Quality of Experience (QoE): The perceived quality of the source media as observed by the audience.

for live broadcast TV streams, instead of relying on multicast technology to send the same quality stream to each consumer. As most of the current internet traffic is multimedia in one form or another [San], analyzing the relation between NDN and Adaptive Streaming is an important topic that we cover in section 2.4.2.

2.2 Internet Addressing

Addressing in IP In the IP system, each computer typically has a fixed IP address which is divided in a 'network' part and a 'host' part. Intermediate routers only need to know the 'network' part to figure out where to send a packet destined for a specific IP address to. Normally every computer that is connected to the internet has a distinct IP address, but a technology called NAT (Network Address Translation) is often used to provide multiple computers within a small network with the same outside IP address. For more information about IP addressing and routing, see [Lin06].

Addressing in NDN Most NDN technologies discussed in this thesis propose a different underlying architecture from IP, with the idea being that globally routable identifiers are not really necessary on a lower level. Instead, they can be replaced with 'path labels', where the return address of a packet is given by a series of labels, or domain identifiers for each node along its path. When a packet is sent towards a server, the node appends its identifier to the path labels. The return packet then 'unwinds the stack': it uses the last path label to forward the packet to the first hop towards the client, which then uses the second topmost label for the next hop, and so on. Take the example in figure 2.2. Here, the client sends a request to host X which passes through 2 intermediate domains (let's call them D1 and D2). It uses its local address on the client domain as the return address (A). This address is only routable within the client domain (much like the 10.x.y.z addresses in IPv4). Each intermediate domain has a number of interfaces over which it can receive and transmit data, and each interface has a label. D1 receives the request from host A on the interface it calls B, and appends this label to the return address (labeled src in the figure). It then sends the packet onward to the next domain, which receives the request

on the interface it labeled 'C'. This label is added to the src address and sent onwards to the final domain. It reaches the domain of the server on the interface labeled D. The return address is thus constructed domain-by-domain, and consists only of identifiers that are exclusively valid within their specific domain. The reply then has to take the exact same route back. The server domain sends the packet over the interface it labels 'D', and then strips this label from the destination address. The next domain sends it back over its interface 'C', and strips that label from the destination address.This way, the reply can only be sent along the same route as the request came [HG04].

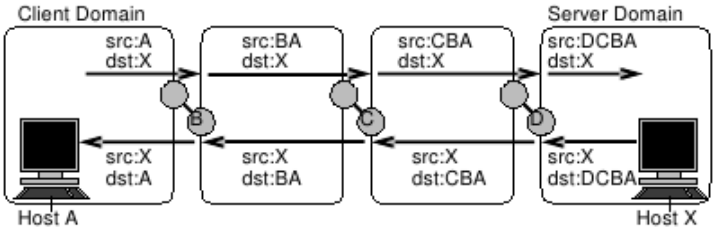


Figure 2.2: Path-based addressing [HG04]

There are many security advantages to using path-based addressing as opposed to IP [HG04]:

- The client return address as known to the server is only valid for the domain the server is in. That means that if an attacker manages to 'harvest' client addresses from a server, they will be of very limited use to him. This severely limits some types of exploits on the internet.
- The client cannot spoof his address, which means that attacks on a server are easily traceable back to its source. This also means that offending clients can easily be blocked (in IP networking, it is common for attackers to spoof their source IP address to keep servers from filtering them out based on IP addresses).
- It is easier for transit ISPs to monitor the network for malicious traffic, by checking if traffic in one direction is followed by traffic on the reverse path (this is not possible in IP, because in IP the reply can take a different path than the original query).
- Because clients do not have a globally routable address anymore, it is very hard to compromise the security of a client

computer unless one of the servers the client tries to access is compromised (whereas in IP, if you know the IP address of your target and the target is online, you can attack it)[3].

The security aspect of NDN is discussed in more detail in section 6.2.

The major disadvantage of path-based addressing is that client addresses are inherently unstable: if the client moves location or if the inter-domain routing changes, the client address will change, which poses a problem for traditional transport connections that require that the underlying address remains the same[4].

2.3 Advantages of NDN

NDN is designed to alleviate the problems with the IP infrastructure as noted in section 1.1.2. Several other advantages have been identified [MPZ10a, Jac06, LRH10, Lau10]:

Mobility Most NDN technologies, like CCN, greatly reduce the complexity and issues of implementing mobile networking because they require less state setup. In particular, they do not need a fixed address for a host, and allow a response to simply take the reverse path of the request, which means a mobile host can send requests to one base station until it detects that the signal of a different base station is stronger, at which point it starts to send requests to the other base station. No reconfiguration on the network layer is required. A concrete protocol for roaming internet over CCN is discussed in section 2.4.1.

[3] Unless this is blocked by NAT, a firewall or some other defense

[4] This is not an issue in most cases, because the relevant protocols do not use dedicated connections.

Energy efficiency Servers and routers in the internet expend considerable energy in serving and routing content to end users. The U.S. Environmental Protection Agency estimates that over the course of 2011, servers and data centers combined consumed about 100 billion (10^{11}) kilowatt hours [LRH10]. Content-centric networking allows for a more energy efficient distribution of content. The issue of energy efficiency is explained in more detail in section 6.6.

Latency Many NDN technologies decrease the latency experienced by end users in typical circumstances. For instance, CCN usually requires Interests to traverse only a few hops, because of the ubiquitous caching of content. This means that access times (latency) should decrease for most content compared with IP (especially popular content).

Consistency CCN and other NDN technologies have a 'fate-sharing' property between routing of content requests and retrieval: if the content can be located, it can be retrieved. This is not always the case in the current internet today: it may be possible to resolve the DNS name of a server on the internet to an IP address, but this does not mean that the server is 'up' or that the desired content on that server is still accessible. This means that with CCN, there will be no more HTTP 404 (File not found) errors caused by broken links. Note that it may still be possible that content cannot be found because it has since been revoked, but at least the user is certain that content that cannot be found does not exist.

Server load reduction Because the NDN architecture allows and encourages caching of content in routers, the load on the original servers will be a lot lighter, and content servers that are suddenly flooded with requests for popular content should become a thing of the past. It will also reduce or eliminate the need to work with a Content Delivery Network for popular content.

Authenticity Instead of trusting hosts or connections, NDN directly authenticates the binding of names to data using crypto-

graphic signatures, which means that better guarantees of authenticity are possible.

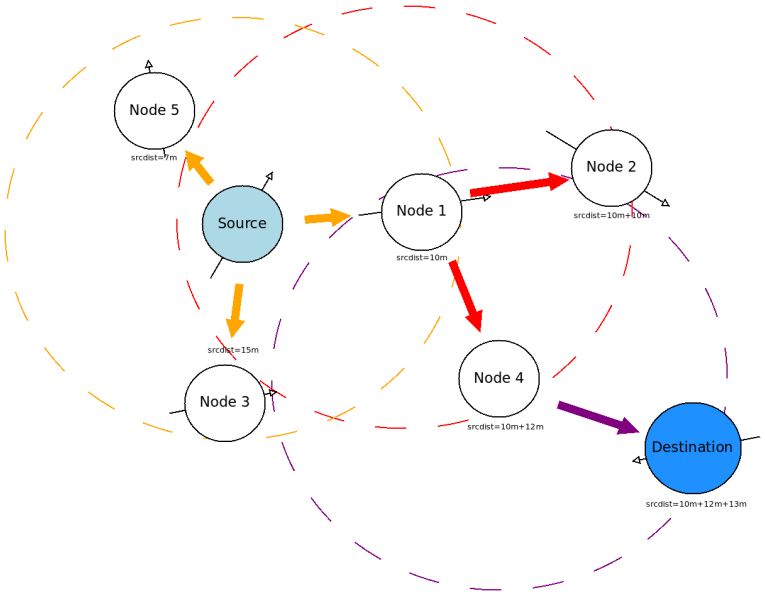
Easier content delivery Due to the popularity of content on the internet, content has to be distributed and cached close to consumers. This is particularly true for popular video files that can be accessed on demand. Additionally, the internet is more and more used to deliver live content, as a replacement of traditional television services. Consumers are turning to tablets, smartphones and PC's to watch live streams. The industry is moving to a new standard called DASH [MA11] for live streaming of audiovisual content. In section 2.4.2 we discuss the possibility of using CCN in conjunction with DASH for optimal adaptive video streaming.

2.4 Initial applications

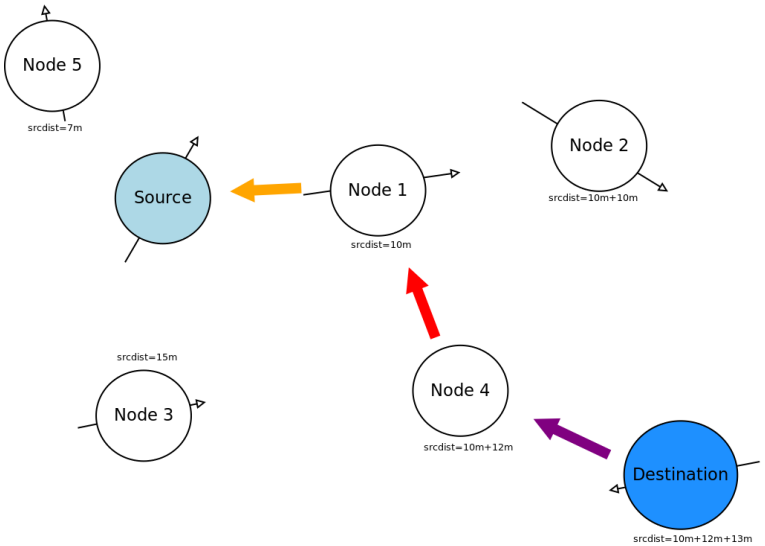
While NDN is generally considered a replacement for the entire IP stack, its advantages carry greater weight in some areas than in others. In particular, the following applications could benefit directly from NDN and perhaps spearhead its adoption.

2.4.1 LFBL

Listen First, Broadcast Later is a protocol designed for use with wireless roaming and mobile hosts, based on Content Centric Networking [MPZ10a, MPZ10b]. By making full use of the inherent broadcast nature of wireless communications, it attempts to find as efficiently as possible a way for disseminating data in a wireless environment where base stations may move about and hosts can come and go. In order to do so, LFBL minimizes its housekeeping and uses data from overheard packets to build up a dynamic picture of the network on each node, so that each node has a rough idea of its distance to the other nodes in the network. A host that desires data will broadcast a request over the wireless network that contains the name of the data. Each host (node) that receives such a request



(a) Request forwarding phase



(b) Response forwarding

Figure 2.3: LFBL request/response routing

will then determine if it is an eligible forwarder (if it is closer to the intended destination than the source of the request). They will wait a short time to see if another node forwards the packet first (to avoid flooding), and then broadcast the packet with a header that includes the cumulative distance from the source. Each node along the way can thus update their own internal estimates regarding the

distance of various other nodes in the network, by remembering the source distance mentioned in the overheard packets. When the intended destination of the packet receives the request, a response is generated which will use the same path as the request back to the source of the request. Unlike true CCN, LFBL does not store routing prefixes, because of the dynamic environments it is intended to be used in. Instead, requests are always broadcast to all recipients.

In figure 2.3, an example is shown. Here, the source broadcasts a request for some data provided by the node labeled "Destination". Each intermediate node will check if it is closer to the destination than the sender, wait for a little while and if no-one else forwarded the request in the mean time, forwards the request using another broadcast. First, the source broadcasts the initial request to all hosts in range, Node 1, 3 and 5. Node 5 realizes that it is more distant from the destination than the source and silently drops the request. Nodes 1 and 3 wait for a while. Node 1 broadcasts first in this scenario, which means that Node 3 also discards the request. Before broadcasting, Node 1 added its own distance from the source to the request. The request is then caught by Node 2 and Node 4. Node 4 then forwards the request to the destination. The reply travels over the same line as the original broadcast, using the Pending Interest Table (see section 3.3).

The architecture of LFBL is based on NDN because of the inherent difficulties of fixed routing tables, and IP address assignment, in a highly mobile, dynamic environment. It is assumed that not only the hosts switch location, but the requested data may also travel between hosts.

2.4.2 DASH

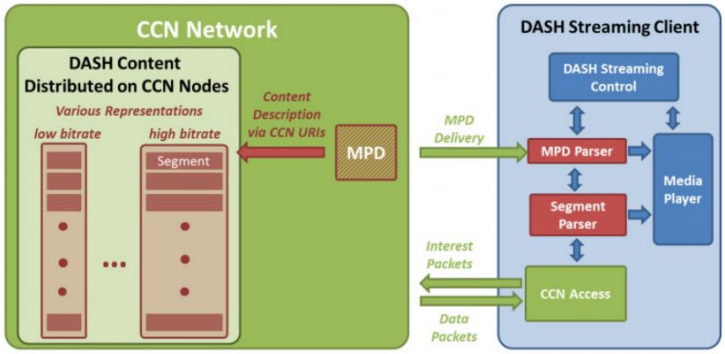
DASH stands for Dynamic Adaptive Streaming over HTTP. It is the standardized version of the techniques that Apple (HTTP Live Streaming), Microsoft (Smooth Streaming) and others have introduced to handle delivery of internet (HTTP) video over a network with varying and unstable performance. With the growing success of web based video delivery like Netflix, Amazon, Apple, YouTube, this technology is rapidly gaining importance. Even “classical” TV providers

are converting their service delivery model from UDP multicast or unicast streaming towards HTTP adaptive streaming. Examples in Belgium include Yelo from Telenet [tel13] and Belgacom's TV Everywhere [bel]. Netflix reportedly uses almost one third of the US Internet traffic every night on its own [San].

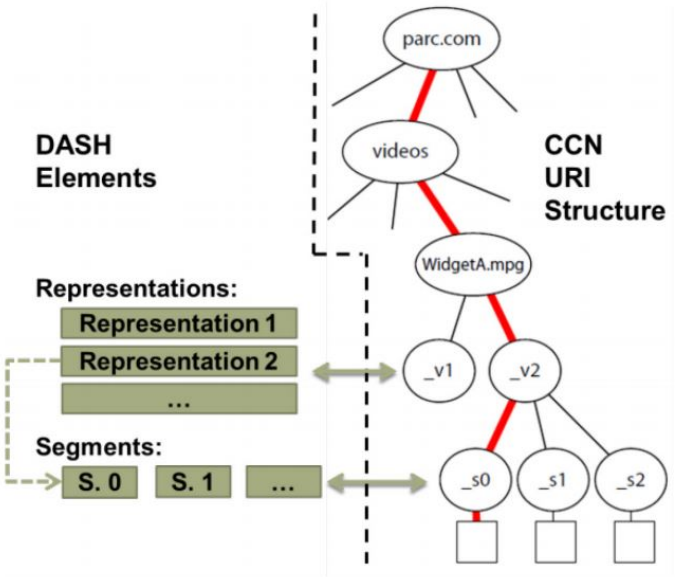
With adaptive streaming, a large media file is subdivided in smaller segments, that are typically a few seconds in length. Each media file is encoded in different quality levels, with segments for each quality level being generated. The media player (client) will sequentially download each segment (HTTP GET). With each download, the client will determine the quality of the connection by measuring the media throughput on the network link. The client will get the next segment with the highest possible quality that is sustainable by the network connection. The selection also depends on device capabilities (screen size, resolution) and user preferences. A DASH encoded stream comes with a Media Presentation Description (MPD) which is a XML file that contains the segment relationships and description (timing, URL, bitrate, resolution,..). Before starting to play the media file, the client will first download the MPD. DASH content delivery is entirely pull-based, i.e. the client is in control of the streaming process. As the delivery mechanism is HTTP, media can be distributed using normal CDNs and caching logic.

As pointed out in [LMR+13b], there are several elements that DASH and CCN have in common, namely the fact that content is handled in segments / chunks and that both rely on a client driven pull approach. DASH representations can be mapped to CCN content versions. Moreover, the intensive use of CDN caching for DASH video can be considered as the equivalent to the CCN principle of caching content within the network. Since the two technologies are operating at different protocol layers - DASH at application and CCN at the network layer - they potentially can be combined and leverage the advantages of both. An approach is suggested where the DASH client has a native CCN interface and adopts the CCN naming scheme (CCN URI) to refer to video segments in the MPD. See figure 2.4

The client would send a CCN Interest packet that reflects the desired characteristics of the segment, like bitrate, resolution, language... within the content name. This fact, combined with the



(a) Architecture



(b) Naming Structure

Figure 2.4: DASH over CCN [LMR+13b]

4kB chunk size of CCN, leads to a huge increase in transmission overhead for CCN compared with HTTP DASH. Moreover, CCN is less efficient than when the HTTP 1.1 feature of pipelining, i.e. usage of multiple simultaneous TCP requests, is used. Nevertheless these disadvantages might trade-off for the intrinsic advantages of CCN like caching and implicit multicast support. Further issues are identified [AVS] that hamper the use of DASH in CCN.

- Naming of data is not aligned with the NDN conventions, in

that all segments might not share a common prefix

- For video delivery, ownership of the content (and authentication of clients that have the right to watch) is often linked with the location from which the content is retrieved.
- The nature of DASH, which estimates the achievable quality by the download rate, assumes that network conditions are more or less the same from video segment to segment, which is the case if the same CDN is being used. This is not compatible with the CCN concept that location should be of no concern when asking for content. This might also introduce unwanted oscillations in video quality. If a standard quality is cached close to the user, it will be retrieved fast, leading the client to ask the next segment with a higher quality. If this one is cached on a remote location, the client will go back to standard quality.

2.4.3 Other possible applications

Named Data Networking has been proposed for several other applications where IP would not work equally well:

Sensor networks Sensor networks are made of small devices that can only spend a small amount of energy on communication. Since the content of the data that sensors want to communicate is more relevant than the identity of the node itself, a data centric approach can be beneficial. An approach is suggested [SRK+03] based on Distributed Hash Tables for identifying the sensor and the data key. For routing GPSR (Greedy Perimeter Scalable Routing), which is a geographic routing system for multi-hop wireless networks, is used in such a way that the closest node that can serve or store the request is used.

Vehicle-to-vehicle communications There are many interesting applications in the field of vehicle-to-vehicle communications such as information sharing for safety, real-time traffic updates and similar. V2V communications are also an important aspect of driverless cars. NDN is of interest here [WWK+12] because it allows vehicles to directly request the desired data from neighboring nodes (i.e. other vehicles), without the need of an IP

layer with all its additional overhead of address allocation and routing. The key to success will however be to define and agree upon a common naming scheme that allows the requester to express exactly what data he wants and suppliers to describe exactly what they have to offer.

Real-Time applications As already discussed in the DASH section, applications like video streaming can create a lot of overhead with NDN. In [YFX12] it is observed that content users do not frequently change their channel when watching live video, and that conferencing also takes time. So overall content interest does not change with every chunk. It is therefore suggested to introduce a concept of long-term interests for live video streaming and (audio) conferencing. This contrasts with the one-to-one mapping of an interest package with the delivered data package of NDN. A long-term interest means that the client 'subscribes' to a real-time data stream and keeps getting the related content packages until he 'unsubscribes' or the lifetime expires. This will not only reduce overhead but also reduce the memory requirements of the content server. A drawback is that some of the advantages of NDN might not be maintained. More in particular, getting the content from various sources is an issue.

Chapter 3

CCN

CCN is the NDN content-centric protocol promoted by Jacobson et al [JST+09]. CCN was created by Jacobson out of his perception that IP and the surrounding protocols no longer accurately reflect the most common use cases for the internet, which are content-oriented, but server agnostic. CCN is an attempt to create a system where "Content has a name, not a location" [JP09].

CCN is a request/reply architecture where data is cached throughout the network: a host that desires a piece of data requests the data by name by broadcasting a request for that data locally. One of the nodes on the local network can then respond to the request with data, or forward the request further along the chain. It uses special Content Routers that cache content that passes through them based on its popularity.

3.1 Packet design

CCN features two types of packets: *Interest* packets and *Data* packets (see figure 3.1 for an illustration of their layout). CCN uses a simple question-answer paradigm: when a host is interested in a piece of content, it broadcasts its interest in the content using an

Interest packet. A node that observes the Interest and has the requested data may respond using a *Data* packet. Data is only transmitted in response to an Interest, and it satisfies ('consumes') that Interest.

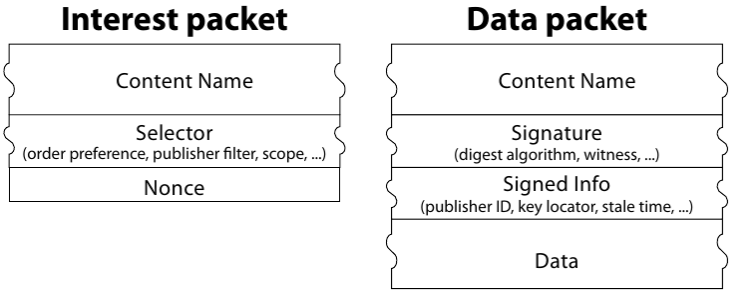


Figure 3.1: CCN packet structure [JST+09]

Interests may also be provided for content that does not yet exist, such as dynamically generated web pages or video content from a live web stream. This allows content providers to generate the requested content on the fly, and to transparently mix cached and dynamically generated content. Routing in CCN uses longest-prefix matching (see section 3.2.1) on the content name. Based on the result of that look-up, a node will either retrieve stored content, or forward the Interest towards potential sources of the requested data.

3.2 Naming

As with all Named Data Networking technologies, CCN addresses content by name, instead of by location. CCN only imposes one restriction on names: they must be hierarchically structured (like IP addresses are). This allows for efficient, hierarchical distributed routing through aggregation of routing information. To achieve some compatibility with traditional networks, DNS names can be used; generally, meaning comes from applications, institutions and global conventions. For instance,

is a valid name, which could refer to this page of this document.

3.2.1 Prefix matching

Because CCN names are hierarchical, prefix-matching can be used for routing. For instance, if one host requests

/example.com/presentation.pdf

a node that has

/example.com/presentation.pdf/page1

can respond with its data. An example of this is given in figure 3.2. Here, an intermediate router gets a CCN request ('Interest') for some data named **/parc.com/videos/other.mpg**. The router does not have that name registered in its FIB, but it does have two names with a matching prefix: one for **/parc.com** and one for **/parc.com/videos**. CCN uses *longest prefix matching*, like IP, so the latter entry is used to make the forwarding decision, and the packet is forwarded over 'face' A[1]. This routing is analogous to IP routing based on addresses, as illustrated in the same figure.

Another advantage of hierarchical names is aggregation: a Content Router does not need to store two separate entries for **/example.com** and **/example.com/images** if requests for both names would both be sent over the same interface; it is sufficient to store the entry for **/example.com**, and let another router further downstream evaluate the rest of the name. This is essential for a stable, scalable network infrastructure.

[1] Interfaces are commonly called faces in CCN parlance

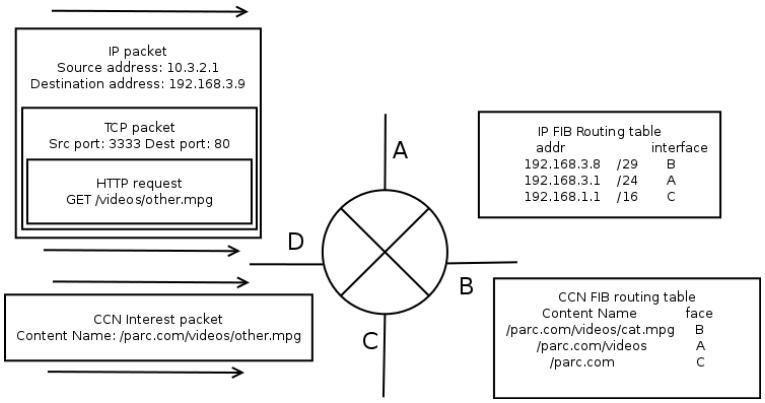


Figure 3.2: Routing decisions in IP and CCN routers: a comparison

3.2.2 Name structure

CCN names are subdivided into several components. Names have to be meaningful to the application, CCN itself only demands that they are properly structured so that they are routable and can be aggregated. Names would typically go further than simply the file name: if a file is sufficiently large, it could not be sent as a single packet, and would have to be split up. Furthermore, there may be different versions of the file available. Because *everything* in CCN is named, these versions and subdivisions (segments) would also have names (see figure 3.3). For example, a client may request the file

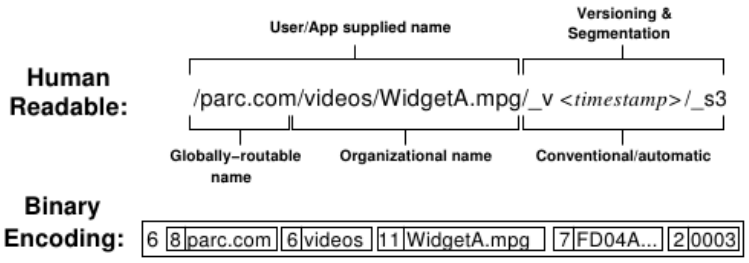


Figure 3.3: Structure and representation of the data names [JST+09]

/parc.com/videos/WidgetA.mpg

and in response get the first segment,

/parc.com/videos/WidgetA.mpg/_v3/_s0.

Because the client does not always know the full name of the content segments he wants to retrieve, CCN names can also be specified relative to other content. For instance, after getting the first segment of data, the client could request

/parc.com/videos/WidgetA.mpg/_v3/_s0 RightMostChild

to get the next segment, continuing in this way until all of the file has been transmitted[2].

3.3 Architecture

The architecture involves three main data structures (see figure 3.4) [JST+09]:

FIB The FIB, or Forwarding Information Base, is used to identify which nodes to forward intercepted Interest packets to. A similar structure exists in IP, where it is also called FIB (see figure 3.2). The CCN FIB stores information about the faces on which a piece of content can be reached. This may be a single interface or a list of interfaces, unlike an IP FIB, where entries are always tied to a single interface.

Content Store The Content Store is a buffer for content. In IP routers, packets are only buffered until they can be forwarded, and are then immediately discarded. In CCN, the Data packets are stored in the Content Store as long as possible, in case the host receives a request for the same data later on. The Content Store

[2]The client could instead request the segments directly because the segmentation rules of the application would probably be known to it

uses a replacement strategy, generally either LFU (Least Frequently Used), which removes packets that are requested with the smallest frequency, or LRU (Least Recently Used), which purges the Data packets that have not been served for the longest amount of time in case of a tie. The Content Store is similar to application-level proxies in a traditional setup, with the advantage that in CCN this is a central feature of the network itself, rather than an extension that violates the separation between layers.

PIT The PIT (Pending Interest Table) tracks Interests that have been forwarded towards possible sources of content, so that the returned Data can be sent back to the requester. PIT entries will be erased once the Data packet is sent back to the requester, or once a pre-set timeout expires (see section 3.4).

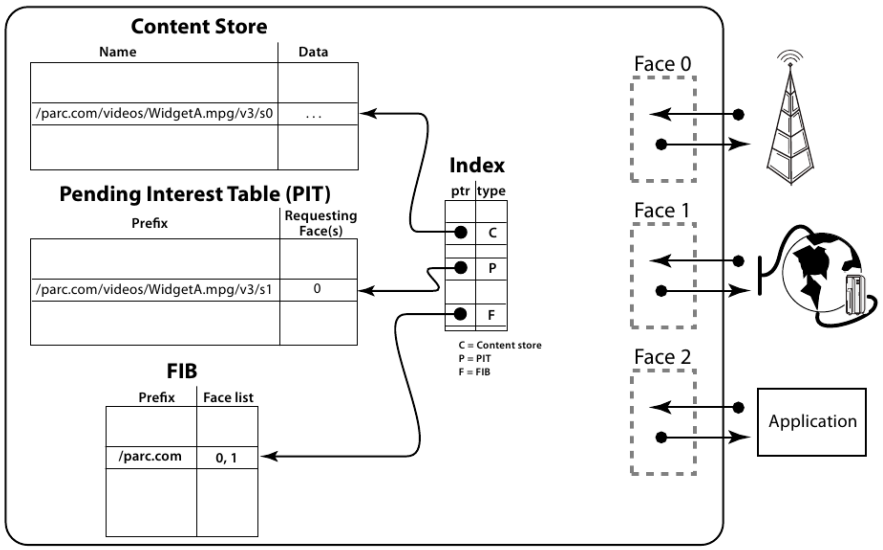


Figure 3.4: The CCN node model [JST+09]

In figure 3.4, we can see that the FIB of this particular Content Router contains an entry indicating that Interests with names starting with **/parc.com** should be forwarded over either face 0 or face 1. The Content Store already contains the data with name

/parc.com/videos/WidgetA.mpg/v3/s0

which it cached as a result of a previous Interest. We can also see that the PIT contains one pending Interest, for content named

/parc.com/videos/WidgetA.mpg/v3/s1

This entry also specifies that when the Data packet for this pending Interest arrives, it should be sent back over face 0.

3.4 Routing

Once an Interest arrives on an interface, a longest-prefix matching look-up is done on the content name. Content Store matches will be preferred, if available. If no matching entry is found in the Content Store, the PIT will be consulted. If there is already someone requesting that exact content, the requester is added to the list of requesting interfaces in the PIT and the Interest will be discarded because it has already been sent upstream. If there is no match in the PIT for this content either, the FIB is consulted. If a match is found, the Interest will be forwarded and the Interest and requesting interface are inserted in the PIT. If no match is found in the FIB, the Interest is discarded. When a Data packet arrives on an interface, a longest-prefix matching is done. If a match is found in the Content Store, the data is discarded because it is a duplicate of what the Content Server already has. If there are PIT matches, the data has been requested by one or more hosts downstream, and the data is first added to the Content Store and then sent back to the hosts in the PIT (see figure 3.5 for an example). Subsequently, the corresponding entry is purged from the PIT.

Any node with access to two or more networks can serve as a content router, including mobile hosts that are periodically connected to different networks, which can use their cache to transfer data between two disconnected areas. Hosts do not need access to the entire internet in order to exchange data; CCN works equally well in disconnected local networks.

Because Data packets follow the PIT entries back to the requester,

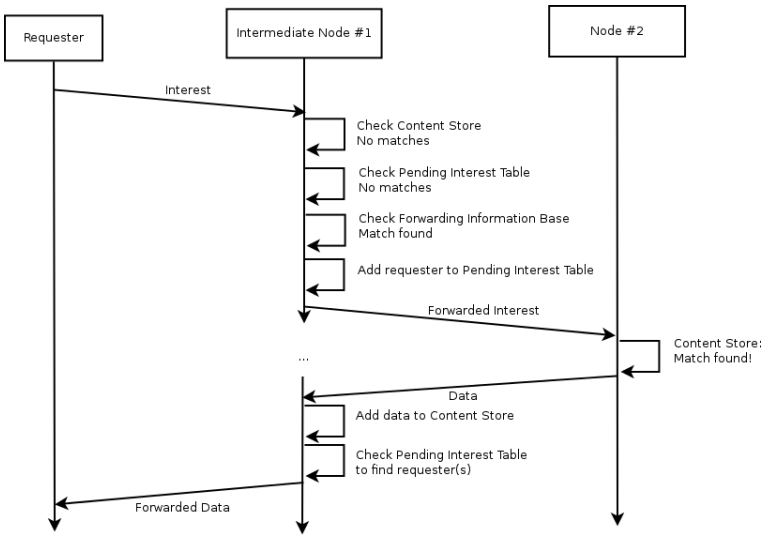


Figure 3.5: An example CCN request

and because Content Routers discard duplicate entries, data cannot loop in CCN. Because CCN's forwarding model is similar to IP's forwarding model (see figure 3.2) and uses the same routing semantics (longest-prefix lookup), routing schemes that work for IP should work for CCN as well with minor adaptations. In particular, CCN routing can be accomplished using established IGP[3] protocols like OSPF and IS-IS [JST+09]. In these routing protocols, all routers possess information about the complete network topology and can make independent decisions about the best next hop for each possible destination in the network. CCN Content Routers can even be attached to an existing routing infrastructure using OSPF or IS-IS without any modifications to the existing network or routers [JST+09], which aids in incremental deployment of CCN within an AS.

A major difference in the routing between IP and CCN is that in CCN all nodes send all matching Interests to all of the announcers, because in CCN a prefix announcement means that *some* content with that prefix can be reached via the router that sent the announcement, whereas in IP the prefix announcement means that all the hosts with that prefix can be reached through the announcing router. CCN routers will forward Interests to all the routers in

[3]IGP: Interior Gateway Protocol, a routing protocol for use within an AS (Autonomous System) http://en.wikipedia.org/wiki/Interior_Gateway_Protocol.

their **FIB** that announced the prefix, which is safe because unlike IP packets, CCN packets cannot loop.

For **inter-domain** routing, it is in the best interest of an ISP to deploy Content Routers in their network once some of their customers start to use CCN to reduce the peering cost (only one copy of the content needs to be fetched from a different domain, the rest of the requests can be served from a local cache). This also reduces the latency their customers experience. The main problems with inter-domain routing of CCN are intermediate domains that do not implement CCN. In that case, the edge Content Routers need to tunnel the CCN requests using ad-hoc UDP tunnels, using a DNS look-up of the other domain's Content Router(s) to determine the IP address for forwarding. This form of routing is a lot less optimal than if Content Routing were supported by all intermediate domains, and may generate considerable overhead for them. To fix this problem, domain-level content prefixes could be integrated into BGP. BGP AS-path information allows each domain to construct an inter-domain topology map for Content, similar to the one for the intra-domain case.

3.5 Transport

Because CCN operates on top of unreliable network infrastructures, packets may get corrupted or lost. CCN relies on a system of retransmissions to achieve reliability, by having the client retransmit Interests that are not followed by Data after a certain timeout. This retransmission is left up to the application itself[4]. The design of the network also means that Interests can loop; a special 'nonce' value is used by the routers to identify specific Interests and to allow them to discard the Interest if it has already been encountered. TCP uses a 'window' of outstanding packets that have not yet been confirmed. Similarly, CCN also allows senders to send new Interests before the Data corresponding to the pending Interests has arrived. Unlike TCP however, CCN does not order its Data responses in the order in which Interests for them were sent, which means one lost packet does not affect the speed of the Data stream as a whole. Optimizing

[4]In some cases, such as live video communication or video streaming, it may not make sense to retransmit Interests

the transport layers of CCN is one of the focus areas for the CCN evaluation in section 8.2.

3.6 Data validation

CCN directly authenticates the binding of content to its name, by using a cryptographic signature in each Data packet over the name, the content and some additional data useful in verification called 'signed data' (see figure 3.1). This way, content publishers can bind content to names they choose. This contrasts with schemes that use self-certifying names (such as DONA, see section 5.2) that securely name content by requiring names to, for instance, contain a cryptographic digest of the content. CCN names can be authenticated by anyone who cares to do so: the signatures embedded in packets are public-key signatures and anyone can verify that a name-content binding was signed with a particular key. Each signed Data packet contains the information necessary to retrieve the public key that is used to verify it (see figure 3.6).

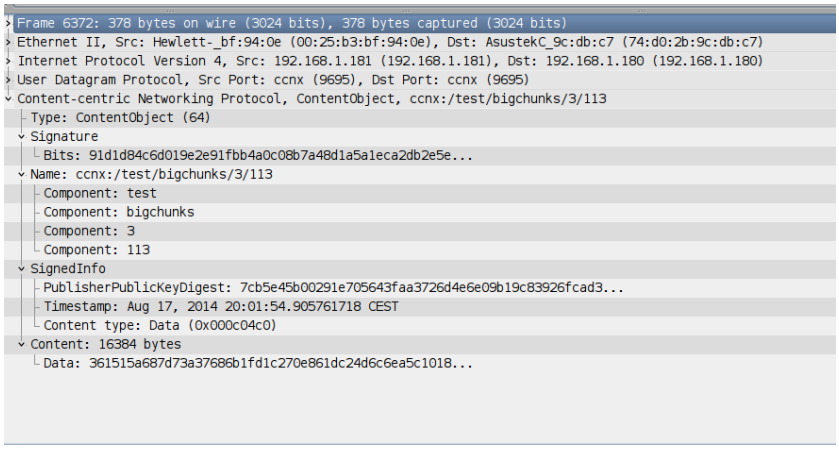


Figure 3.6: CCN packet example, from a Wireshark trace, showing the embedded signature. Note that CCN is running on top of UDP in this example

To verify the content, the consumer needs to request the public key of the publisher, which in CCN is just another piece of content, which is named according to a naming convention. Simply generating a key as CCN content generates a certificate of it, binding a

name to that key as authenticated by the publisher. Because content is organized in terms of hierarchical namespaces, CCN allows the signing policy and keys to be attached to particular namespaces so that the authorization at one level is accomplished by a signature from a higher level (see figure 3.7). For instance, an associate of PARC could be authenticated by the organization (PARC) itself. If the key for parc.com is known, it can then be used to sign its employees.

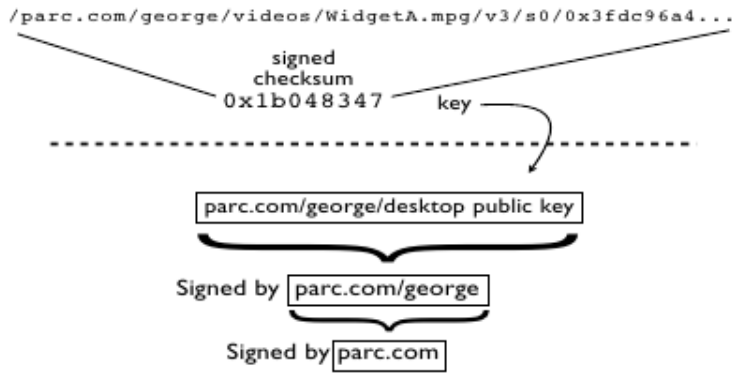


Figure 3.7: CCN trust establishment associating content namespaces with publisher keys [JST+09]

3.7 Content protection

CCN controls access to sensitive content by encryption. Content protection is organized by an application-specific key distribution scheme to allow applications to keep full control over how they encode and distribute keys and content. Following is an example of a client, Alice, buying some secured content on a web page owned by Bob:

1. Alice visits the (unsecured) web page of Bob
2. Alice sends a log in request to a web page of Bob containing Alice's public key
3. Bob verifies that Alice is a registered user of the web site.
4. Bob sends a challenge containing the public key of Bob, encrypted using the public key of Alice

5. Alice replies with the correct answer to the challenge of Bob, proving that Alice has Alice's private key. This reply is encrypted using the public key of Bob. At this point, both parties are confident that they are talking to the right partner. Alice is logged in.
6. Alice buys content, named C, on the site of Bob
7. Alice sends payment to Bob
8. Bob provides a key to Alice to access content C
9. Alice wants to access content C: Alice sends a request containing a request for C and a random nonce value, encrypted using the key provided earlier (the nonce ensures that the content will be addressed under a different name each time, to prevent replay attacks)
10. Bob decrypts the request, extracts the content name and sends the encrypted content back to Alice

Chapter 4

VoCCN: A case study

VoCCN [JSB+09] is an application level architecture for voice conversations. It is a mapping of the VoIP protocols[1] onto CCN, in order to demonstrate that CCN works well in the traditional IP territory of host-to-host conversations, while preserving the security, interoperability, and performance of VoIP. The architecture consists of a SIP/RTP implementation over CCN which provides interoperability with VoIP using a stateless IP-to-CCN gateway.

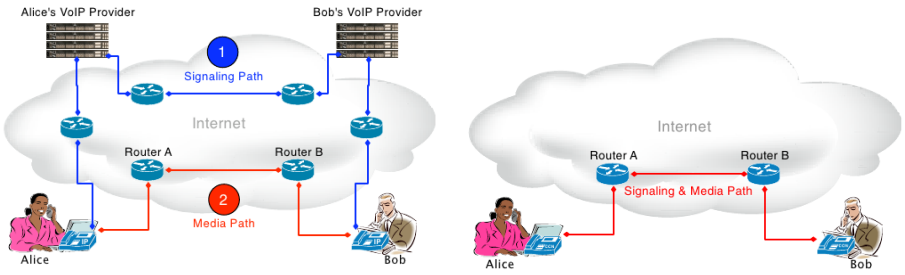
4.1 VoIP

VoIP (Voice Over IP, commonly known as IP telephony) uses a complex infrastructure to set up a link between two (or more) participants. Each participant in the conversation has a VoIP provider (SIP gateway) that keeps track of where they are (the current IP address where they can be reached) at any given time. In order to set up a call, participant A's VoIP provider makes a connection with participant B's VoIP provider, which then forwards the connection request to participant B. This link, involving two VoIP providers, is called the signaling path. It is necessary in order to locate the

[1] VoIP is not strictly a protocol in itself, it is a suite of different protocols such as *SIP* and *RTP*, implemented on top of IP.

other participant(s) in a conversation, and to set up a connection even when there are restrictive network devices such as firewalls or NAT devices on either participant's network. Once the necessary configuration has been completed over the signaling path, the participants can communicate over a direct, bi-directional path known as the 'media path' (see figure 4.1(a)). In order to provide security (and privacy), the conversation phase (which uses RTP) needs to be secured. RTP is usually secured by wrapping it in DTLS[2], a variant of TLS for use over UDP-based protocols which relies on public key infrastructure (PKI). An alternative solution is to use SRTP, which secures RTP by using symmetric key encryption [Vol]. The key for SRTP needs to be exchanged beforehand, for instance in the signaling path. In this case, the SIP conversation needs to be encrypted as well (usually using DTLS).

Because of the separate signaling and media paths, the needed provisions for security and the architectural requirements, VoIP is a complicated technology that relies on multiple independent standards and is quite difficult to implement. This is the result of a fundamental mismatch between (simple) user goals (such as calling a friend), and the possibilities of the network (sending packets to a pre-specified IP address).



(a) VoIP architecture (b) VoCCN architecture

Figure 4.1: Architectural comparison of VoIP and VoCCN [JSB+09]

4.2 CCN Implementation

VoCCN (Voice Over Content Centric Networking) is an implementation of SIP and RTP over CCN in a way that is secure and compatible

[2] http://crypto.stanford.edu/~nagendra/projects/dtls/

with existing implementations. The design of VoCCN aims to remove most of the complexity of VoIP by allowing the data to flow directly to the interested observer (see figure 4.1(b)). There are some difficulties that need to be overcome to implement the voice protocols on top of CCN:

Service contact point In order for SIP to operate, a way is needed for an Interest to get access to one particular service on one particular host. This *service contact point* would be similar to an IP address and port number in TCP/IP. In other words, it is a way to contact the destination directly, an operation normally avoided in CCN.

On-demand publishing On-demand publishing is the ability to express an Interest in content that has not yet been published, and to route that Interest to a publisher that creates the appropriate content in response to the Interest (similar to dynamically generated web pages on the world wide web). That content is then sent back in response to the Interest.

Bi-directional conversation flow IP packets contain the information of their origin and their destination, which means bi-directional conversation is trivial to implement. In CCN, however, that is not true: one participant sends an Interest, the other participant replies with Data. Because voice conversations are rarely one-way, we want *both participants* to be able to route Interest packets to each other. This requires both on-demand publishing of data, and an algorithm that allows both participants to arrive at the same name for all the content that either one of them publishes, and those names must be routable based on the available data. Also, the names must be unique, or else there is a risk of receiving a Data packet from a previous conversation out of an intermediate cache instead of a Data packet from the current conversation.

In VoCCN, SIP INVITE messages are mapped to CCN Interest packets, in such a way that the entire INVITE is included in the content name of the request. The name also has an appropriate prefix to allow it to be routed towards the callee. For instance, if all messages with a unique name starting with

/example.com/SIP/Bob

will be routed to Bob, Alice could send a CCN Interest packet with a name like

/example.com/SIP/Bob/invite/352356234/example2.org/users/Alice/SIP/352356234/INVITE/...

Such a name would be routed towards Bob, and would allow Bob to route Interest packets back to Alice. The callee unpacks the name and generates a SIP response as a Data packet satisfying the Interest. This data exchange in SIP also includes a seed for a function that generates names to use for RTP packets between the participants within the INVITE message. A more detailed example of the exchange is displayed in figure 4.2.

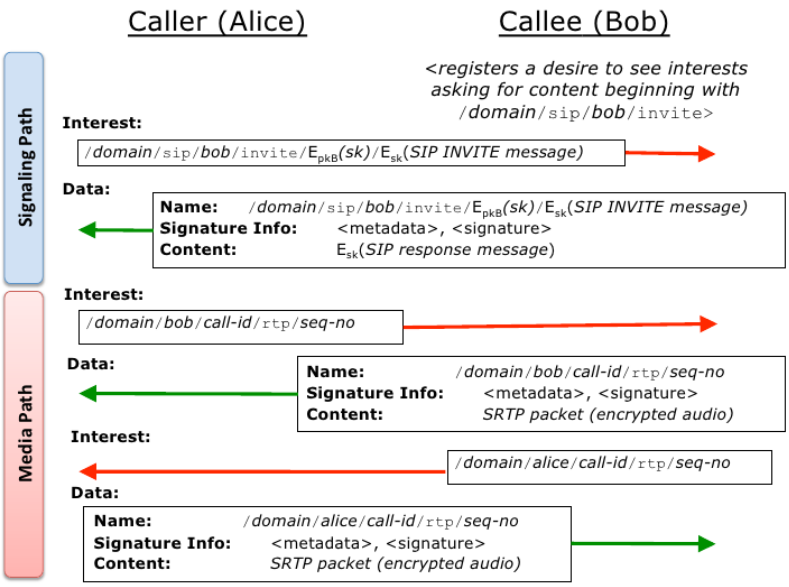


Figure 4.2: An example VoCCN conversation [JSB+09]
$E_{pkB}(sk)$ is a *symmetric key* that has been encrypted with the *public key of Bob*.
$E_{sk}(message)$ is the message encrypted with the *symmetric key*

Each host keeps multiple outstanding Interests, similar to a *window* in TCP or pipelining of requests in HTTP, to achieve better performance. The number of outstanding Interests is typically kept constant.

4.3 Security

In order for people to communicate with comfort, a VoCCN conversation should be secured. As mentioned in section 4.1, VoIP typically implements this by using PKI for the negotiation phase, followed by symmetric encryption for the actual conversation. In CCN, the SIP INVITE is encrypted using PKI, and encapsulated in the CCN Interest name (see figure 4.2). The (encrypted) SIP INVITE also contains a key exchange message using the MIKEY key exchange protocol[3], which is used to establish a key to use with the symmetric encryption of SRTP.

4.4 VoIP interoperability

Despite the different underlying transport architecture, VoCCN remains quite close to VoIP: both use RTP and SIP, and both use similar methods to secure the content. Because of this, it is possible to construct a stateless gateway that maps VoIP into VoCCN (and vice versa) [JSB+09]. The VoCCN-VoIP gateway, which also serves as a SIP Proxy, translates both SIP and SRTP packets between the different transport protocols. When translating a packet from VoIP to VoCCN, the proxy bases the name of the CCN packet on the header of the inbound packet. The gateway caches these packets until an Interest arrives for them.

To illustrate how the interoperability works, consider one participant called Alice, who is using VoIP, and another participant called Bob who is using VoCCN. Alice sends a SIP INVITE to Bob which passes through the VoIP-VoCCN gateway. The gateway uses this INVITE to generate a CCN Interest packet containing the INVITE and sends it on towards Bob. Bob replies using a Data packet, which is translated by the gateway into an UDP packet. Because the IP addresses of the participants are included in the SIP and RTP packets, the gateway does not need to 'remember' who to forward the SIP response to, but can derive the IP address of Alice from the SIP packet. Once Alice gets the SIP response, the involved

[3] http://tools.ietf.org/html/rfc3830

parties can begin to communicate directly using RTP. Alice and Bob both send SRTP packets to the gateway. If an SRTP packet arrives from Alice, the gateway stores it until an Interest arrives for it from Bob. The gateway also generates Interests for Bob's SRTP packets. Bob replies with Data packets, which the gateway forwards towards Alice. Because the VoIP and VoCCN packets can be derived from each other, the gateway itself does not need to keep any state apart from queuing packets until an Interest arrives for them. The key exchange and media path encryption are end-to-end (providing that it is supported by the VoIP client); the gateway only plays a role in setting up the security of the Signaling path. At the VoIP end, this is done using traditional methods between Alice and the gateway. The gateway then signs the message and sends it on into the CCN infrastructure.

4.5 Mobility

One of NDN's supposed advantages is a better support of host mobility. In this discussion we use the term **Mobile Node (MN)** for a host that changes its content router (CR) in the middle of a conversation, and the conversation partner is labeled the **Correspondent Node (CN)**. The standard VoCCN protocol as described in [JSB+09] makes no provisions for the mobility of the hosts. This can cause certain problems in a conversation between two mobile hosts [lLZ+12]:

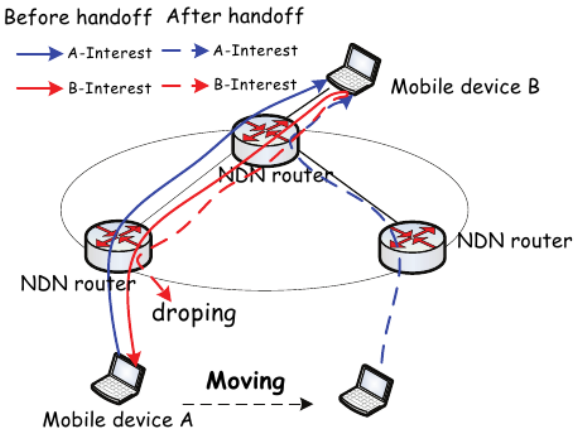


Figure 4.3: Problems with a Mobile Node and CCN [lLZ+12]

Outstanding Data Packets Data packets can still be routed back to the MN, provided that the client re-sends the relevant Interest packets, by following the Pending Interest Table entries set up by the new Interest. However, this causes extra latency, requires retransmission of Interests, and increases resource consumption.

Interests from the CN Interests from the CN will not reach the MN at its new position because they are sent with the wrong routing prefix. In practice, this would mean that the MN will still hear the CN, but not the other way around.

These problems are pictured in figure 4.3.

To alleviate the second problem, it is sufficient that the MN sends an Interest containing a SIP re-INVITE message with the new prefix to the CN. This gives the CN the opportunity to send its Interests to the new prefix. This setup is illustrated in figure 4.4 (compare with 4.2).

Figure 4.4: Dealing with the MN's (Alice's) changing prefix [lLZ+12]

This approach still has disadvantages: it will result in a large handoff delay, which is unacceptable in an application that requires a low delay, such as voice communication. The delay is caused by

the fact that the new name prefix needs to be registered. Also, the Interests sent by the CN between the handoff and the arrival of the re-INVITE message will be lost. To alleviate this problem, a more complex handoff procedure may be adopted: a new name prefix is allocated to the MN *before* handoff happens, based on link-layer indicators such as relative wireless signal strength indicators.

1. The MN detects that a different access point's signal strength is rapidly increasing while the signal strength from the current access point is decreasing.

2. The MN sends a message to the current Access Router (AR)[4] to find the new AR for the new access point, including the new content name prefix.

3. The MN sends a FBU (Forwarding Base Update) message to the old AR to redirect traffic to the new router.

4. The old router contacts the new router to configure the handoff.

5. The new link becomes active: the old router starts to forward new packets from the CN towards the new router, through which they reach the MN.

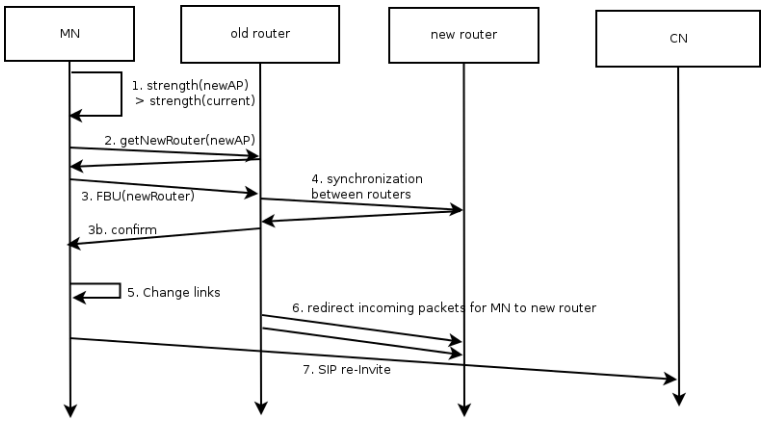
6. The MN sends the re-INVITE message to the CN.

7. The CN can now send its new Interest packets directly to the MN.

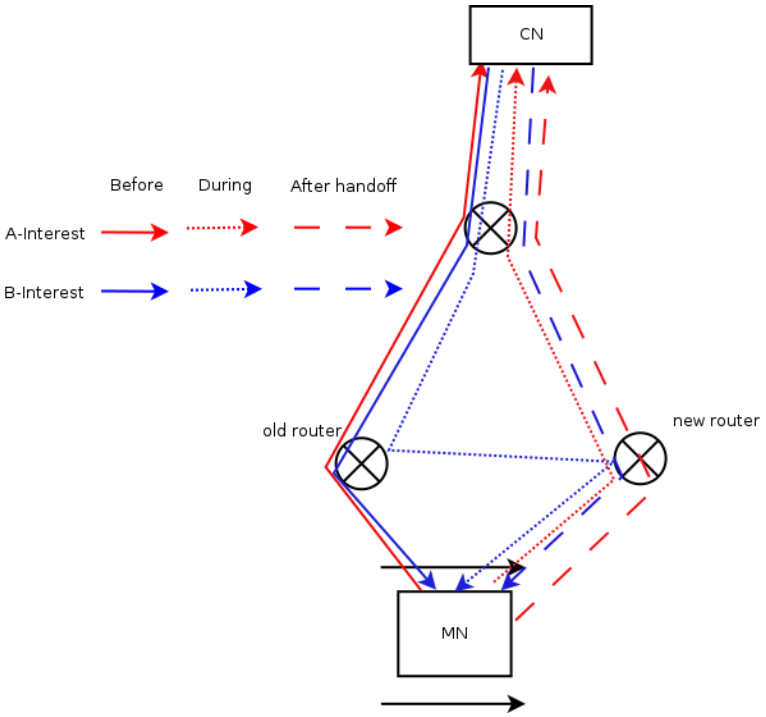
This is illustrated in figure 4.5(a). The routing paths during the different stages are shown in figure 4.5(b). In the latter picture, the 'during handoff' path designates the path taken by Interests between the moment the MN switches links and the moment the CN receives the SIP re-INVITE message.

This approach was tested in several scenarios and proved to have a much lower delay and retransmission of Interests than the simpler method [lLZ+12].

[4]Access Router: First-hop Content Router.



(a) The transition process



(b) Interest routing before, during, and after Access Point switching

Figure 4.5: Seamless mobility provisions in VoCCN

4.6 Evaluation

The VoCCN architecture, as described above, has been implemented by Jacobson et al. [JSB+09]. The implementation works perfectly,

and performance is said to be similar to the UDP-based protocols. A small fraction (less than 0.1%) of the packets were dropped by the implementation for arriving too late. The VoIP-VoCCN gateway was not implemented. Another team has implemented and tested VoCCN in situations of varying mobility, with generally good results [lLZ+12].

VoCCN has some advantages over VoIP [JSB+09, lLZ+12]:

- Content routing supports multi-point routing, so the request can be easily routed to all the places it might be answered (cell phone, home phone, home computer, ...), without requiring the complex infrastructure that is necessary to support it over IP.
- VoCCN is architecturally simpler and more scalable than VoIP because it does not require any SIP proxies.
- Because of its architectural simplicity and content-oriented focus, it is relatively easy to add advanced services such as voicemail, call logging and recording, and conference calling.
- With the changes discussed in section 4.5, VoCCN can be used in situations where the user is highly mobile (for instance, on a train).

However, the fact that Interests need to be generated for each piece of RTP Data, even if no data is actually available, does increase the overhead of the protocol on the network. This can be alleviated using Publish/Subscribe systems for real-time protocols like these (see section 5.5).

Chapter 5

Alternative NDN technologies

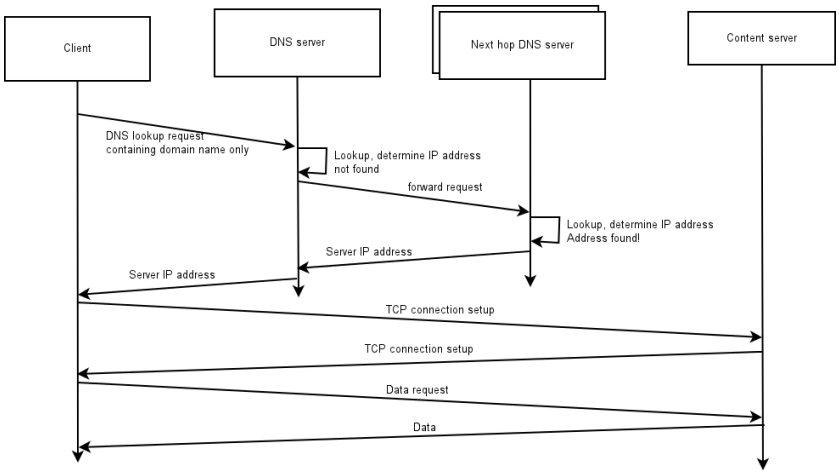
5.1 TRIAD

TRIAD, an acronym for "Translating Relaying internet Architecture integrating Active Directories"[1], is a proposed new internet architecture that works on top of IP [CG00b]. It uses "Content Routers" (CRs) to route a request for content by name as part of a 'Content Layer' that sits between the Network and Transport layer. The Content Routers provide DNS name request forwarding, and distribute information regarding name reachability through name-based routing in order to provide network-integrated support for content routing, caching, content transformation and load balancing.and Independence between address realms.

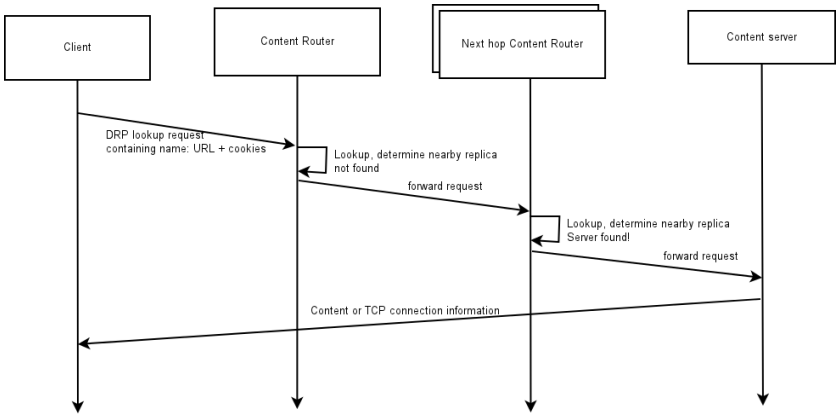
TRIAD was born from the observation that in order to connect to a website, a completely separate, and sometimes expensive, DNS lookup phase to find the IP address of a server must be performed first. This step can easily be the dominant delay in the delivery of content to a user. Where in the IP and DNS model, server lookup,

[1]The authors also joke that it may also stand for *Time to Rescue the internet from Address Depletion*[CG00a]

server connection and content retrieval are largely separate, in TRIAD they are integrated using a protocol called DRP, or 'Directory Relay Protocol'. A DRP request is similar to a DNS request, but instead of a domain name it incorporates the entire URL of the desired content, along additional information (referred to as 'cookies'). In addition, DRP contains TCP connection information such as a sequence number, port information and TCP options, allowing it to function as a TCP connection setup protocol. Because of this, the two steps of server location and connection establishment are merged into a single client-side round-trip (see the comparison in figure 5.1).



(a) DNS/TCP data location and retrieval



(b) TRIAD data location and retrieval

Figure 5.1: A comparison of DNS/TCP (top) and TRIAD (bottom) methods for content location and retrieval

The end-to-end identification of a host interface[2] or a multicast

[2]The notation *host interface* is used, because a host can have multiple interfaces

interface is a hierarchical DNS name. That name is used for all identification and authentication of hosts. No other globally significant addresses are necessary. TRIAD therefore does not rely on IP addresses being globally unique. Relaying of packets is done by a new protocol called WRAP, or 'Wide area Relay Addressing Protocol', a path-based addressing protocol that exists as a shim between IP and transport protocols. It carries a pair of *internet relay Tokens (IRTs)*, the *forward token* and the *reverse token*, to extend addressing beyond IP. These tokens are path labels of WRAP, identifiers of network interfaces only routable within one specific network (see section 2.2 for a description of path-based addressing).

In this architecture, replicated content is supported through hierarchical transparent caching of not just names, but also content itself, which minimizes the damage from an explosion of demand in popular content without requiring any proxy configuration or ad-hoc measures. Caching is done in a CR by forwarding DRP packets from clients with the CR's own source address, allowing the caching CR to store the content in its cache before forwarding it to the client.

TRIAD also inherently supports single-source multicast content distribution. A client wishing to join a multicast session sends to name lookup to the source of the session, which returns the information necessary to join the session, with the necessary state being established along the path.

Dynamic routing on content names in TRIAD is done by routing on name suffixes. The protocol to do this is called the 'Name-Based Routing Protocol' (NBRP). Content servers advertise the content names with an associated path of content routers. This is accompanied with rough load information to allow routing based on the current load of intermediate nodes. Content routers distribute routing updates to each other. This allows the routing to evolve with the network topology, and name resolution and packet forwarding are 'fate-sharing' operations: if one works, the other will as well. In order to make name resolution scale for millions of (not necessarily hierarchical) domain names, routing paths are aggregated to distribute updates to many different names as a single update. Most names map to a small number of aggregates, and names in an aggregate have an equivalent topological location. CRs learn about ag-

which each have a different address, like in IP

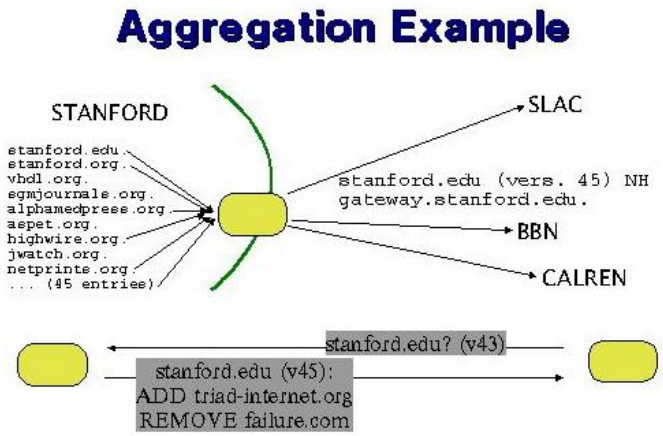


Figure 5.2: An example of domain name aggregation for content routing [Gri]

gregate memberships by sending queries towards their creator (see figure 5.2). Experimental results suggests that this system scales at least as good as BGP [CG00b].

TRIAD can be easily, and incrementally, deployed into the current internet architecture because of its backwards compatibility.

5.2 DONA

DONA, short for "Data Oriented Network Architecture", is a content-centric architecture explicitly designed to address three problems from the IP architecture: persistence, availability and authenticity (see 1.1.2). DONA's authors argue that the main step that is necessary to resolve these issues is to change the way internet names are structured and resolved [KCC+07]. To do this, DNS names are replaced with flat, self-certifying names and DNS name resolution is replaced with a name-based anycast primitive on top of IP. As such, DONA is essentially a shim between the network (IP) and transport (TCP/UDP) layer. It enforces a strict separation of naming and name resolution, where the naming provides persistence and availability, and name resolution guarantees authenticity.

5.2.1 Naming

Unlike IP, DONA chooses to route based on flat, self-certifying names. This means names are invariant, and provide for easy authentication. The downside to such names is that they are hard for users to recognize; as such, it is expected that users use their own names, which are mapped onto DONA names using some third-party mapping system (similar to how the DNS system provides a mapping of user-friendly domain names to IP addresses). One such naming concept for user-understandable names is proposed in [FSLL+06].

DONA introduces a new naming scheme, based around the actors, called *principals*, that use the network. Each principal has his or her own public and private key. Content created by a principal is given the name **P:L**, where **P** stands for the hash of the public key of the principal, and **L** is a label the principal assigns to the content, which is unique for that principal.

Each datum comes with metadata, which includes a principal's public key and the principal's *signature* of the data. A signature is a hash of the data, encrypted with the principal's private key (see figure 2.1 on page 12). The signature of the data is used to verify the authenticity of the data by decrypting the signature with the principal's public key, and comparing it against the hash of the received data. If the decrypted signature matches the hash of the data, it means the data has not been altered while transiting the network. In this way, the architecture of DONA inherently meets the authenticity requirement.

An alternative naming method is used for the special case of *immutable* data: data that cannot be altered once published. In this case, the second part of the name, **L**, is a hash of the contents of the data. This effectively ensures its integrity: if the data changed, the name would change as well. In this way, the client does not need to rely on the principal to ensure the authenticity of the data. For immutable data, the principal is simply a distributor of the data, not the owner.

There is no connection between a principal and the hosts that store its data: a principal **p** authorizes an entity to serve his data

(see next section). This authorization has an expiration date (TTL). If the owner moves his hosting, he or she simply authorizes a new host to host the content and lets the old host's authorization expire. The name of the data does not change in this process, so the name will always identify the proper content, which means there will be no more 'broken links'.

5.2.2 Name resolution

To request data by name, one of two systems is used:

Lookup-by-name This method is used by DNS, it returns the location of a nearby copy by looking it up in a distributed database.

Route-by-name In this method, the routing protocols itself are designed to find the shortest path to content and route around failures. This method is used by both TRIAD (see 5.1) and DONA.

DONA introduces a new class of network entities called Resolution Handlers (RHs). There are two kinds of operations supported by these RHs: FIND(P:L) and REGISTER(P:L). When a client issues a FIND(P:L) command, the RH routes the request to the nearest copy. The REGISTER(P:L) command is used to set up the state to allow RHs to route requests for this datum. Each domain or administrative entity has one logical RH. A Resolution Handler of X RH_x is the provider/customer/peer of RH_y if and only if X is the provider/customer/peer of Y in the context of AS-level relationships[3]. An RH uses the local policy, which is consistent with the domain's peering agreements of BGP (see [BGP]), when processing FINDs and REGISTERs.

A client knows the location of its local RH through local configuration, possibly using some configuration protocol such as DHCP. A machine authorized to serve content with the name P:L sends a REGISTER(P:L) message to its local RH. Registrations can also look like REGISTER(P:*) if the host is serving all of the data associated

[3]AS: Autonomous System

with this principal, or if it will forward all the incoming find packets to a local copy. The RH maintains a registration table that maps a content name to the next hop, along with the distance to the copy (using some custom metric, such as the number of hops). There are separate entries of the type P:*. In response to a FIND(P:L), a RH will use longest-prefix matching: it first searches for a match of the type P:L; if none is found, it searches for a P:* match. If the RH does not have any match for the content, the FIND will be forwarded to its parent. For immutable data, requests can also look like FIND(*:L).

If a REGISTER(P:L) message is received from a client, the message is not forwarded unless the RH does not yet have any entry for P:L or the REGISTER comes from a copy closer to the RH than the previously registered one. If either of these situations is the case, the RH will update its registration table and forward the message to its parent and peers, so that they can update their own registration table, if necessary (see figure 5.3). If a REGISTER is received from a peer, it will be forwarded or not based on the local policy. In order to prevent abuse, REGISTER messages need to be authenticated. To do this, the RH issues a challenge with a nonce. The client then signs the challenge with the private key of the principal, so that the RH knows that the host is authorized to serve the content (by decrypting it with the principal's public key). However, this method requires a host to know the private key of the principal, which is usually not desirable. Therefore, it is also possible to sign it with another key, and include a certificate from the principal authorizing the other key to sign this piece of data. When forwarding a REGISTER, a RH will sign it itself, so the receiving RH knows that it came from a trusted RH. The signatures of all the RHs along the path are included with the message, along with the cost from each RH to its previous hop, before sending it on. REGISTER messages have an expiration date (TTL), and must be periodically refreshed. There is also an UNREGISTER command, for when a host wishes to stop serving a particular piece of content.

A FIND(P:L) message is a simple shim between the network and transport layers. They also serve to initiate a transport exchange, and ensure that a packet reaches the appropriate destination. If a designation does not exist, the FIND will reach a tier-1 AS and fail to find a record. In that case, an error is returned to the source of the FIND message.

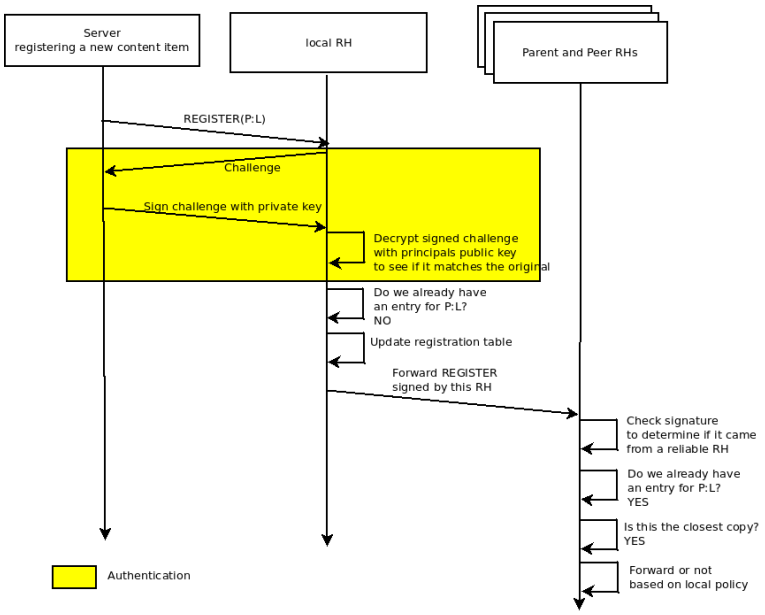


Figure 5.3: An example of content registration in DONA

The only change needed to existing protocols to support these messages is that transport protocols need to bind to names instead of addresses. It can also significantly reduce the complexity of application-layer protocols: for instance in HTTP, the most important parts are the URL and the headers. The URL can be replaced by the DONA name because the data is named in a lower layer, and the headers can sometimes be included in the name (such as a separate name for each language version, which removes the need for a 'Language:' header).

5.2.3 Path-based addressing

DONA can use a different form of internet addressing instead of relying on IP. It removes pressure on lower-level addressing structures by providing a mechanism for path discovery separate from IP and enables IP to use *path labels* rather than globally routable addresses (see section 2.2). Path labels are a chain of domain-specific addresses that do not have any meaning outside of that particular domain. A client sends a FIND to his RH using his domain-specific address as the source address. The RH then adds its own address on the next-hop network to this address before sending the mes-

sage on. At each hop, the next-hop address is appended to the source address. Then, when replying, the reply is sent back over the same path, following addresses back down the chain, 'peeling' off one layer of addresses at each RH. These per-hop addresses can be quite short, because it only needs to select between a few possible next-hop domains (for instance, if an RH has only 4 distinct interfaces, the per-hop address could be just 2 bits). The inter-domain routing tables in this approach are quite short, just enough to forward per-hop instructions to the next-hop AS. There is no longer any globally meaningful address; DONA FIND/REGISTER messages are necessary for establishing end-to-end connectivity. Endpoints are responsible for detecting AS-level path failures and need to resend the FIND on failure.

5.2.4 Security

Several attacks are possible against DONA, some of which can be made difficult or impossible by the architecture itself, and others which need external mechanisms to be resolved.

Denial-of-Service DONA does not provide any specific mechanisms to counter DoS attacks, but leaves it to IP-level mechanisms to restrain unwanted packet streams that overwhelm a RH. Also, if path-based addressing is used, DoS attacks will be much harder to successfully execute (see section 2.2).

Resource exhaustion attack Resource exhaustion attacks can be prevented by having providers place contractual limits on customers to restrict the amount of FINDs and REGISTERs they can execute over a given period.

Malicious RH A malicious RH could refuse to forward FINDs and REGISTERs, which is a failure of the AS. It could also forward REGISTER messages overheard from other RHs. This can be stopped by

including the hash of the public key of the next hop with a REGISTER message, after which any RH receiving a REGISTER with a faulty hash can simply ignore that message. Another way a malicious RH can do damage is by simply refusing its clients service. There is a proposed extension to allow clients to request access to other copies of data than the closest one, which would mitigate this risk, unless the RH lies on the path to all the copies of the content. In all of these cases though commercial pressures should ensure that any problems get fixed quickly. RHs are commercially related to the clients they serve (nobody offers a service, like internet routing, for free), and a client can presumably switch to a different RH if the current one does not perform as expected.

Key compromise As in CCN, key compromise is perhaps the biggest risk because the design is very key-centric. There is no remedy for this in DONA, but third party key revocation lists could be used, in combination with key status query protocols.

5.2.5 Applications

DONA can be employed to significantly facilitate existing problems on the internet:

Server selection Each server authorized to serve P:L sends a REGISTER message to its local RH. DONA will route any request for this content to the closest server. These different servers could all be part of a content delivery network, or part of a P2P infrastructure.

Mobility A roaming host unregisters itself from one location and registers itself again at the next location. Subsequent FIND requests will then be routed to the new location once the new registrations have taken effect. If there is a sudden loss of connectivity, there may be a wait until the previous register expires.

Multihoming A host can register with each local RH; a multi-homed domain forwards REGISTER messages to each provider. FIND requests can then use multiple paths.

Session Initiation Session management protocols such as SIP are highly compatible with DONA, with SIP INVITE messages mapping perfectly onto DONA FIND messages, and SIP REGISTER messages mapping effortlessly onto DONA REGISTER messages.

Caching A node can support this by changing the source IP of a FIND packet to its own IP, so that the return data goes through this node (if path-based addressing is used, this is always possible without changing the packet, because the reply takes the same route as the request). The node can then install this data into its cache before forwarding it, with some suitable timeout. For each FIND request, the cache is examined; if a cache hit is found, it can respond directly to the source IP. If the alternative addressing method is used (see 5.2.3), caching is trivial because all the return data travels over the same route.

5.3 Content-Oriented Transport Protocol

The concept of Content Oriented Transport Protocol (COL4) [ZN11] starts from the observation that a clean-slate architecture like CCN is not compatible with the current Internet and is therefore unaffordable and undesirable. COL4 therefore proposes to maintain the IP layer to replace the transport layer (TCP) with a new header. This new transport protocol is referring to the content name. By doing this, basic packet inspection could already identify the content of the packet while routers that do not support COL4 can fall back to IP forwarding. The essence of COL4 is that by having the content identified at transport layer, routers can make intelligent decisions and advise better locations for content retrieval than the original packet destination, without having to rely on DPI (Deep Packet Inspection). At the same time, care is taken that the new protocol

can co-exist with TCP and that the retransmissions and congestion avoidance characteristics of TCP are preserved in the new protocol.

5.3.1 Naming

In COL4, content is identified by its *publisher* , its *name* and *version*. The publisher is a globally unique name that needs to be registered, similar to a domain name in the internet. The name can be allocated by the publisher, although the paper also suggest unique names that represent the same content that is available from different publishers. Content requests are allowed to be ambiguous, by omitting the publisher to get the named content from any publisher. Also the version number can be ambiguous to get any version newer than the specified one. COL4 also supports the partial transfer of content, which are called *partitions*. A partition is a minimum unit of content. This is considered as being interesting for video delivery in chunks, although COL4 puts limits on the number of partitions that content can have and requires that this number is known upfront, making it hard to apply for live streaming.

5.3.2 Architecture

COL4 argues that the current sockets based on IP address and TCP/UDP ports are too much a constraint for realizing Content Centric Networking and proposes a new transport layer protocol over IP, based on following principles: remain

- Compatible and be able to be deployed incrementally.
- Abandon socket addresses, replacing those with content naming and connectionless
- As being a transport protocol over IP, it has the have the characteristics of TCP with respect to congestion control.

As other Content Centric Networking approaches, in-network handling of content is essential. Intermediate devices (routers) should be made aware of the content they are transporting, and this without relying on high computational techniques like DPI. COL4 does this by placing the content reference in a new header that is on top of IP, replacing as it where TCP or UDP. See figure 5.4

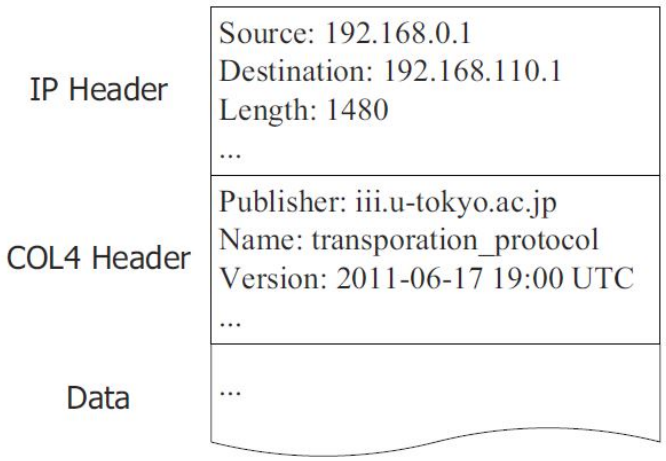


Figure 5.4: COL4 Header. Source: [ZN11]

This allows routers that are not COL4 aware can simply ignore the new headers and forward packets based on IP address. The COL4 header is used for four primitives: *availability announcement, content request, content transmission* and *transfer control*.

- A content host sends availability announcement to neighboring nodes in the network. These nodes can decide to forward the announcements or take part in the content caching. To ensure that no announcement loops are created, an incremental weight is added to the announcement message so that routers who have a cached copy will not forward the announcement.
- Content request can be specific or ambiguous, complete or partial (asking for a partition).
- Content transmission is done in unit of partitions which may require several packets. The header is identifying the content and can be more than 500 bytes. To reduce the overhead only the first message will have this complete header including a 16 bit hash, while consequent messages will only carry this same hash.

- COL4 adopts TCP - like congestion control and uses Explicit Congestion Notification (ECN)

5.4 Serval

Serval [NSG+12] goes one step further in transforming the traditional host centric internet. Instead of focusing on content, Serval is a Service Centric Networking technology. Similar to CCN, it has the ambition to replace the TCP/IP networking stack with something more suited for today's needs. Serval however starts from the observation that the internet is increasingly used to access services and not necessarily content only. As such Serval might be seen as a generalization of CCN, but in fact it is quite different as services are still being offered by hosts (servers or sensors), while content can be cached in intermediate systems like the CCN Content Store. In this respect Serval is less disruptive, also because the IP networking layered is preserved. Serval includes a new Service Access Layer (SAL) that sits on top of IP. So similar to DONA, a new addressing layer is added between IP and TCP.

5.4.1 Naming

The SAL will operate on serviceIDs, which represent a hierarchical naming of service. Similar to CCN, serviceIDs can be aggregated by prefix for scalability. Service endpoints only refer to serviceIDs. The applications are therefore no longer aware of IP and TCP/UDP port. This introduces the possibility for multihoming and dynamicity, important for modern online services that are increasingly accessed on mobile devices with changing connectivity. Similarly, service hosting is done on cloud servers that are located on different (virtual) hosts that can change over time depending on network conditions and sever load situations.

Within the concept of Serval, ServiceID naming is left open, as it is the intention to have forwarding rules provisioned in the routers by a service controller. Serval does not dictate how ServiceID are

learned. ServiceID are like URI that are exchanged between applications. High level service descriptions can be resolved into ServiceID in many ways, like directory services, search engines or social media. The Serval prototype uses a 256-bit serviceID namespace. A large namespace creates the possibility that a central naming authority like IANA could allocate prefixes to organizations. The serviceID prefix can thus be used to identify the authorized provider of the service. The prefix would be followed by bits that provide a hierarchy within the organization for service resolution. The serviceID would end with a hash of the service provider public key, so that the client can verify that it is dealing with the authorized instance of the service.

5.4.2 Architecture

Serval is using the ServiceID in intermediate nodes only for service selection. The first packet of a connection is forwarded to service routers which will then decide which service endpoint will service the request. This is called "'late binding"' as it defers the selection on the service instance to the part of the network that has the details and is up- to-date on the status. So this is a sort of in-channel resolution which can have many advantages, for instance is avoids complex structures with server farms, proxies and load balancers. Applications traditionally cache addresses, while Serval re-resolves service names,leading to fast failover and intrinsic load balancing. Serval features serviceID to identify the service and also flowID that identifies each flow within a socket. The flowID takes over the role that the TCP/UDP port traditionally has for identifying connections. TCP is maintained as transport layer in Serval, but only for reliable data delivery including retransmission and congestion Like other technologies discussed in this paper, this represents a rethinking of the network stack. Serval goes further in this by making clear separations between naming (serviceID), multiplexing (flowID), endpoint identity (IP) and reliable delivery (TCP). (see figure 5.5). control. Once the the service instance has been discovered with the first packet, the remaining packets between the two endpoints travel via IP network routers.

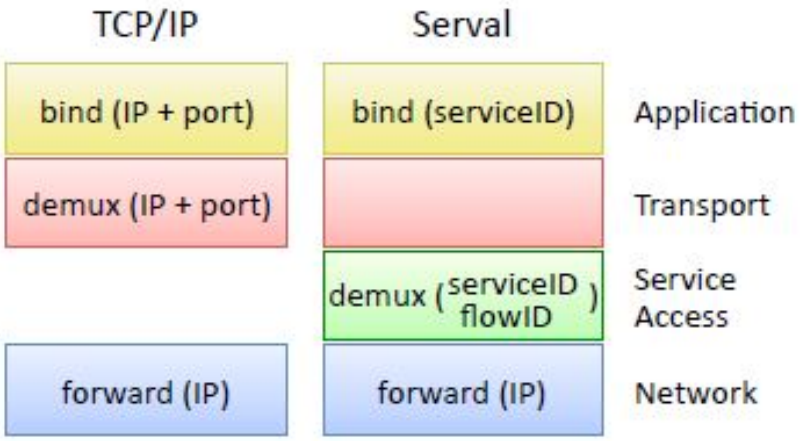


Figure 5.5: Serval identifiers. Source: [NSG+12]

5.4.3 Service name resolution

The service routing stack (figure 5.6) does a longest prefix match on the the serviceID for resolution. It decides to either FORWARD, DEMUX, DELAY or the first packet. In case of FORWARD, the packet is given an new IP destination address of the next SAL capable hop, typically several destinations. In case of DELAY the packet is queued while a service controller figures out how to handle it. This allows for extensible service discovery, as Serval features a split between user space control and data plane Service Table (SIB). The service controller controls the SIB and installs new resolution rules, leading to potentially supporting a wide scenario range. Rules can be installed "'on-demand"'. DEMUX is used to deliver the packet to the local socket, when there is an application listening for it, typically at service endpoints. It is the intention to establish multiple flows, each with their own flowID per service session, in order to allow uninterrupted service over multiple paths. The flow table demuxes packets of established flows, directly to the transport layer. Interestingly, the SAL features a default forwarding rule, which matches any serviceID and which is pointing to the broadcast address interface of a host. This enables service communication on the local network segment of for identifying the local service router.

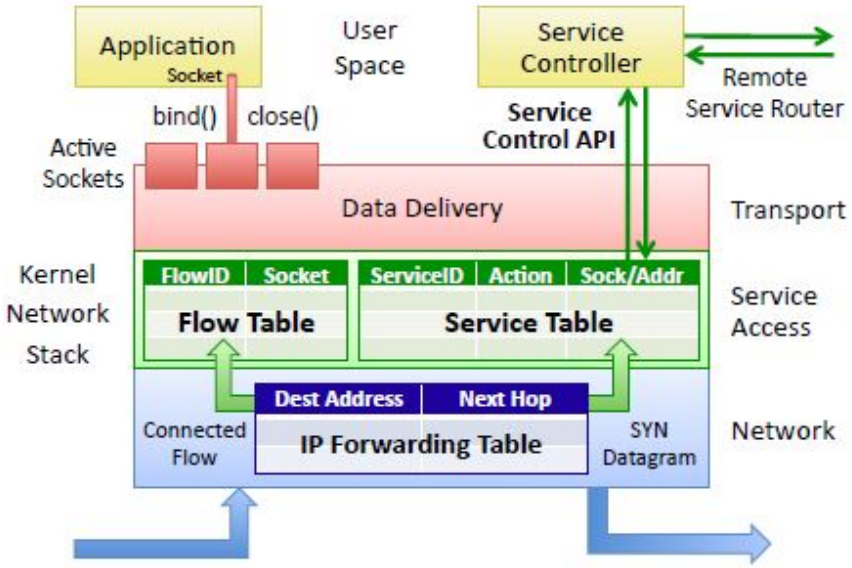


Figure 5.6: Serval stack. Source: [NSG+12]

5.4.4 Application support and adoption

Making applications work with Serval requires in first instance adding support for a new Serval socket. When the paper was published, several were demonstrated, including the Firefox web browser. There is little evidence that Serval has enjoyed any further major successes in recent years. The most recent work focuses on mobile apps that work seamless over several networks, like Wi-Fi and 4G.

5.5 Publish/subscribe systems

There is much interest in the CCN world for ways to make CCN more efficient for real-time applications like VoCCN (see previous section). Previously, such protocols require continuous sending of new Interests to sustain the flow of data. While several different solutions have been proposed (see, for instance, [YFYX12] for a recent example), Publish/Subscribe systems (and their integration into CCN) are in particular gaining attention.

5.5.1 Principles

Publish/subscribe systems are messaging systems designed to support full decoupling in *time*, *space* and *synchronization* between producers (*publishers*) and consumers (*subscribers*) of content on the internet [EFGK03]:

Time coupling Time coupling means that the publisher and subscriber need to be involved in the action *at the same time*, in other words, they both need to be on-line at the same time for the action to succeed.

Space coupling Space coupling means that communicating parties need to know each other's address. This means that the client will only accept replies coming from one specific address.

Synchronization coupling Synchronization coupling means that a publisher can only transmit the information when a subscriber explicitly asks for it.

In IP networking, the publisher is explicitly addressed (space coupling), the publisher needs to be on-line for any communication to take place (time coupling), and data is sent as a response to a query for that data (synchronization coupling). In CCN, the host that sends the data does not matter (no space coupling[4]) and the originator of the data doesn't need to be on-line for the data to reach an interested party (no time coupling[4]).

Coupling	**IP**	**CCN**	**publish/subscribe**
Time	yes	no[4]	no
Space	yes	no[4]	no
Synchronization	yes	yes	no

Table 5.1: A comparison of the different kinds of coupling in publish/subscribe, CCN and IP

However, CCN does have synchronization coupling, as hosts always need to explicitly ask for data in order to receive it. Publish/-subscribe systems lift this requirement (see table 5.1).

[4]This depends on whether caching is used effectively. There is time and space coupling in VoCCN, for instance

Because of the lack of coupling, publish/subscribe systems provide multiple advantages:

- Subscribers get their content as soon as it becomes available.
- Publishers don't need to be online at the same time as their subscribers.
- There are no scalability or traffic issues for publishers.

Operation

There are three distinct entities in publish/subscribe systems: *publishers*, who push their content to the network, *subscribers*, who register their interest in some content by subscribing to it, and an *event system* (see figure 5.7). When a publisher has new content, that content is sent to the event system, and the event system is responsible for notifying all subscribers who registered an interest in content of that type [EFGK03].

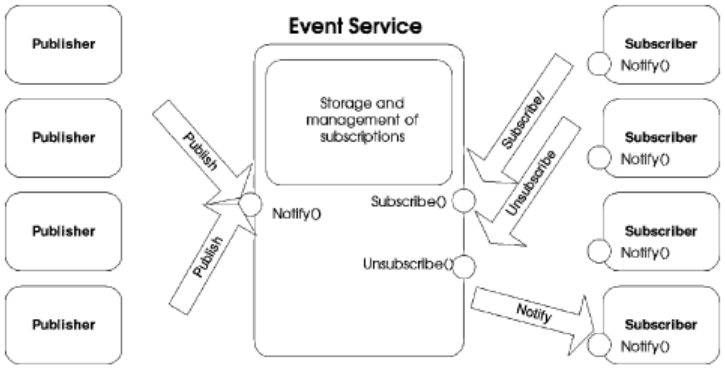


Figure 5.7: How publish/subscribe systems work [EFGK03]

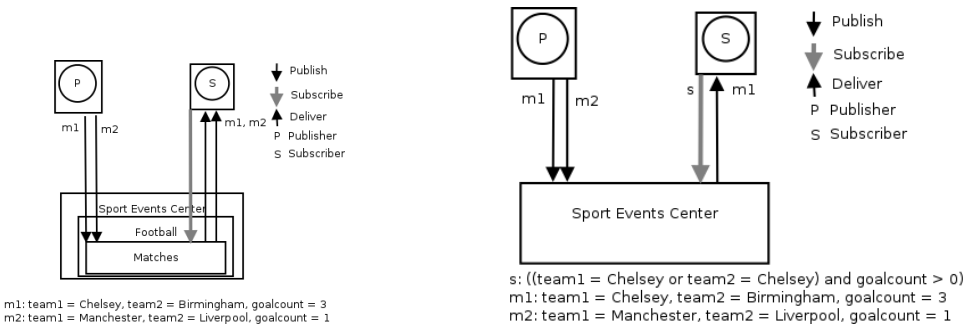
There are three methods for selecting the content for subscribers to subscribe to [EFGK03]:

Topic-based Topic-based publish/subscribe systems group events by topic, usually in a hierarchical fashion (e.g. /sports, /sports/football, /sports/football/FIFA, ...). This is similar to

USENET newsgroups (comp, comp.os, comp.os.linux, ...). Topic names can contain wildcards.

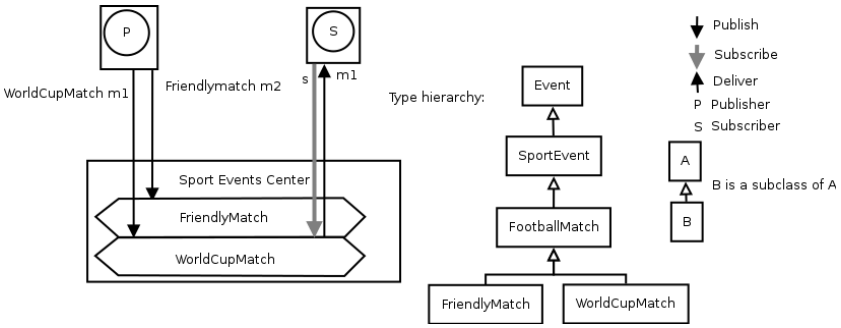
Content-based Content-based systems focus more on the actual content of the events, rather than the category. The subscriber gives criteria ((team1 = 'Chelsey' or team2 = 'Chelsey') and goalcount > 0). Some systems also allow the subscriber to specify a combination of events which need to occur before being notified.

Type-based Type-based systems are similar to the topic-based systems except that each topic is represented by a specific event type registered with the event system (its properties are known to the event system). For instance, a friendly football match and a FIFA world cup match would be represented by different event types (both derived from the same subclass, Match). There is also filtering similar to content-based systems, so subscribers can subscribe to Matches (of any kind) featuring Chelsey as one of the teams, or subscribe to all FIFA world cup matches.



(a) Topic-based classification

(b) Content-based classification



(c) Type-based classification

Figure 5.8: Content classification in publish/subscribe

The difference between these are illustrated in figure 5.8.

Desirable features

A complete publish/subscribe system should ideally possess each of these features [CAJ+11]:

Push-enabled dissemination The system has to push information to online subscribers interested in it. This has the added advantage that subscribers will be notified of the event at the earliest possible time, which would be useful whenever quick updates are required, such as news dissemination, stock market quotes and emergency notifications (such as tsunami warnings).

Decouple publishers and subscribers In a publish/subscribe system, it is important for the network to be content centric in order to decouple publishers from their subscribers, while still being able to route updates to subscribers.

Scalability Publish/subscribe systems must be able to deal with a large amount of publishers and subscribers, and should ideally scale with the number of publishers and subscribers.

Efficiency The system should use network and server resources efficiently in order to minimize the overhead on end-points.

Incremental deployment A publish/subscribe system will likely only be rolled out gradually, so it should be incrementally deployable and ideally provide a seamless transition from an IP-dominated environment.

Support for hierarchies and context in naming content By being able to exploit context as well as hierarchy in names, content can be described in a more natural and accurate way. A single hierarchy cannot support all possible ways users might want to query for information; for instance, if content names are arranged by topic, for instance /international/business/mid-east and /international/politics/mid-east, someone who is interested in all new information about the middle east should not have to subscribe to both content descriptors, but can just subscribe to /international/*/mid-east.

Support a two step information dissemination In a two-step publish/subscribe system, the publisher originally publishes just a snippet of the information available to him into the network, along with a description of the method of obtaining more information if the subscriber wants it. This avoids unnecessary transmission of content the subscriber is not really interested in, and allows the publisher to control access to (and potentially charge for) information they publish (while still making effective use of the publish/subscribe network). This idea is similar in principle to newsfeed protocols like RSS.

Support for offline subscribers If a subscriber is offline when a particular piece of content he has subscribed to us published, the subscriber should still receive the data that they have missed. A related issue is that new subscribers should be able to retrieve previously published content.

5.5.2 COPSS: Content-Oriented Publish/Subscribe Systems

COPSS is a Publish/Subscribe system implemented over CCN. It uses CCN's Content Name as a basis for a Publish/Subscribe system that can use name- and topic-based identification of content along with more fine-grained contextual and hierarchical specification of information. It adds a push-based multicast capability to CCN's pull-based information delivery, along with support for offline subscribers and a 2-step delivery model that provides access control to publishers.

There are several problems with implementing publish/subscribe over CCN [CAJ+11]:

Multicast CCN automatically caches content in the network, in the Content Store of intermediate Content Routers (see 3.3). However, this only occurs if one or more intermediate routers have lots of downstream subscribers who regularly send Interests. If not, every Interest will still be routed back to the publisher. But event with caching in the network, CCN (and NDN in general) are still push-based networks: anyone interested in data

still has to actively ask for it.

Dealing with multiple publishers In CCN, if a subscriber wants data, he needs to know the exact name of the data he wants, or he will only receive a portion of it (see figure 5.9). The subscriber either needs to specify *exactly* what data he is interested in, or find a way to explicitly exclude previously received content from its Interests, and then repeatedly send Interests with an exclude field until no new data is received. Obviously, this is not a satisfactory solution. For a publish/subscribe system, the subscriber typically wants all desired content to be retrieved simultaneously, without needing to repeatedly resend Interests.

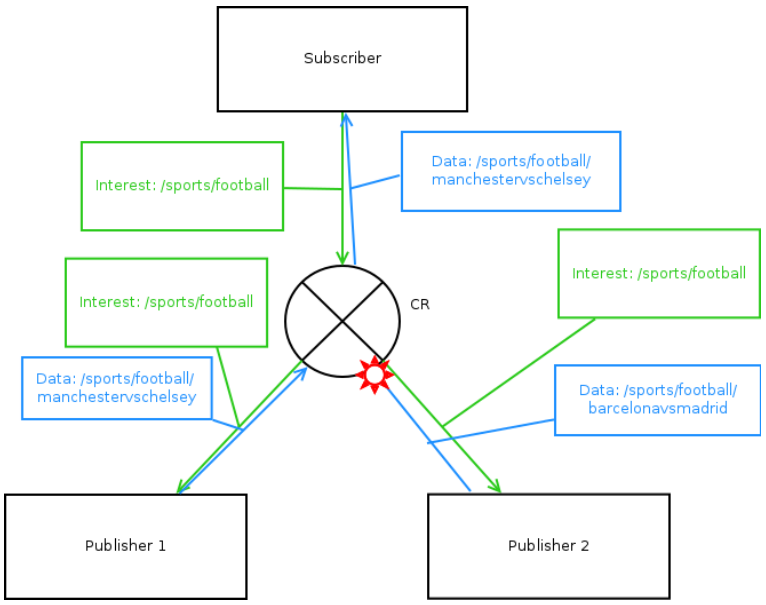


Figure 5.9: Illustrating a problem with CCN for publish/subscribe systems: when a Content Router encounters one response to the Interest /sports/football, in this case /sports/football/manchestervschelsey, it considers the Interest fulfilled and will discard extra Data replies

COPSS is a system built on top of CCN (and backwards compatible with it) that adds push-based networking and an effective publish/subscribe functionality. In order to accomplish this, Content Routers are updated to maintain a subscription table (see figure 5.10). A Subscription Table is a list of faces with downstream subscribers and the content they've subscribed to. In addition to the existing packet types **Interest** and **Data**, COPSS also adds **Subscribe** and **Publish** packet types. Subscribe packets are sent by a Subscriber to indicate their interest in receiving updates when pub-

lishers publish new data matching the supplied Content Descriptor. Publish packets are sent by publishers when they have new data. Finally, a COPSS network also contains Rendezvous Nodes (RNs), who maintain subscriptions. Each RN has a list of CDs for which they are responsible. When a publisher publishes some data, they send a Publish packet containing the Content Name /rendezvous/ followed by the Content Descriptor. The packet is forwarded towards the RN responsible for that content descriptor. The RN will then strip the /rendezvous/ prefix and forward it towards all the subscribers for that CD. Publishers don't expect a reply when publishing content, so Publish packets are not put into the Pending Interest Table.

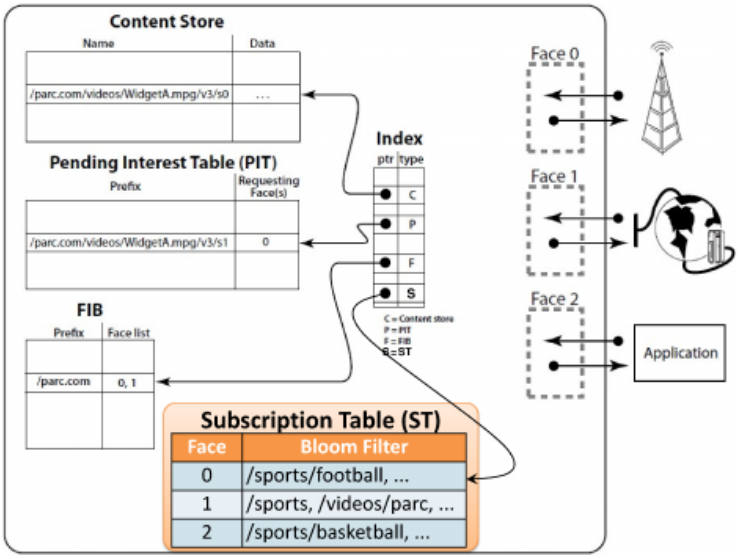


Figure 5.10: CCN's forwarding engine extended with a Subscription Table [CAJ+11] (compare with figure 3.4)

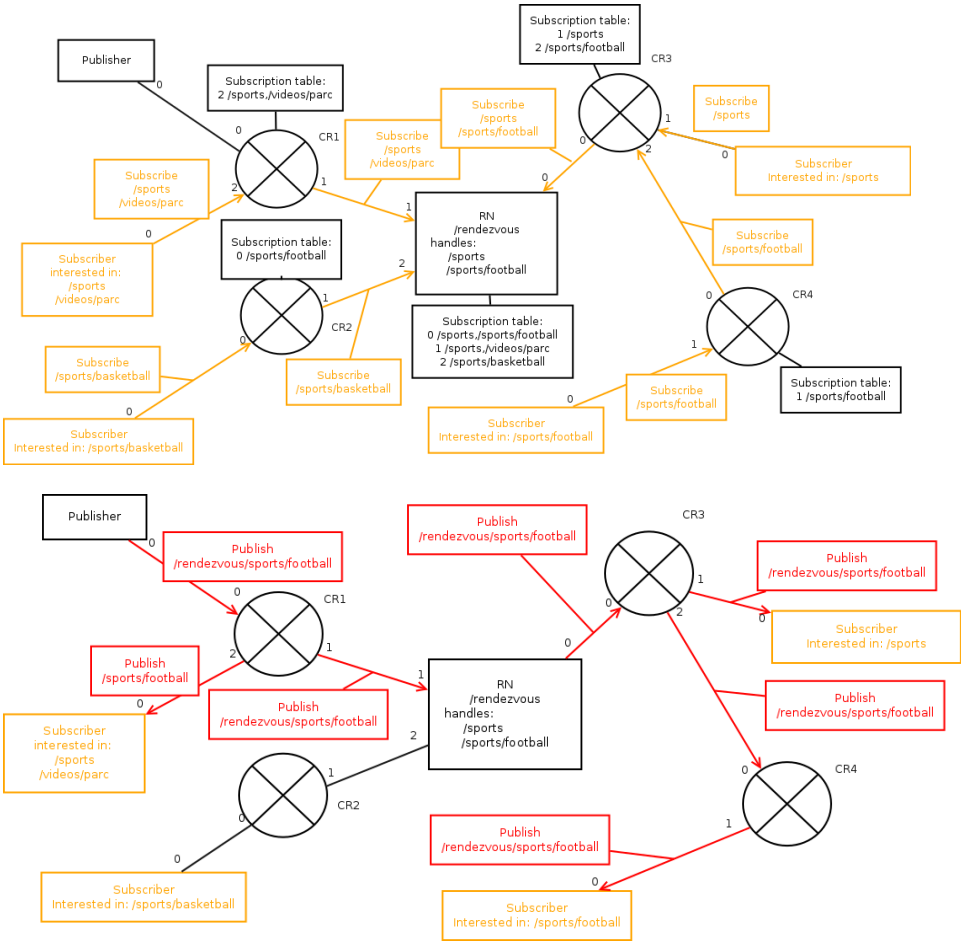


Figure 5.11: The subscription and publishing process in COPSS.

Chapter 6

CCN design challenges

Although CCN has many advantages (see section 2.3), the unique architecture proposed by it poses some unique challenges not faced by the current design of the internet. Although only CCN is discussed in this chapter, many of the same issues apply to other NDN technologies like DONA and TRIAD.

6.1 Content revocation

Inevitably, some content will be created that is not intended to live forever, or that has to be superseded by a new version. Additionally, cryptographic keys to sign content may need to be revoked if they are compromised or no longer trusted. All users of the data signed by such a key would need to be notified of the revocation. CCN does not specify an algorithm for either data or key revocation, despite the obvious need for such mechanisms. A possible way to implement this would be a revocation list that is periodically published by routers, and also enforced by them. The disadvantage is that this, too, would need to be signed to prevent abuse by malicious third parties. Alternatively, content could be released with a limited lifespan, after which cached copies are no longer served. This approach has the disadvantage that content publishers would need to

serve their content more frequently, increasing their load as well as the load on the network in general [Lau10].

6.2 Security

Even though CCN has been designed with security in mind, there are still some serious concerns. In this section, the principal concerns will be sketched out.

6.2.1 Architectural reliance on cryptography

CCN relies heavily on cryptography to secure its contents, but this means that one form of cryptography might become a standard part of the system which is difficult to replace[1]. As shown by history, cryptographic techniques eventually become obsolete because computing power tends to increase to the point where brute-force attacks become possible. The time between the invention of a strong encryption algorithm and its obsolescence is generally considered to be about 30 years [SJ09, Lau10].

6.2.2 Denial of Service

Denial of Service (DoS) attacks against content sources in CCN are more difficult than in IP because of the need to avoid using the same names (the response would then be cached, and the DoS attack would only tax the edge router). However, by using a name with the same prefix but a different postfix each time, an attacker could ensure that all of its Interests make their way to the original server, and overload it. The adverse effect of DoS attacks is that they can either make content unavailable to legitimate users,

[1] This depends on the concrete implementation. If the encryption method is negotiated between CRs rather than it being an integral part of the specification, this will not be a big issue.

or force wrong responses to reach the clients. Both of these goals could be accomplished in a variety of ways, some of which are listed below (see figure 6.1 for an overview) [Lau10].

Disrupting the source By simply flooding the source with Interests, an attacker can increase the load on that source in two ways: by flooding it directly and by decreasing the efficiency of caching in intermediate routers, which means most of the requests of legitimate users would be sent directly to the source instead of being processed by a cache, which would dramatically increase the load as well. This does assume that it is possible to construct a large number of Interests which will all be routed to the same content source.

Routing disruption Routing can be hijacked by having malicious or compromised routers in the network that do no forward requests, or by forcing routers to have bad timeouts for responses to Interests in its Pending Interest Table (PIT). It is not clear how such a thing could be accomplished.

Disrupt an intermediate router Routers could be disrupted by forcing them to conduct expensive operations, which would slow them down to such an extent that they can no longer properly function as network elements. One such technique is forcing the router to verify signatures for content items that are signed with different keys, forcing the routers to fetch each of these keys. If the keys are hosted by a malicious server under the control of the attacker, that server could delay the delivery of those keys. Also, an attacker could choose the keys that are most computationally expensive.

Disrupt links Links can be disrupted by flooding them with a large amount of Interests so that the link reaches its maximum capacity and requests will be queued in routers. This can also be a side-effect of other kinds of DoS attacks.

Returning fake content Because Interests are often broadcasted, an attacker could pick up an Interest which it should ignore and generate a fake response to it that is not signed (or is signed with the wrong key), hoping that the clients will not verify the credentials; or attackers could "replay" old, signed content as the response to an Interest that would normally return newer content (or different content from the same source). An attacker might also gain access to a source's signing key and thus be able to spoof content at will.

Blocking valid content Requests for valid content can be blocked by routers that believe the content does not actually exist. An attacker could simply generate a 'Content does not exist' response to Interests, in the same way that they could generate fake content.

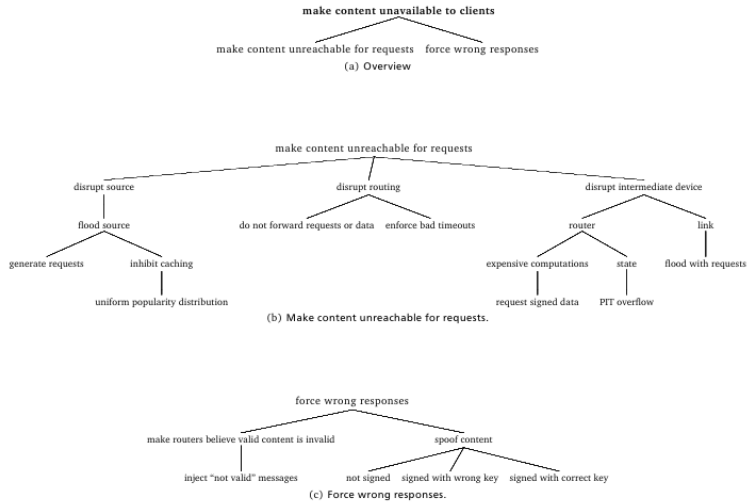


Figure 6.1: An overview of the different possibilities for DoS attacks in CCN [Lau10]

6.3 Privacy

Because in CCN content is cached by intermediate routers, CCN is prone to a number of privacy concerns which are less prominent in the current internet: while content is only occasionally cached in the internet (at the application layer), in CCN caching is an integral

part of the design. Attackers can figure out whether certain content is cached or not by measuring the response time when requesting the content, and even encrypted content can 'leak' some sensitive information in the form of meta-data such as the name, the public key of the host, the content length, and the request time. Because all data is cached, attacks do not need to be performed in real time, but can happen long after the actual conversation took place.

Another big privacy concern are ISPs, which will have their own content routers. The availability of cached content on its routers will make it trivial for ISPs to spy on its own customers by monitoring their edge routers, and logging requests. The only defense that could be used against such snooping would be to use obfuscated names, but their use would either work against caching by having different names per user, or allow ISPs to construct a dictionary mapping obfuscated names to content sources, making the obfuscation ineffective. ISPs could also create a database of content hashes and match incoming content against this database, to profile users in terms of content use. As always, whether this is a big issue depends on the opinions of the people that use the network. Being able to effectively monitor the content passing through its network will make it easier for ISPs to cooperate with law enforcement, for example to ban illegal content such as child pornography [Lau10].

6.4 Accountability

In traditional IP-based networks, if a host is under attack, it can at least get some information about where an attack is coming from. The IP address can be faked, but not if the attacker needs to receive answer packets. In CCN, an attacker could attack a content server with impunity because only Interests are routed, while Data responses simply follow the path back to the attacker. This means that content servers only know the previous hop of an attack, but not its origin. This is problematic because a content server can not blacklist an attacker, making traditional measures against DoS attacks unusable. In order to properly identify where an attack came from, there would need to be a special protocol to look up the source of an Interest, or all the ISP access routers would need to be mon-

itored by law enforcement agencies. An alternative solution would be to require users to sign all their Interests, or to do so automatically at the access routers, using a key provided by the ISP (because the ISP knows what client an Interest came from, it could provide a unique key for each of its users). These keys could be changed every day to maintain the users' privacy. This solution would allow the server to block a specific user (on the basis of the key used to sign the Interest), even if he remains anonymous. A downside of this approach would be that it makes aggregation of Interests more difficult. [Lau10]

6.5 Paid and sensitive content

The CCN architecture was designed, and works well, for publicly accessible content without access restrictions, but supporting restricted content is more difficult. Because restricted content would also be cached, a person wishing to receive the content without paying the publisher simply needs to figure out (or guess) the content name, provided that the content is cached in a router along the path from the miscreant towards the source. Publishers also may want to only make content available to a specific user for a limited amount of time, which could be very difficult to accomplish because of caching. Also, a person paying to access restricted content could easily share its access with users who do not pay by sharing the content names. Encryption schemes could be worked out between client software on users' machines and the content servers, but once that encryption scheme is compromised, all cached content would be available for all to view. This underscores the need for a revocation scheme for content in CCN (see section 6.1). Also, a publisher may want its sponsored content to be accessed only from a certain playback environment that includes advertising. [Lau10]

6.6 Scalability

Push networking The design of CCN uses a 'pull' model for accessing content: content is filtered through to the client upon ex-

plicit request. A lot of real-life applications are based on a 'push' model instead, where a server pushes a continuous stream of data (such as an audio stream, or timing information) to clients. In CCN, this can be accomplished in two ways: by pushing Interests or by continuous polling. In the Interest pushing method, the content server sends its data as (part of) the name of an Interest packet towards the client, which is inefficient considering that for each Interest packet the client needs to send a Data packet back towards the content server containing the same name as the Interest. Embedding content in names compromises security and privacy unless it is somehow encrypted or obfuscated. Also, both client and server would need to be reachable under globally unique names. In the other method, the client sends out a constant stream of Interests towards the content server, and the data is sent back as Data packets in reply to these Interests. This means that client and server need to agree on an algorithm for generating predictable names for content that is not yet published. So, while implementing push networking in CCN is certainly possible (as shown in the VoCCN implementation, see section 4), it is not simple to implement this in an efficient manner.

Moving the focus from nodes to content also raises scalability requirements to the extreme. The Internet addressing deals today with on the order of 10^9 nodes. When content is directly addressed, FIB tables will become several orders of magnitude higher, since there is more data than there are nodes on the internet, and content names take up more space than IP addresses.

Broadcast and multicast As with push networking, the problem in implementing broadcasting and multicasting in CCN is that CCN can only accept data responses after an Interest for it has been sent. In a multicast environment, that means that not only must all the participants agree on the names to use for data, but they need to keep sending Interests regularly regardless of whether or not the other participants have new data they wish to share. Even with the suppression of duplicate Interests at the router level, this will still dramatically increase the load on the network.

Reliability CCN itself makes no guarantees for reliability or fairness, and no transport-level protocols on top of CCN have been proposed for supplying either. Interests can time out, but it is currently not defined how such time-outs need to be chosen or who is responsible for re-sending Interests if they do time out.

Hardware CCN routers necessarily need to be more powerful than IP routers because they have more tasks: they need to be capable of verifying signatures, to keep state for each Interest, possibly run algorithms to detect and prevent Denial of Service attacks, keep a lot of content in a local cache and work at speeds similar to modern IP routers. This requires a lot of processing power and large amounts of fast, random accessible memory (if content is stored on a hard disk in the router instead of in RAM, the delay for the client is likely to be worse than if it was not cached at all because of hard drive seek times, although the move towards faster SSDs may alleviate this problem). The FIB, PIT and content cache each impose big memory requirements to be effective.

Energy efficiency In general, CCN is thought to be a more energy efficient way to do networking because it reduces the number of hops. In one test, it was shown that even a 20% deployment of CCN can reduce energy consumption by 15% [LRH10]. However, other sources dispute that the savings would be that big in real-world conditions [Lau10]. CCN routers are more complex than routers in an IP network and would probably draw significantly more power.

A lot of these issues will be affected by the decision of how to implement CCN: if CCN is implemented on top of IP, use cases which are difficult to implement in CCN can still use IP-based protocols. This is also the best option for gradual deployment of CCN. However, in such an over-the-top deployment it will be tempting for service providers to stick to traditional solutions.

6.7 Performance

For the discussion of the performance of CCN, it is important to realize that different usages of the network can produce highly varying performance statistics. If the network is used only to distribute publicly available content, CCN can be more efficient than TCP/IP at this dissemination in some cases, such as when the most popular 1% of content is requested 99 times out of 100, which makes caching in the network very attractive. In the opposite case, when caching is ineffective or useless, such as when each piece of content generated on the network is encrypted or customized for each client, CCN's caching features will not play a role in the network performance. For a more detailed discussion, see section 9.3.

Chapter 7

Conclusion

In this thesis, a new networking paradigm called “Named Data Networking” is described, which routes requests based on content names. This contrasts with the current, IP-based internet that requires a separate lookup phase to find the location of content. The fact that in Named Data Networking content is placed centrally aligns nicely with the primary usage of the internet, which consists to a large degree of accessing named content. Furthermore, Named Data Networking can speed up lookups by eliminating round-trip times and by providing possibilities for network-integrated transparent caching. One technology, called CCN, was studied in detail. It promotes a simple pull-based networking doctrine using Interest (requests) and Data packets (responses). It binds the name of the content to the content itself to make it impossible to corrupt in transit, and also supports encryption natively, whereas IP requires additional infrastructure for this.

However, while NDN in general and CCN in particular are able to solve a lot of the problems that in IP-based networks require complicated workarounds, it also provides its own set of difficulties and problems. In particular, generation of content names can be problematic, as there is no built-in convention and different applications might see the need for different standards. Also, CCN headers have a very high overhead compared with TCP (or even HTTP). Integrating authenticity and encryption in the network layer also has the potential to diminish the performance of routers because of the high

amount of processing per packet required, and standardizing on any given type of encryption or hashing technology is risky because of the evolving security landscape. Content and key revocation issues are also still largely un-addressed.

The aim of this thesis is to examine whether CCN can be a viable replacement for the TCP/IP stack (see section 1.1.3). So far, the theoretical foundation of CCN was examined to try to determine an answer to this question. The issues mentioned above still stand in the way of CCN becoming a viable internet replacement in the immediate future. However, most of these problems can be solved with sufficient research and investment. A gradual deployment of CCN seems like the most viable option for deployment, either by using CCN-to-IP gateways (as discussed in the context of VoCCN, see chapter 4) or by initially implementing CCN as just another transport protocol on top of IP. In my opinion, CCN will not become a major success, because of the huge investments required by the intermediaries, and because the benifits of CCN would only show up once it was relatively widely deployed. The second part of this thesis will try to provide a more detailed view of CCN by examining the technology's performance in the field of video streaming.

Further research is needed on the following issues:

Content revocation / expiry The need for either a content revocation protocol, or provisions to allow content to be expired has been discussed in section 6.1. How to best implement either option is still open for debate.

Key revocation An infrastructure that depends a lot on public and private keys needs to have a mechanism to allow keys to be revoked if they are compromised. This could be implemented using key revocation lists and key status query mechanisms. CCN currently has no such mechanism.

CCN energy efficiency While some have argued that a CCN architecture for the internet would increase energy efficiency, this is far from certain (see sections 2.3 and 6.6). More research needs to be done to determine the approximate energy efficiency of CCN in realistic deployment scenarios.

CCN-to-IP gateways As yet, there is no efficient, general-purpose CCN-to-IP gateway. More research on the viability of such

gateways in a general case (rather than separately for each application-layer protocol, such as for VoCCN (chapter 4)) is essential to determine if such a setup is viable.

Part II

Implementation & Testing

Chapter 8

Setup

8.1 Introduction

An important use case for CCN is the efficient distribution and streaming of multimedia content to end users. Indeed, the proportion of internet traffic devoted to multimedia content keeps on rising; it now accounts for over half of the internet traffic worldwide [San]. The DASH protocol has been recently introduced for optimal video distribution to consumers over HTTP (see section 2.4.2). For this reason, the combination of DASH and CCN is especially interesting: by combining the efficient caching features of CCN with the adaptive streaming features of DASH, it is possible to both increase end-user media quality and reduce the load on intermediate nodes, particularly for ISPs.

8.2 Test goals

In this thesis, the focus will be mostly on efficient video delivery by leveraging the strengths of both CCN and DASH. In particular, the efficient distribution of video content over multiple links will be investigated, as well as the most efficient parameterization of chunk

size and CCN window size for efficient content dissemination.

Multilink transfer CCN's design is inherently link independent, while a TCP connection is bound to a particular interface. This offers the possibility not only of using the best link available at any time to fetch DASH video segments (as in [LMR+13a]), but to fetch different segments over different interfaces in parallel.

Chunk size parameterization The overhead of the CCN protocol is about 650 bytes, mostly due to the security features underpinning CCN (see section 3.6). Ccnx [CCN], the open-source CCN network stack, uses a chunk size of 4 KB by default, which means that about 16% of the packet is CCN communication overhead. By increasing the chunk size, we can reduce the overhead of CCN to make it competitive with TCP. Additionally, there is a large amount of computation that must be performed for each chunk, which means a small chunk size causes a large CPU load on the computer in question. The reason the chunk size is kept low by CCNx is that the segmentation of CCN chunks into IP packets when sent over UDP/IP causes CCN performance to decrease when competing for limited bandwidth with other (e.g. TCP-based) protocols, since the loss of any part of a chunk will force the CCN implementation to re-send the Interest for it. Conversely, when not contending with other protocols, quite large chunk sizes, in the order of about 256 KB should be feasible [SDC+12]. The impact of CCN chunk size on the transmission of video using DASH over CCN will be examined, in order to improve the efficiency of video delivery.

Interest window size parameterization Previous research has indicated that CCN does not make efficient use of network resources when the RTT delay is big [LGP+13]. We will examine the influence of the CCN Interest window size on the average video bitrate and compare it with the HTTP/1.1 pipelining performed by the authors of [LGP+13].

8.3 Related work

[LMR+13b] analyses overhead from headers and protocol communications when using DASH over CCN in comparison with HTTP/1.0 and HTTP/1.1. They also investigate delivery performance. They show that DASH over CCN has a lot more overhead than DASH over either HTTP/1.0 or HTTP/1.1 (about 23.5-25% of network traffic as compared to HTTP/1.1 5-12%). For delivery performance, they show that CCN's performance is roughly equal to HTTP/1.0 (and HTTP/1.1) when the RTT is really low, but that CCN's poor link utilization at high RTT and its overhead produce much lower bitrates at higher RTTs. Their analysis however hinges on the assumption that the CCN chunk size has to be 4 KB (it doesn't; [RR11] suggests that chunk sizes smaller than 10 KB are sub-optimal because they generate too much overhead. [SDC+12] suggests that segment sizes of hundreds of kilobytes may be more appropriate). Furthermore, they are using CCN over UDP/IP, which brings with it a slight increase in overhead compared to running CCN natively. Finally, they acknowledge that their DASH over CCN implementation is quite basic and could be further extended for efficiency.

[LGP+13] presents the author's DASH over CCN implementation (DASC) and investigates the performance and caching of CCN in the context of DASH. They show that a setup with a single CCN client performs worse than using a single HTTP/1.1 client; because of CCN's higher overhead, the client witnesses a lower average video bitrate and hence a lower video quality. However, they also establish that as more clients request parts of the same video stream that are already cached, the clients who come later get progressively better quality video. By the 8th DASC client, video bitrates were 500 KB/s for over 95% of the time even though the bandwidth to the origin server was limited to 250 KB/s. There is a hitch, though: because of the ubiquitous caching that is taking place within the CCN network, the throughput estimated by the DASH algorithm is inaccurate (i.e. over-estimates the available bandwidth occurs when the requested content is served from the cache instead of from the origin server). When requesting chunks at a higher bitrate that are not yet cached, video playback will stall for that reason.

[LMR+13a] examines the performance of DASH over CCN in mobile environments with multiple links. Its conclusions are that

in general, DASH over CCN can compete with some DASH implementations regarding average media bitrate and number of quality switches, but not with the most efficient ones. Regarding multi-link communication, they note that CCN uses the bandwidth of the fastest link available, but does not split up its data transmission between multiple links, so that the effective throughput is limited by the bandwidth of the fastest link available at any given time, rather than the sum of all available links.

8.4 Test methods

Several test methods were considered for investigating the test goals outlined in section 8.2: simulation, using the existing DASH player over CCN, and using a custom test bed.

8.4.1 Simulation

For the purposes of measuring the effects of the window size and chunk size, we can make use of a simulation environment called **ndnSIM**[ndn], which is a module within the popular **NS-3** network simulation environment. NS-3 simulations consist of small C++ programs that can make extensive use of pre-written classes which simulate every part of the networking process, from detailed modeling of link propagation losses to inter-domain routing and TCP retransmissions. A basic NS-3 simulation looks like this:

Listing 8.1: NS-3 echo example

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/applications-module.h"
using namespace ns3;

//Used as a prefix for log messages
NS_LOG_COMPONENT_DEFINE ("Example");
```

```
int main (int argc, char *argv[])
{
  //Tell the echo client and echo server to produce
  //logging output of the 'INFO' log level and above.
  LogComponentEnable ("UdpEchoClientApplication",
     LOG_LEVEL_INFO);
  LogComponentEnable ("UdpEchoServerApplication",
     LOG_LEVEL_INFO);

  //Create the end nodes
  NodeContainer nodes;
  nodes.Create (2);

  //Create a Point-to-Point (PPP) connection
  PointToPointHelper pointToPoint;
  pointToPoint.SetDeviceAttribute ("DataRate",
     StringValue ("5Mbps"));
  pointToPoint.SetChannelAttribute ("Delay",
     StringValue ("10ms"));

  //Connect the devices using the Point-to-Point
     connection
  NetDeviceContainer devices;
  devices = pointToPoint.Install (nodes);

  //Install a standard IP stack on the nodes
  InternetStackHelper stack;
  stack.Install (nodes);

  //Assign IP addresses to the PPP end points
  Ipv4AddressHelper address;
  address.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer interfaces = address.Assign (
     devices);

  //Create a UDP echo server on the second node
  UdpEchoServerHelper echoServer (9);
  ApplicationContainer serverApps = echoServer.Install
     (nodes.Get(1));

  //Tell the server to start after 1 second and run
     for 10 seconds.
  serverApps.Start (Seconds (1.0));
  serverApps.Stop (Seconds (10.0));
```

```
  //Install a UDP echo client on the server.
  //It will send 5 1024-byte packets
  //with 1 second intervals between packets
  UdpEchoClientHelper echoClient(interfaces.GetAddress
      (1),9);
  echoClient.SetAttribute("MaxPackets",UintegerValue
      (5));
  echoClient.SetAttribute("Interval",TimeValue(Seconds
      (1.0)));
  echoClient.SetAttribute("PacketSize",UintegerValue
      (1024));

  //Install the echo client application on the first
      node
  //Tell it to start after 2 seconds, and run for 10
      seconds.
  ApplicationContainer clientApps = echoClient.Install
      (nodes.Get(0));
  clientApps.Start (Seconds (2.0));
  clientApps.Stop (Seconds (10.0));

  //Run the simulation.
  //Will block until both client and server have
      finished
  Simulator::Run ();
  //Clean up
  Simulator::Destroy ();
  return 0;
}
```

This code will run a simulation using a UDP echo server on one host in the 10.1.1.0/24 network listening on port 9. Another host ('node') is connected to the first one with a P2P (point-to-point) connection with a delay of 10ms and a maximum data rate of 5 Mbps. The echo client will send 1 packet per second for 5 seconds, each 1024 bytes in size. The server will then echo these packets back to the client. The server starts up 1 second into the simulation, the client after 2 seconds. After 10 seconds, both client and server shut down and the simulation ends. Globally speaking, the simulation sets up the various links, stack, addresses, configuration and applications.

Each of the key CCN technologies are implemented. For instance, the CCN FIB is implemented as the class *ns3::ndn::Fib* which has methods such as

```
Add (const Name &prefix, Ptr< Face > face, int32_t metric)
Find (const Name &prefix)
Remove(const Ptr< const Name > &prefix)
```

which respectively add a new prefix to the FIB, search for the face(s) that correspond with a specific prefix, and remove a prefix from the simulated FIB.

On a higher level, there are "helper" classes such as *ns3::ndn::StackHelper*, *ns3::ndn::AppHelper* and *ns3::ndn::GlobalRoutingHelper* to help set-up the CCN stack, application bindings and global CCN routing.

The ndnSIM simulation environment is very convenient for testing certain properties of CCN, but it does not fit in very well with the stated desire to experiment with DASH over CCN. There is not yet any simulation of DASH in NS-3, and it would furthermore be difficult to integrate with the data set used in [LGP+13, LMR+13b], so this approach was discarded.

8.4.2 DASH player

To demonstrate the effectiveness of CCN as the underlying protocol for the transportation of DASH streams, *qtsampleplayer* was used. This reference DASH player is shipped with libdash, the reference DASH implementation[1]. This player is available in both a HTTP and a CCN version; the CCN version required several manual fixes to get it working. Also used was code from a fork by arawind[2] that added logging and dynamic quality adaptation, which is lacking from the base version. This dynamic quality adaptation was eventually dis-

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

[2] Available at `https://github.com/arawind/libdash-sampleplayer`, cloned on 7 July 2014

abled, because it did not work to the level needed for this feature, either with HTTP or with CCN.

This player provided a working proof of concept of DASH over HTTP and of DASH over CCN, but even with the logging added from the fork, it was unsatisfactory because it lead to unrepeatable results and did not easily allow variations of chunk window size, segment window size and RTT, which are all parameters that required tweaking for the testing, so this approach was discarded in favor of using a fully scripted setup (see next section).

8.4.3 Testbed environment

In addition to simulating a CCN environment, a testbed setup was created with real computers networked using the CCNx network stack (see [CCN]). This approach was selected because it allows tweaking of the parameters whose effects needed to be examined, and still approximates a real deployment of CCN. Because there is no simulation, and very little overhead on top of CCN involved, it can be established beyond doubt that the most efficient approach is taken, and the results can be compared with those of [LGP+13] and [LMR+13b].

CCNx is a full CCN networking stack, but it runs exclusively on top of either UDP or TCP, instead of running directly on top of the hardware. In this thesis, only CCN over UDP is considered, since this is closer to what a "raw" CCN implementation would be like, and because TCP has flow control and other mechanisms which are duplicated by CCN and which might interfere with the results. Each computer was running identical versions of the CCNx stack. This stack was modified to remove artificial limitations imposed by the stack on the size of chunks (8192 bytes) and Interest windows (128). The CCNx version in use was version 0.8.2, the latest version at the time of writing.

The testbed setup itself consisted of 3 computers were connected to a 100 Mbps Ethernet switch (see figure 8.1). The computer labeled 'server' hosted the content, the computer labeled 'client' down-

loaded the content, and the computer 'cache' was not used in the test. It was intended to use the third computer to simulate the performance of CCN over multiple hops, but this part of the test was later discarded.

For testing the raw performance of sending chunks using CCN, the utilities ccnsendchunks and ccncatchunks2 were used. These programs are provided along with the CCNx stack. In addition, a new program was created using the CCNx API to test the simultaneous downloading of several small files as needed for DASH over CCN. The code for this application can be found in appendix B.1.

When comparing against HTTP, the cURL downloader[3] was used, either the main binary, or a custom application coded against its API (for simultaneous downloads). The code for the custom HTTP download application used can be found in appendix B.2. The version of cURL and its libraries that was used was 7.35.0. On the server side, the lighttpd HTTP server[4] was used.

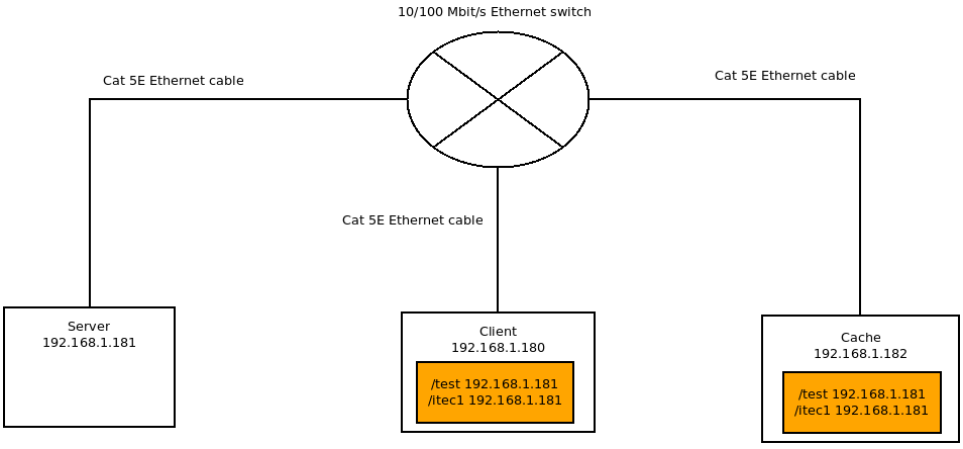


Figure 8.1: The testbed setup used for the experiment. Note that the cache was not used in any of the main experiments. The part in orange is a simplified representation of the FIB of those hosts.

The specifications of the computers used in this setup are:

[3]To be found at `http://curl.haxx.se`
[4]see `http://www.lighttpd.net`.

Name	IP	Processor	Mem	Kernel
Client	192.168.1.180	Intel® Core™ i5-4670 @3.40GHz	8GB	Linux 3.13
Server	192.168.1.181	Intel® Core™ 2 Duo P8600 @2.40GHz	4GB	Linux 3.13
Cache	192.168.1.182	Intel® Core™ i3-3110M @2.40GHz	4GB	Linux 3.15

Table 8.1: Testbed computer specifications

8.5 Experiment selection

The target of the experiments is to look at the viability of CCN as a means for e.g. streaming video in a DASH-like way. For this, it is important to first look at the parameters that govern large file downloads (i.e. a 100MB file), as to isolate the underlying variables independent of start-stop effects when downloading multiple (smaller) files (.e.g. DASH segments). Also, it can be argued that since CCN already does segmentation by splitting content up into chunks, this is actually one of the more realistic CCN media streaming options. The disadvantage of this approach is that switching between different representations would need to be handled differently compared to the DASH philosophy. Once this is established, a second experiment looks more in detail at some of the same variables, in particular when downloading smaller files of 2 MB each. We perform the download 50 times, both to be able to compare the results with the first experiment, and to average out the measurements so small aberrations do not adversely affect the result. Finally, an experiment was conducted with a real CCN stream using the same data set as was used in [LGP+13] and [LMR+13a]. In contrast to those papers, this experiment does not examine how CCN responds to variations in the available bandwidth but instead the impact of various settings that affect CCN performance in various network delay environments is tested.

Chapter 9

Experiment 1: Large file download

9.1 Motivation and setup

The first thing to accomplish was finding out how CCN performed when downloading a single large file, before looking how to deal with multiple smaller files as needed for DASH. The purpose is to look at the impact of the chunk size[1], chunk window size[2], and network RTT[3], while minimizing the impact of start and stop effects (CCNx implements a slow start mechanism, similar to TCP's). This constitutes experiment 1: the download of a **single large file** of 100 000 000 bytes (100 MB), using a **100 Mbit/s** link (enforced by the 100Mbit/s Ethernet switch), between two hosts directly connected to the same switch, while varying the **chunk size**, **chunk window size** and **RTT**. Varying the network delay was accomplished using the Linux *tc* utility with the *netem* filter, to artificially introduce a delay in outgoing packets on the host acting as a server, and thus

[1]Size of the content of a Data packet ('chunk'), excluding headers. The header overhead is the same in each packet (regardless of size), hence larger packets have a smaller header relative to packet size.

[2]The number of chunks simultaneously downloading, also referred to as Interest window size.

[3]Round-Trip Time, the time needed for a packet to travel from host A to host B and back.

increasing the RTT. The following **chunk sizes** were tested: **1024**, **2048**, **4096** and **8192** bytes (the maximum allowed by the CCNx stack without modification). Chunk sizes of 256 and 512 bytes were also briefly considered, but preliminary testing indicated they were too inefficient to be viable and would stretch the experiment out for several more days without adding any real value. For the **chunk window size**, values of **1**, **20**, **100** and **400** were tested. For the **RTT**, delays of **0**[4], **10**, **40** and **100** ms were examined.

9.2 Code

Listed below is the code used for this experiment. It was written in bash, in order to make use of its command chaining properties. The same script uses ssh to control both the client computer (on which the script is executed) and the server computer, where the chunks are generated. Results for HTTP are also generated; the curl tool is used for this purpose. There is no variation of chunk or window sizes for HTTP, given that this is not usually applicable[5].

Listing 9.1: Experiment 1: Transmission of a single 100MB file

```
#!/bin/bash
# Author: Frederik Van Bogaert
# Send a 100MB file using CCN from the other host in your
    CCN network.
# Prerequisites:
# - Both sides have ccnd running
# - sshd is running on the other host
# - ssh is configured for passwordless login
# - tc with netem is installed on the other host
# Script parameters: other host hostname or IP, CCN name
    and output file
# CCN parameters tested: chunk size, RTT and CCN Interest
    window size

OTHER_HOST=192.168.1.181
```

[4] 0 ms delay is impossible to achieve in practice; packet storing and forwarding in the network cards of both computers and the intermediate switch, as well as a slight cable transfer delay, create an effective average RTT of 0.234 ms (σ 0.085 ms), as measured by the ping utility.

[5] Chunked downloading using HTTP can be emulated, e.g. by using HTTP byte ranges; this was not considered for this test as the HTTP test is just used to establish a baseline for comparison.

```
CCN_NAME=ccnx:/test/file
LOGFILE=out.log
REMOTEFILE=/var/www/ccndata.tmp

rm $LOGFILE
echo "Creating file on $OTHER_HOST"
ssh $OTHER_HOST dd if=/dev/urandom of=$REMOTEFILE bs
    =1000000 count=100
#Make sure the proper FIB entry is in place
ccndc add $CCN_NAME udp $OTHER_HOST
for rtt in 0 10 40 100 200;
do
  echo "Setting RTT to ${rtt}ms"
  #Use tc on the server to set a delay on outgoing packets
      from the server
  ssh $OTHER_HOST tc qdisc replace dev eth0 root netem
      delay ${rtt}ms
  for chunk_size in 1024 2048 4096 8192;
  do
    for window_size in 1 20 100 400;
    do
      echo "Preparing to send data with chunk size
          $chunk_size"
      #Remove cached copies from both hosts
      #This ensures the segments will be re-chunked and re-
          sent every time
      ssh $OTHER_HOST ccnrm $CCN_NAME
      ccnrm $CCN_NAME
      #Tell the server to serve the chunks. This is done in
           the background.
      #ccnsendchunks will convert the input file into
          chunks
      #and put them in the server's content store.
      ssh $OTHER_HOST "ccnsendchunks -b $chunk_size
          $CCN_NAME <$REMOTEFILE &" &
      #Wait for a while to be sure the content is available
      #  from the server's content store
      sleep 5
      #Get the content. Sends Interests for chunks of the
          file
      #  to the server and waits for all the data to arrive
      #The output of the command is captured in RESULTLINE
      RESULTLINE=$(ccncatchunks2 -d -p $window_size
          $CCN_NAME 2>&1)
      if [ -z "$RESULTLINE" ]; then exit; fi
      #Filter the result, to obtain the time needed to
          download the file.
      RESULT=$(echo $RESULTLINE | egrep -o "[0-9.]+ seconds
          " | cut -d' ' -f1)
```

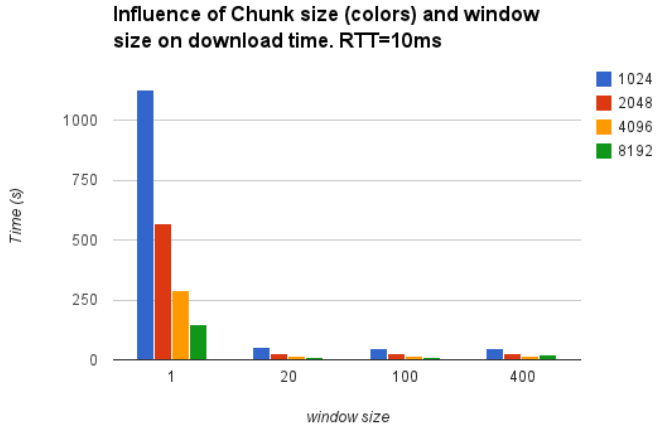
```
      #Log the parameters and the result in CSV format to a
          file
      echo "$rtt;$chunk_size;$window_size;$RESULT" >>
          $LOGFILE
      #Stop the server process, to be sure.
      ssh $OTHER_HOST pkill ccnsendchunks
      echo "Done in $RESULT seconds"
    done
  done
  #Repeat the experiment for HTTP using cURL.
  #Note that chunk and window sizes were not varied,
  #so there is just one result for HTTP per RTT value.
  #The 'time' utility is used to obtain the run time,
  #since this is difficult to derive from curl output.
  #time -f "%e" returns total time between
  #starting the process and the process stopping.
  echo "$rtt;http;1;" $( ( /usr/bin/time -f "%e" curl -o/
      dev/null -s http://${OTHER_HOST}/$(basename
      $REMOTEFILE) ) 2>&1 ) >>$LOGFILE
done
#Remove the test file.
ssh $OTHER_HOST rm $REMOTEFILE
#Set the network delay back to its default setting.
ssh $OTHER_HOST tc qdisc del dev eth0 root
```

9.3 Results

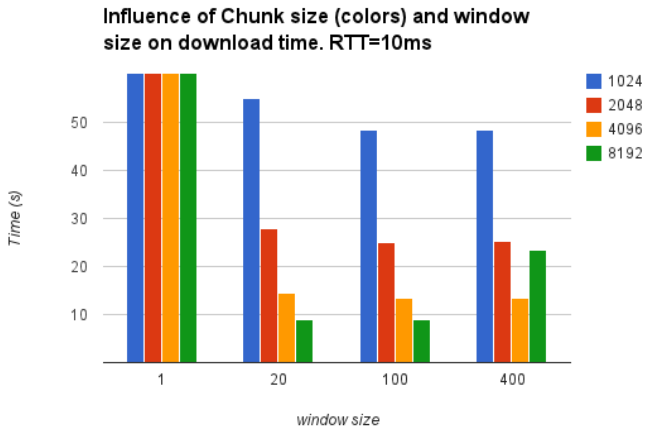
As was expected, a window size of one leads to very large download times due to inefficient link utilization. The download time decreases linearly with increases in chunk size and window size at low link utilization. As long as the link is not saturated, the throughput is limited by the formula: $(1/RTT) * (Windowsize) * (Chunksize)$. This is due to the fact that CCNx will not issue new Interests beyond the window size as long as the first chunk is not received. Each Interest corresponds with exactly one chunk (Data packet), so the maximum data outstanding at any one time is given by $(Chunksize * Windowsize)$. Because it takes at least *RTT* time for an Interest to result in a received Data packet, we arrive at the formula above.

An additional limit is the link data rate, at 100 Mbit/s, plus

the fact that header overhead also needs to be accounted for. This results in a theoretical lower limit of 8 seconds to download this 100MB file ($(100MB)/(100Mbit/s) = 8s$). In practice, the large header and delays in the CCNx stack implementation mean that the effective limit is about 8.8-8.9s in this test.



(a) Total view



(b) Zoomed in on t=[0-60]

Figure 9.1: The effect of chunk and window sizes on the download time of a large file. Note that a window size of 1 is very sub-optimal in comparison with the alternatives.

Our experiment uncovered a third limit, caused by the CCNx stack's computational overhead: the stack was unable to serve more than 2000 chunks per second in our tests, in various configurations. In practice, this means for instance that when the chunk size is 1000 KB, the download speed is limited to 2MB/s. The cause of this last limit is the Pending Interest Table, which has serious per-

formance issues. To investigate this issue, a tool to monitor both the CPU load and the network throughput were used. the results are shown in figure 9.2. Given that the CPU is the limiting factor in this case, the choice of CPU for the server becomes important. As was established in table 8.1, the CPU in this case is a slightly older Intel Core 2 Duo. With a better CPU this problem would not be as prevalent. On the other hand, CPUs in routers tend to be less powerful than those in desktops.

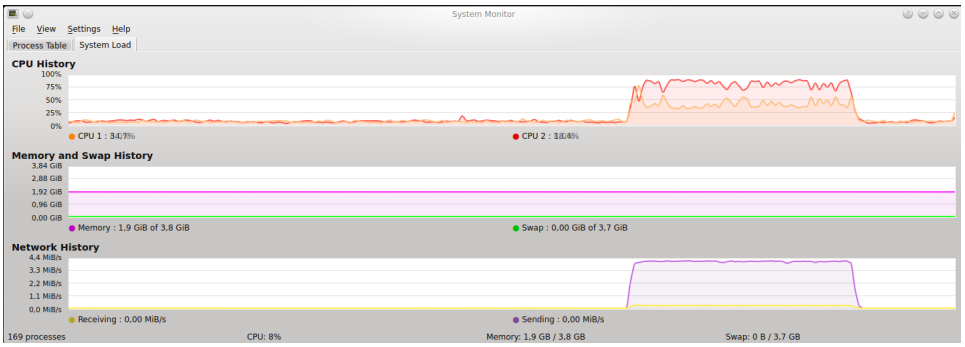
With high chunk sizes, high window sizes and very low RTT, there is another problem: the server tries to put too much data on the link, and because of that it fails to send all the data. Specifically, the sendto() function fails because the IP packet output buffer is full, which causes the server to drop the packet. In turn, the absence of some chunks causes the client to ask for the data that was lost to be retransmitted after a timeout (in this case, 4 seconds) has elapsed. This causes a sizable drop in the effective throughput due to chunk download timeouts and the need to retransmit those chunks.

To summarize, we reach a limit when:

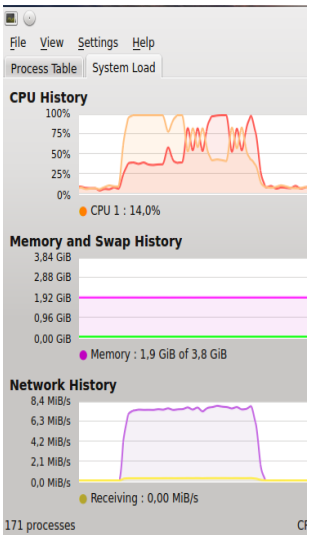
- **Link speed**
  The link speed is reached (100Mbit/s in this case). This is visible in the case where the chunk size is 8192 and the window size is 20 or 100 in figure 9.1(b): the download time here is very close to the ideal case of 8s.

- **Link underutilization**
  The RTT is too long for efficient link utilization due to the fact that packets do not arrive in time so the client has to wait to send new Interests. This effect can be mitigated by increasing the window size or increasing the chunk size. It is especially visible when the window size is 1 in figure 9.1(a).

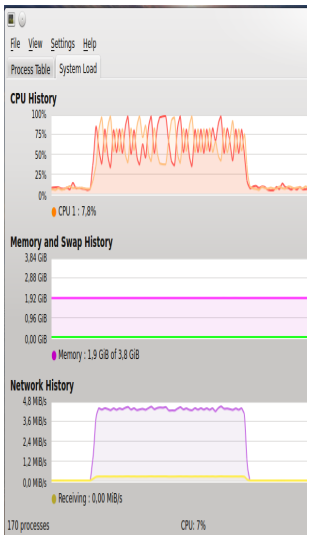
- **Server CPU overload**
  Interests are sent out at a rate higher than 2000 per second, leading to a CPU limit on the server. This effect explains why the results for window size=20 and window size=100 are so similar in figure 9.1(b): this limit is reached, so increasing the window size does not help. See figure 9.2 for CPU load plots.



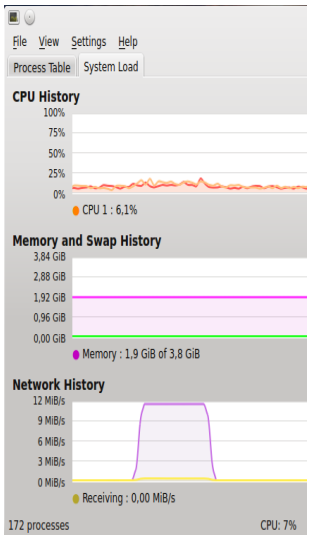
(a) RTT=10 ms Window size=20 Chunk size=2048 bytes



(b) RTT=10 ms Window size=20 Chunk size=4096 bytes



(c) RTT=10 ms Window size=100 Chunk size=2048 bytes



(d) RTT=10ms HTTP 1.0 (curl)

Figure 9.2: The effect of CCN network traffic on server CPU load. Note that HTTP does not cause any significant CPU load.

- **Server packet buffer overload**
  So much data is requested that some data is dropped at the server, which has to be re-requested by the client after a timeout. This effect is visible when the chunk size is 8192 and the window size is 400 in figure 9.1(b).

The experiment was repeated once, which confirmed the accuracy of the original data and the conclusions.

Chapter 10

Experiment 2: Downloading multiple smaller files

10.1 Motivation and setup

The previous experiment revealed important factors and configuration settings that constraint on the performance of CCN, but did not identify the optimal parameters for CCN. Specifically, the built-in chunk size limitation of 8192 bytes was not exceeded, but the experiment did show that higher chunk sizes generally lead to higher performance. Also, downloading singular large files is not the most relevant use case for content access on the internet. In particular, in DASH setups content is subdivided into smaller segments, and each is downloaded either serially or in parallel. The setup of this experiment is similar to the last one, with the important difference that this time instead of downloading one file of 100 MB, 50 files of 2 MB will be downloaded. The chosen file size is in line with high-resolution DASH segments. This way, the start and stop effects of CCN should be more visible.

In this experiment, we vary the chunk sizes between 1024 bytes

and 65000 bytes, with a factor of 4 difference between each step: **1024** bytes, **4096** bytes, **16384** bytes and **65000** bytes. To use chunk sizes above 8192 bytes, a built-in CCNx limit had to be raised. It was increased to 65535 bytes, out of a concern that the size might be stored in a 16-bit integer somewhere and consequently higher sizes might lead to hard-to-debug errors. Because this limit includes the CCN header, a chunk (content) size of 65000 bytes was used. Experiment 1 showed that a window size of 1 is suboptimal in every case, so in this experiment it was decided to go with a window size of 5 as minimum. An extra step between the window sizes of 20 and 100 was also added, to add some granularity. Furthermore, in experiment 1 the maximum window size of 400 did never result in an optimal download speed, so for this experiment the maximum was reduced to 200 simultaneous Interests. For the RTT, the same values as for the previous experiment were used, except that the case of 200ms RTT was also examined.

10.2 Code

The code for this experiment is similar to that of experiment 1, and is listed in appendix B.3.

10.3 Results

Figure 10.1 shows that, as expected, as RTT increases the download time increases as well. This is in line with the findings from experiment 1, where it was demonstrated that the download time is linearly related to RTT as long as no other limits are encountered.

This time the most prominent result was the anomaly at RTT=0ms, which is better explained using figure 10.3. When the RTT is very low and the chunk and window sizes are high, the CCN stack on the server side tries to push more bytes onto the interface than the interface can handle at that point, filling up the IP packet buffer, and causing an overflow which causes the server to drop packets.

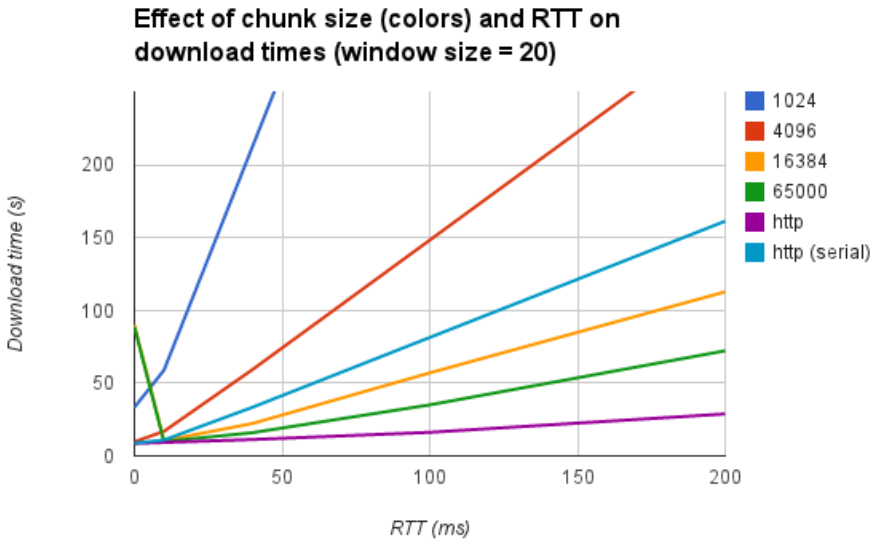


Figure 10.1: Download time versus RTT. Note that CCN can outperform HTTP in sequential downloading of content (when chunk size >= 16K and RTT >= 10ms). The line labeled http is using HTTP/1.1 pipelining to download all segments at once.

The client notices the chunks time out and re-sends the Interest for them after 4 seconds, causing delays. This is the same effect as was observed in experiment 1 for a window size of 400 and a chunk size of 8 KB at RTT=10ms. The effect diminishes at higher chunk sizes and higher window sizes because in such cases, the average link utilization remains optimal even though many pending interests time out. The time-outs decrease the effective window size, but at high chunk and window sizes, the effective window size remains big enough for a near-optimal data transfer.

In figure 10.2 it is shown that increasing window sizes beyond 20 does not give a real difference except with very small chunk sizes. And even with small chunk sizes, increasing the window size beyond 60 does not give any performance benefit. This is caused by the fact that the system is fully occupied by the CCNx stack, as is evidenced by figure 10.4. In this figure, the CPU load is plotted for a download with chunk size 1024 and a window size of respectively 5, 20 and 60 at an RTT of 10ms. It can be clearly seen that the CPU load is a limiting factor: with a window size of 20, the CPU load climbs to 80%. When we increase it even further, the CPU load reaches 100% which slows down the CCNx stack.

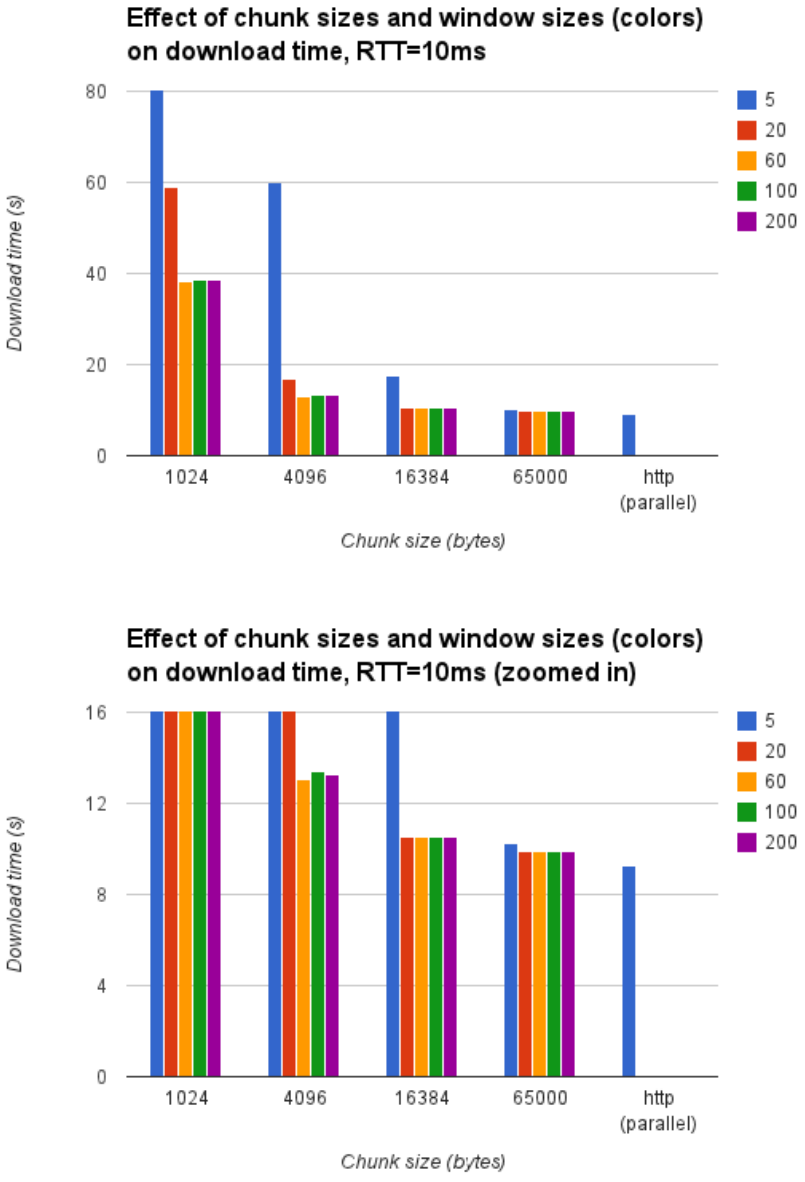


Figure 10.2: Download time decreases as chunk size increases. HTTP in this figure means pipelined HTTP 1.1

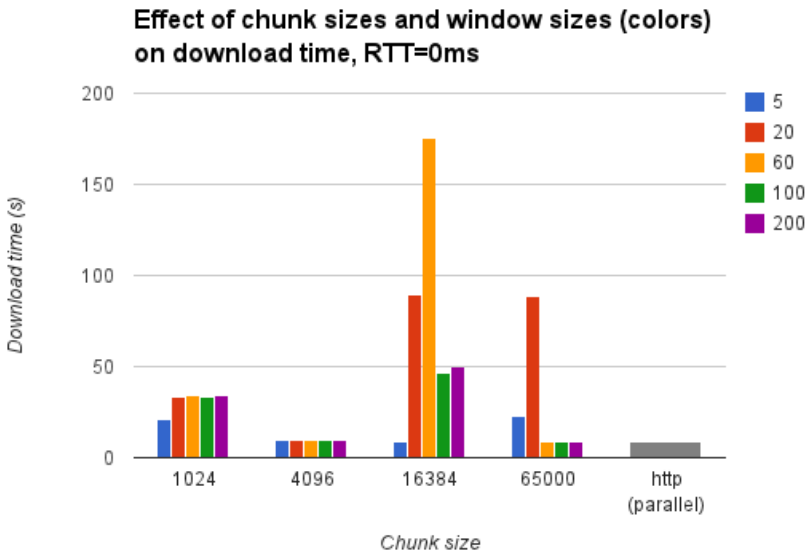


Figure 10.3: Anomaly at RTT=0ms, with chunk size=16-65K and window size=20-60 download time increases dramatically.

Figure 10.4: CPU and network bandwidth. From left to right: Window size 5 (end of download run), window size 20, window size 60. Note that CPU load increases with the window size, reaching 80% at window size 20 and reaching 100% somewhere between 20 and 60. This means that there is no gain in increasing the window size because of server CPU load.

Chapter 11

Experiment 3: DASH-like CCN download

11.1 Motivation and setup

The previous experiments showed various limitations of the CCNx stack and CCN in general, and also showed the optimal parameters for sequential download of files. However, they did not use a realistic DASH-like data set, and they downloaded each file (segment) in series instead of downloading some in parallel for CCN. The next experiment therefore looks at the potential benefits of downloading multiple segments in parallel, with real, variably sized DASH content. The same data set as was used in [LGP+13] and [LMR+13a] is used, in particular the 500kbit representation of the data set.. Each segment lasts for 2 seconds of playback time, leading to an average segment size of 125 KB or 1 Mbit (the actual average size was 124954 bytes with a standard deviation of 76942 bytes). This data set uses a chunk size of 4KB for the entire data set. To download these files in parallel, separate programs needed to be written for both CCN and HTTP 1.1 (see appendix B.1 and B.2). The CCN program was written against the CCNx API, more specifically the ccn_fetch API. The HTTP program was written using libcurl (version 7.35), with the pipelining option. To simulate a realistic network scenario where content is downloaded from a remote server, the

network throughput was reduced to 10Mbit/s.

For more thorough examinations of the resulting network traffic, Wireshark was used (version 1.10.6), in combination with the CCN Wireshark plugin. This plugin did not correctly show the CCN Data packets with unconventional chunk sizes, but served as a baseline to detect and filter CCN Interests.

The variables considered in this experiment are as follows:

- **Window size**
  By "window size" is meant the amount of pending Interests per segment, rather than in total. The total window size of the CCN stack is given by the window size times the amount of segments that are downloading simultaneously. The window size is varied between **1**, **2**, **3**, **4**, **6**, **8**, **12**, **16**.
- **Number of simultaneous segments ("Segment window")**
  By "Segment window" is meant the number of DASH segments downloading in parallel. The segment window is varied between **1**, **2**, **4**, **8**, **12** and **16**.
- **RTT**
  For the RTT the following values are used: **0ms**, **1ms**. **10ms**, **20ms**, **40ms** and **100ms**. The extra value 1ms was added to better examine the effects of low RTT on download speed, which has some curious properties as shown by the previous experiments.
- **Chunk size**
  The chunk size of 4KB is fixed by the use of the ITEC data set, and not varied in this experiment. The effect of varying this parameter has been sufficiently shown by the previous two experiments,

11.2 Code

The code for the program that downloads multiple segments in parallel with configurable window size and segment window is available

in appendix B.1. The code for the equivalent program in HTTP is in appendix B.2. The script that iterates over the various settings and calls these programs is in appendix B.4.

11.3 Results

Downloading many segments in parallel did indeed result in a better download time for small segments, with the effect rising proportionally with the RTT of the link. For links with negligible RTT, it can be shown that the segment window does not really affect the throughput a lot, although there is always some improvement from downloading two segments in parallel rather than only downloading one (see figure 11.1). This is because as a segment download starts and ends, the link utilization drops below the maximum, which is covered by having a second segment download during this interval.

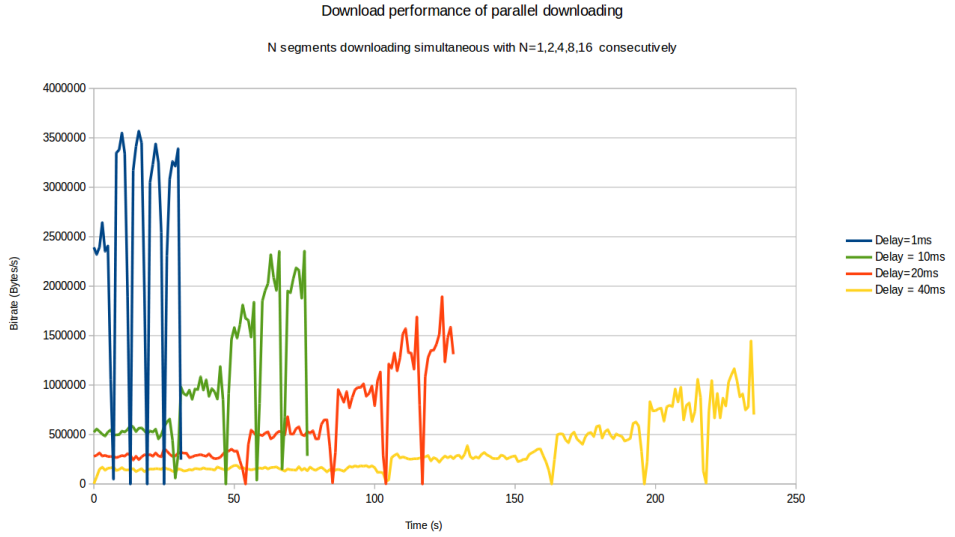


Figure 11.1: Download trace of 5 consecutive downloads with various RTT values (colors), each time increasing the segment window size. Note that for delay=1ms, the optimum is reached with 2-4 segments simultaneously, with a slight decrease for higher segment window sizes. The window size is 5 in this particular test.

A particular surprise in this experiment is that there is clearly an optimal segment window size for each combination of RTT and chunk window size. The server CPU is not a limiting factor in this case, but instead this phenomenon was probably caused by the

fact the high protocol overhead, both on-the-wire and in terms of computations necessary before sending out chunks, although this assertion was not tested. Other effects are into play. As can be seen in figure 11.2, a maximum download speed that can be reached for any given RTT was observed (in that figure, the limit is about 650 KB/s for an RTT of 40ms, for example). For other RTTs, the same effect can be observed, as can be evidenced from the table below.

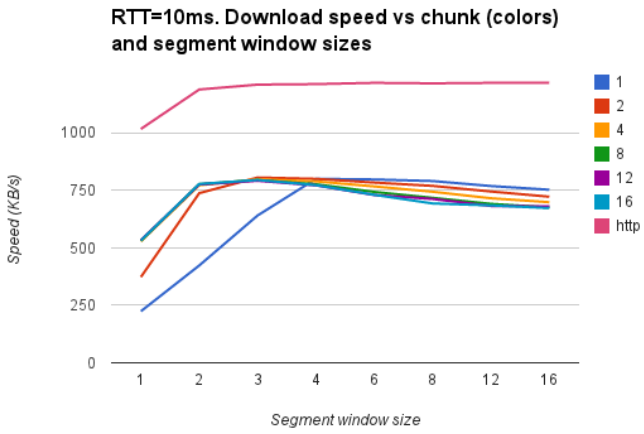
RTT	**Observed maximum download speed**
0	850 KB/s
1	840 KB/s
10	800 KB/s
20	750 KB/s
40	650 KB/s
100	400 KB/s

This is also visualized in figure 11.3. There is a linear relationship between RTT and download time. In experiment 1, it was shown that there was a relationship between RTT and download speed, but this limit also depended on the window size, which does not appear to be the case here for window sizes greater than about 4. Figure 11.2 also shows that download speed can be improved by either increasing the chunk window size or the segment window size, at least up to this limit.

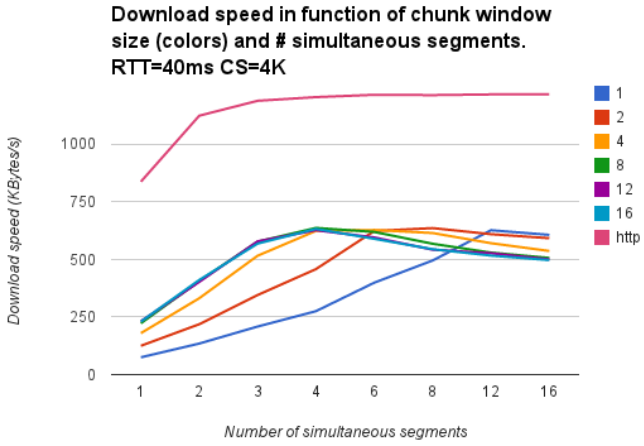
From the results, we can attempt to derive optimal parameter sets for each RTT. Since once of the core concepts of DASH is the ability to switch to a better (or worse) stream depending on changing network environments, it is best not to download too many segments at any one time, because that leads to more wasted packets and a slower reaction when switching to the different quality stream. Given this, we see that CCN's maximum performance can be obtained (among other configurations) by downloading 3 concurrent segments at a low RTT (1-20ms), rising linearly to 6 segments in the case of higher RTT (100ms), with a chunk window size of 8. Also, it is important to keep in mind the important of high chunk sizes: as shown by the previous experiment, higher chunk sizes perform better (see section 10).

Compared with [LMR+13b], the results of this thesis do show the same RTT problems of CCN relative to HTTP 1.1 (see figure

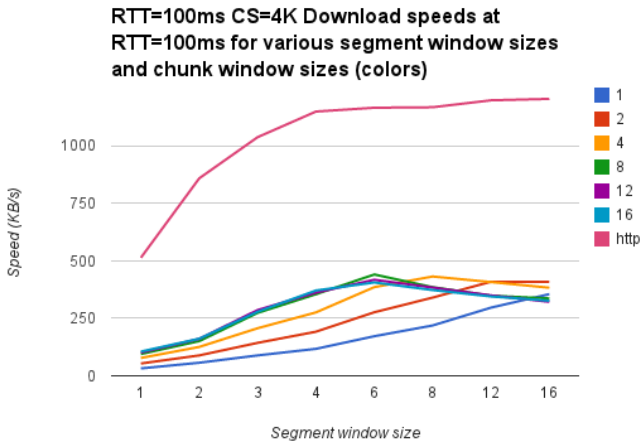
11.4). In the results of this experiment, there is a steeper drop-off for both HTTP and CCN, which can be attributed to lower segment sizes in this experiment (we use segment sizes of 125 KB on average ($500Kbit/s * 2s = 125KB$), whereas in the ITEC case, dynamic adaptation was probably used (although this is not explicitly confirmed in the source).



(a) RTT=10ms



(b) RTT=40ms



(c) RTT=100ms

Figure 11.2: Download speed with various parameters at various RTT values. Note the apparent ceiling each time for CCN, irrespective of the number of segments and the window size. HTTP in this case refers to HTTP/1.1 with pipelining, downloading the same amount of segments in parallel as in the CCN case. HTTP quickly climbs to near-optimal link utilization.

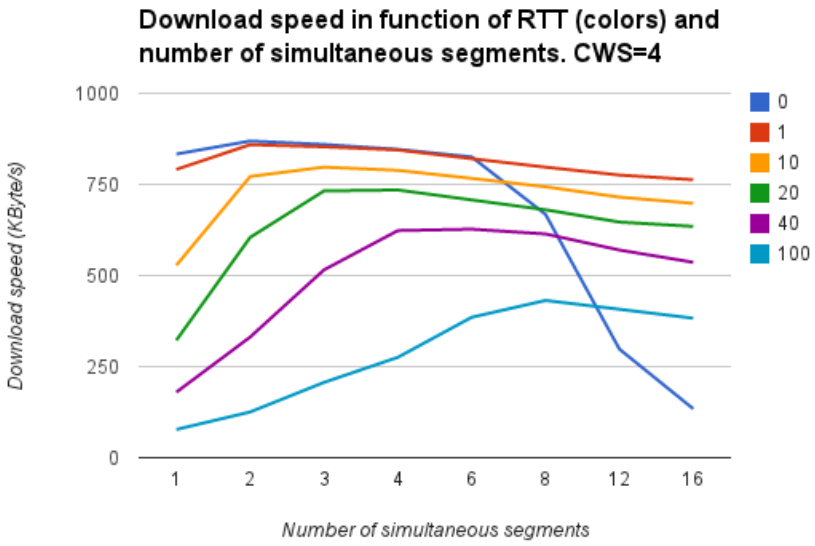
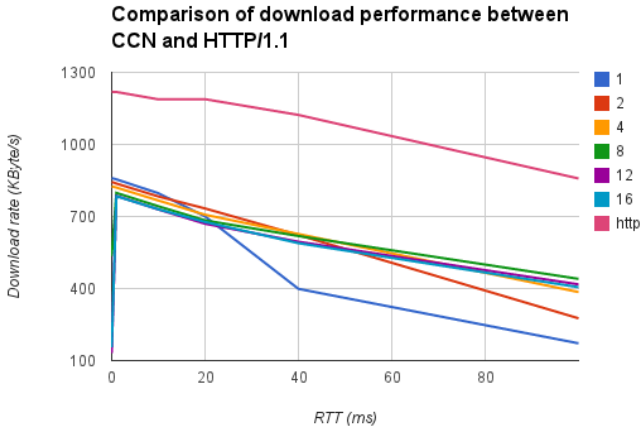


Figure 11.3: Download speed in function of RTT and number of simultaneous segments, showing the different ceilings for each RTT. The anomaly at RTT=0ms and high number of simultaneous segments is explained by the server packet buffer overload, discussed in experiment 1.



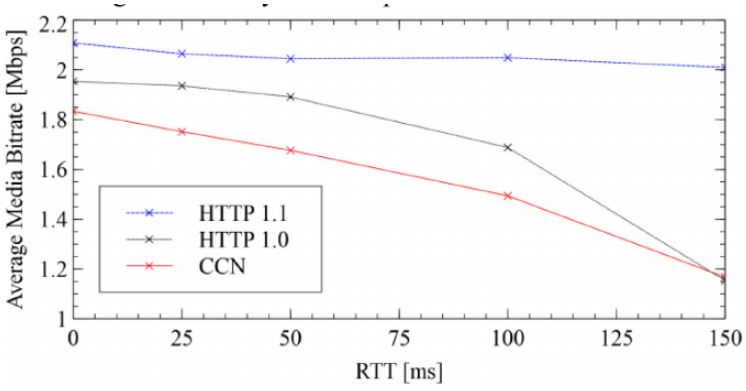


Figure 7: Average Media Throughput for different RTTs

Figure 11.4: A comparison of the results with those of [LMR+13b]. Our results (which are subdivided into the results for different window sizes,colors) are on top.

Chapter 12

Other experiments

12.1 Multi-source CCN

One of the initial goals was to investigate the potential for faster download in CCN by exploiting the fact that many hosts, especially mobile ones, can potentially be connected to the internet through multiple interfaces at once. One of the major features of CCN is that it is link agnostic, so it will simply switch to a different link if the old link becomes unavailable, or even if another link simply provides faster throughput than the interface currently being used. It does this by occasionally sending Interest packets out over other links as well as the one it is currently using. If it receives a reply on a different interface first, it is an indication that the other link may be faster and it can switch to this link. While this functionality is useful by itself, there is also the potential to significantly increase the download rate by fully utilizing both links at once. However, some research discovered that this is currently not supported by any available CCN implementation, and in particular would need some tricky modifications to the CCN strategy layer. The strategy layer decides where to forward an Interest to, and thus over which interface data will be received. The strategy layer would need to send out Interests over both interfaces for different chunks in direct proportion to the current link strength of both interfaces to achieve an efficient load balancing; however, accurate link perfor-

mance and reliability information is not always easy to obtain. In addition, whenever two links are available to a host, usually one will be far superior to the other, meaning that any potential benefits will be modest. This might change in the future with the increased adoption of 4G technologies, which can get as fast as Wi-Fi.

Chapter 13

Conclusions

CCN has the potential to be a contender for HTTP when it comes to video streaming, but does not fit neatly underneath the current DASH approach. For CCN, it is not a good idea to segment video files into small segments, because CCN already uses variably sized chunks. In particular, the benefits of parallelism level out in CCN at a certain level (determined by the RTT), so it is better to download one large file using a large window size than several small ones with high window sizes for each. Hence, CCN is not currently optimized for DASH-like content delivery (many small files).

Our results show that there are several limits imposed by the CCN stack on the download speed:

1. Link underutilization due to RTT and a low Interest window size/chunk size.
2. Server CPU overload with high number of outstanding Interests (more than 2000 per second on a dual-core Intel Core 2 Duo running at 2.4 GHz).
3. Server packet buffer overload: the server tries to send more data back to the client than the link allows, causing it to fail to send some data, which is later re-requested by the client.
4. The observed ceiling reached when issuing many requests in

parallel for chunks of different files.

That said, with careful tuning it is usually possible to approach or even exceed the performance of HTTP when downloading single large files, and to approach the link speed at low RTT and high chunk and window sizes. However, large chunk sizes increase the chance that chunks will be lost due to packet drops when clashing with other protocols such as TCP over highly utilized links, so network conditions must be considered in every situation to find the optimal balance. Large chunk sizes are to be preferred because they minimize the impact of CCN's very large header (compared with TCP/IP headers), and to diminish the processing overhead associated with a high number of outstanding Interests. Increasing Window sizes will not have a big impact beyond a certain point, because either the link speed or the CPU limit will be reached. At very low network delay (<= 1ms) care must be taken not to overwhelm the link. In such cases, a chunk size of 4 KB performs better than higher chunk sizes.

Given the above conditions and limitations, the optimal settings for the playback of DASH content were determined in function of the RTT of the link, which is accurately measurable. Our results show that at high RTT, high window sizes and higher segment window sizes are appropriate; on the other hand, for low RTT, low levels yield a slightly higher download rate.

Because HTTP 1.1 presently outperforms CCN in every measurement regarding DASH, additional incentives will be necessary for anyone to switch to CCN, such as its automatic caching behavior. Whether this offers a compelling reason to switch from using CDNs remains to be seen.

The central question this thesis tries to answer is whether CCN can be a viable replacement of TCP/IP (see section 1.1.3). Our testing has revealed that CCN has different performance characteristics than TCP/IP, and that, while implementing a use case like dynamic video streaming over CCN is not a big problem, CCN may not be suited to all use cases. Certainly, as we have established, CCN cannot beat TCP in raw performance, and needs another reason (such as its built-in caching) to give it an edge.

Regarding the future uptake of CCN, it's important to note that CCN's two main advantages over TCP/IP from a user's perspective (addressing content by name and ubiquitous caching) rely on network effects (and network investments) to truly become effective:

- Addressing content by name is useful only when the content you search for can be found within the CCN network
- Caching only works when there are intermediate hosts willing (and capable) to store it, and only with popular content (i.e. video)

The current CCNx stack still has many deficiencies, some of which are inherent to CCN (such as the link underutilization problem). Others are subject to improvement in future implementations, such as the CPU load problems with a high number of pending Interests, and load balancing.

In my personal opinion, while I believe strongly in what CCN sets out to achieve, it tries to do too much. Addressing content by name is a good idea, but could be achieved using a DNS-like protocol on top of IP, rather than necessitating an overhaul of the network layer protocols. Similarly, it is good to build encryption into the network layer, but these are easier to achieve using dedicated initiatives (such as IPsec), rather than lumping it together with the content centric approach. Trying to fix everything at once means that CCN is not optimal for any one use case (e.g. the computational burden and header overhead decrease performance when simply disseminating content that does not benefit from the encryption or authenticity guarantees). Trying to do everything at once decreases the appeal to anyone who is interested in only one aspect of CCN, makes implementing and optimizing the CCN stack difficult (or impossible where two design goals conflict), leaves the door open to abuse of the system (such as using the computationally complex signature verification to execute a DoS attack on a server which is difficult to trace back to the source), and constrains applications to a strict model, necessitating workarounds such as pushing content as part of an Interest packet for push networking. All this goes against the one redeeming feature of IP, which is its simplicity that has over the past decades allowed anyone to build complicated systems on top of it which were never foreseen by the original authors

of IP. In the long term, such flexibility is the thing that ultimately matters most in networking.

Bibliography

[ADD+08] Bengt Ahlgren, Matteo D'Ambrosio, Christian Dannewitz, Marco Marchisio, Ian Marsh, Börje Ohlman, Kostas Pentikousis, René Rembarz, Ove Strandberg, and Vinicio Vercellone. Design considerations for a network of information. In *Proceedings of the 2008 ACM CoNEXT Conference*, page 66. ACM, ACM, December 2008.

[AVS] Adaptive video streaming over icn. Internet draft: draft-irtf-icnrg-videostreaming-00, expires Sept 10th, 2014. Accessed: 2014-08-31.

[bel] Streaming quality of tv everywhere. `http://support.en.belgacom.be/app/answers/detail/a_id/15350/~/streaming-quality-of-tv-everywhere`. Accessed: 2014-08-31.

[BGP] Border gateway protocol. `http://en.wikipedia.org/wiki/BGP`. Accessed: 2014-08-31.

[CAFR12] Jiachen Chen, Mayutan Arumaithurai, Xiaoming Fu, and K.K. Ramakrishnan. Coexist: a hybrid approach for content oriented publish/subscribe systems. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, ICN 2012, pages 31–36, Helsinki, Finland, 2012. ACM.

[CAJ+11] Jiachen Chen, Mayutan Arumaithurai, Lei Jiao, Xiaoming Fu, and K.K. Ramakrishnan. Copss: An efficient content oriented publish/subscribe system. Technical report, Institute of Computer Science, University of Goettingen, Germany and AT&T Labs Research, Florham Park, NJ, U.S.A., 2011.

[CCN] Ccnx. http://www.ccnx.org/. Accessed: 2014-08-31.

[CG00a] David R. Cheriton and Mark Gritter. Triad: A scalable deployable nat-based internet architecture. Technical report, Stanford University, 2000.

[CG00b] D.R. Cheriton and M. Gritter. Triad: A new next-generation internet architecture. http://www-dsg.stanford.edu/triad/triad.ps.gz, 2000. Accessed: 2014-08-31.

[EFGK03] Patrick Th. Eugster, Pascal A. Felber, Rachid Guerraoui, and Anne-Marie Kermarrec. The many faces of publish/subscribe. *ACM Computing Surveys*, 35(2):114–131, June 2003.

[FSLL+06] Bryan Ford, Jacob Strauss, Chris Lesniewski-Laas, Sean Rhea, Frans Kaashoek, and Robert Morris. Persistent personal names for globally connected mobile devices. In *Proceedings of OSDI 2006*, pages 233–248, November 2006.

[GC01] Mark Gritter and David R. Cheriton. An architecture for content routing support in the internet. In *Proceedings of the 3rd USENIX Symposium on Internet Technologies and Systems*, volume 1 of *USITS 2001*, pages 4–4, Berkeley, CA, USA, 2001.

[Gri] Mark Gritter. The triad content layer. http://www-dsg.stanford.edu/slides/triad-content-netseminar/. Accessed: 2014-08-31.

[HG04] Mark Handley and Adam Greenhalgh. Steps towards a dos-resistant internet architecture. In *Proceedings of the ACM SIGCOMM workshop on Future directions in network architecture*, FDNA 2004, pages 49–56, Portland, Oregon, USA, 2004. ACM.

[Int] Internetworking: communicating protocols and basic tcp/ip. http://www.highteck.net/EN/Basic/Internetworking.html. Accessed: 2014-08-31.

[Jac06] Van Jacobson. A new way to look at networking. http://video.google.com/videoplay?docid=-6972678839686672840, August 2006. Accessed: 2014-08-31.

[JP09] Van Jacobson and Craig Partridge. A conversation with van jacobson. *ACM Queue*, 7(1):8–16, January 2009. Available at http://queue.acm.org/detail.cfm?id=1508215.

[JSB+09] Van Jacobson, Diana K Smetters, Nicholas H Briggs, Michael F Plass, Paul Stewart, James D Thornton, and Rebecca L Braynard. Voice over content-centric networks. In *Proceedings of the 2009 workshop on Re-architecting the internet*, pages 1–6, Rome, Italy, December 2009. ACM.

[JST+09] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 1–12. ACM, December 2009.

[KCC+07] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. In *Proceedings of the 2007 conference on Applications, technologies, architectures, and protocols for computer communications*, SIGCOMM 2007, pages 181–192, Kyoto, Japan, 2007. ACM.

[Lau10] Tobias Lauinger. Security & scalability of content-centric networking. Master's thesis, Technische Universität Darmstadt, 2010.

[LGP+13] Yaning Liu, Joost Geurts, Jean-Charles Point, Stefan Lederer, Benjamin Rainer, Christopher Müller, Christian Timmerer, and Hermann Hellwagner. Dynamic adaptive streaming over ccn: a caching and overhead analysis. In *Communications (ICC), 2013 IEEE International Conference on*, pages 3629–3633. IEEE, 2013.

[Lin06] D.I. Manfred Lindner. Ip technology basics. Technical report, Vienna University of Technology, 2006. https://www.ict.tuwien.ac.at/lva/384.081/infobase/L30-IP_Technology_Basics_v4-6.pdf.

[lLZ+12] Haibo li, Yang Li, Zhijun Zhao, Tao Lin, Hui Tang, and Song Ci. A sip-based real-time traffic mobility support scheme in named data networking. *Journal of Networks*, 7(6):918–925, June

2012. https://academypublisher.com/~academz3/ojs/index.php/jnw/article/view/jnw0706918925.

[LMR+13a] Stefan Lederer, Christopher Mueller, Benjamin Rainer, Christian Timmerer, and Hermann Hellwagner. Adaptive streaming over content centric networks in mobile networks using multiple links. In *Communications Workshops (ICC), 2013 IEEE International Conference on*, pages 677–681. IEEE, 2013.

[LMR+13b] Stefan Lederer, Christopher Müller, Benjamin Rainer Rainer, Christian Timmerer, and Hermann Hellwagner. An experimental analysis of dynamic adaptive streaming over http in content centric networks. In *Proceedings of the IEEE International Conference on Multimedia and Expo 2013*, pages 1–6, San Jose, USA, July 2013. IEEE.

[LRH10] Uichin Lee, Ivica Rimac, and Volker Hilt. Greening the internet with content-centric networking. In *Proceedings of the 1st International Conference on Energy-Efficient Computing and Networking*, e-Energy 2010, pages 179–182, Passau, Germany, 2010. ACM.

[MA11] Ahmed Mansy and Mostafa Ammar. Analysis of adaptive streaming for hybrid cdn/p2p live video systems. In *Proceedings of the 2011 19th IEEE International Conference on Network Protocols*, ICNP '11, pages 276–285, Washington, DC, USA, 2011. IEEE Computer Society.

[MPZ10a] Michael Meisel, Vasileios Pappas, and Lixia Zhang. Ad hoc networking via named data. In *Proceedings of the fifth ACM international workshop on Mobility in the evolving internet architecture*, MobiArch 2010, pages 3–8, Chicago, Illinois, USA, 2010. ACM.

[MPZ10b] Michael Meisel, Vasileios Pappas, and Lixia Zhang. Listen first, broadcast later: Topology-agnostic forwarding under high dynamics. In *Annual Conference of International Technology Alliance in Network and Information Science, London, UK (September 2010)*, 2010.

[ndn] ndnsim documentation – overall ndnsim documentation. http://ndnsim.net/. Accessed: 2014-08-31.

[Nig08] Eddy Nigg. Untrusted certificates. https://blog.startcom.org/?p=145, December 2008. Accessed: 2014-08-31.

[NSG+12] Erik Nordström, David Shue, Prem Gopalan, Matvey Arye, Steven Ko, Jennifer Rexford, and Michael J. Freedmano. Serval: An end-host stack for service-centric networking. In *Proceedings of USENIX NSDI*, pages 85–98, April 2012. http://www.serval-arch.org/docs/serval-tr-Oct11.pdf.

[PAR] PARC. Named data networking. http://named-data.org/. Accessed: 2014-08-31.

[RR11] Dario Rossi and Giuseppe Rossini. Caching performance of content centric networks under multi-path routing (and more). *Relatório técnico, Telecom ParisTech*, 2011.

[RSA78] R. L. Rivest, A. Shamir, and L. Adleman. A method for obtaining digital signatures and public-key cryptosystems. *Commun. ACM*, 21(2):120–126, February 1978.

[San] Global internet phenomena report 1h2014. http://www.sandvine.com. Accessed: 2014-08-31.

[SDC+12] Stefano Salsano, Andrea Detti, Matteo Cancellieri, Matteo Pomposini, and Nicola Blefari-Melazzi. Transport-layer issues in information centric networks. In *Proceedings of the second edition of the ICN workshop on Information-centric networking*, pages 19–24. ACM, 2012.

[Sig] Digital signature. http://en.wikipedia.org/wiki/Digital_signature. Accessed: 2014-08-31.

[SJ09] Diana Smetters and Van Jacobson. Securing network content. Technical report, Palo Alto Research Center, 2009.

[SRK+03] Scott Shenker, Sylvia Ratnasamy, Brad Karp, Ramesh Govindan, and Deborah Estrin. Data-centric storage in sensornets. In *ACM SIGCOMM Computer Communications Review*, volume 33, pages 137–142. ACM, Januari 2003.

[tel13] Yelo tv een 360Âř beleving - medianet vlaanderen. Technical report, Telenet, September 2013. http://www.medianetvlaanderen.be/content/user/File/130905%20StreamingIII/Telenet%20Yelo%20TV-%20een%20360%C2%B0beleving.pdf.

[VoI] A comparison of voip software. http://en.wikipedia.org/wiki/Comparison_of_VoIP_software. Accessed: 2014-08-31.

[WWK+12] Lucas Wang, Ryuji Wakikawa, Romain Kuntz, Rama Vuyyuru, and Lixia Zhang. Data naming in vehicle-to-vehicle communications. In *IEEE INFOCOM 2012 Workshop on Emerging Design Choices in Name-Oriented Networking*, pages 328–333. IEEE, 2012.

[YFX12] Chunfeng Yao, Lingyuan Fan, and Yanping Xiang. long-term interest for realtime applications in the named data network. *AsiaFI'12, August 20-24, 2012, Kyoto University, Kyoto, Japan*, 2012.

[YFYX12] Chunfeng Yao, Lingyuan Fan, Zhefeng Yan, and Yanping Xiang. Long-term interest for realtime applications in the named data network. In *Proceedings of AsiaFI 2012*, Kyoto, Japan, August 2012. Asia Future Internet Forum, ACM. http://asiafi.net/meeting/2012/summerschool/submissions/WS/AsiaFI2012-ws-01.pdf.

[ZN11] Yuncheng Zhu and Akihiro Nakao. Content-oriented transport protocol. In *Proceedings of the 7th Asian Internet Engineering Conference*, AINTEC '11, pages 104–111, New York, NY, USA, 2011. ACM.

Appendix A

Figures

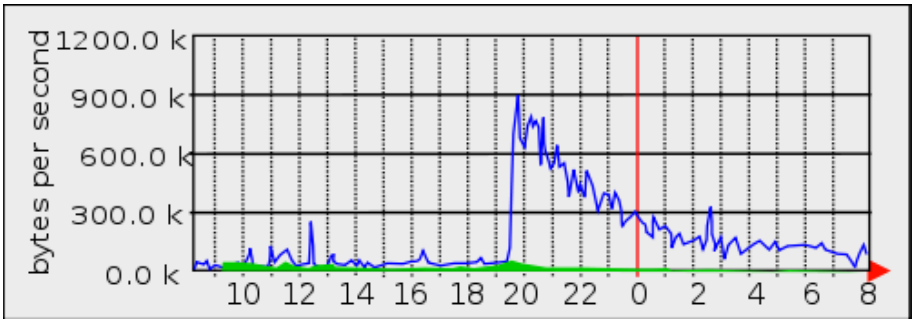


Figure A.1: An illustration of the Slashdot effect: The graph shows the increase in web traffic to a small server after content on it has been posted to a popular website like Slashdot. Many small websites are unable to deal with this. Source: http://en.wikipedia.org/wiki/File:SlashdotEffectGraph.svg, retrieved September 16, 2013

Appendix B

Code

B.1 DASH over CCN test application

Listing B.1: Asynchronous download application for CCN: Downloads multiple DASH segments simultaneously

```
#include <stdlib.h>
#include <string.h>
#include <stdio.h>
#include <unistd.h>
#include <sys/time.h>
#include <stdbool.h>
#include <ccn/ccn.h>
#include <ccn/fetch.h>

//The highest-numbered segment to download
#define MAX_SEGMENT 299
//The lowest-numbered segment to download
#define MIN_SEGMENT 1
//The maximum size of a chunk in this experiment
//Used as a buffer size.
#define MAX_CHUNK_SIZE 65000

//Current chunk window size
static unsigned chunkWindowSize;
//Data attached to a single request, used to identify
//the segment as well as the download start time
//(to measure download times for individual segments).
typedef struct {
  unsigned num;
```

```
  struct timeval start;
} segmentData;

//Create a new ccn_fetch_stream object,
//used to download the next segment ('top' segment)
static bool request_next(struct ccn_fetch *handle, unsigned
    *top)
{
  struct ccn_fetch_stream *fs;
  struct ccn_charbuf *name;
  char uri[256];
  segmentData *segdata;
  //Stop downloading once MAX_SEGMENT is reached.
  if (*top == MAX_SEGMENT)
    return false;
  segdata = malloc(sizeof(segmentData));
  name = ccn_charbuf_create();
  //Memory allocation failure
  if (segdata == NULL || name == NULL)
    return false;
  //Create the name
  snprintf(uri,sizeof(uri),"ccnx:/itec1/dash/bunny/
     bunny_2s_500kbit/bunny_2s%d.m4s",*top+1);
  ccn_name_from_uri(name,uri);
  //Create the ccn_fetch_stream
  //representing this chunk download
  //Can fail when creating lots in a row
  //hence the while loop
  while ((fs=ccn_fetch_open(
            handle,//master handle
            name,   //CCNx-formatted name
            uri,    //unformatted name
            NULL,   //Interest template
            chunkWindowSize,
            CCN_V_HIGHEST,//Use versioning, prefer highest
                version
            1 //Can assume each chunk has the same size
           )) == NULL)
    usleep(1);
  ccn_charbuf_destroy(&name);
  ++(*top);
  //Fill in the segment data
  segdata->num = *top;
  gettimeofday(&segdata->start,NULL);
  ccn_fetch_set_context(fs,segdata);
  return true;
}

// Main download function.
```

```
static void download_run(unsigned nsegs)
{
  //Create the master handle
  struct ccn_fetch *handle = ccn_fetch_new(NULL);
  unsigned top = 0;
  char buf[MAX_CHUNK_SIZE];
  unsigned count=0;
  struct timeval start,end,interval;
  if (!handle)
  {
    perror("ccn_fetch_new failed");
    return;
  }
  //Get the current time, before starting
  gettimeofday(&start,NULL);
  //Queue nsegs simultaneous downloads
  for (count=0;count<nsegs && request_next(handle,&top);++
      count)
    usleep(1);
  while ( count != 0 )
  {
    //Check if any handles have changed state
    if (ccn_fetch_poll(handle)!=0)
    {
      struct ccn_fetch_stream *s;
      //Iterate over all active handles
      //and check them individually
      for ( s = ccn_fetch_next(handle,NULL);
        s != NULL;
        s = ccn_fetch_next(handle,s))
      {
        intmax_t res = ccn_fetch_read(s,buf,sizeof(buf));
        if (res > 0)
        {
        //In a real application, this is the point where
        //you would do something with the data
        //(write it to file, put it into a display buffer
            ,...)
          continue;
        }
        switch(res)
        {
          // If a timeout occurs, retry
          case CCN_FETCH_READ_TIMEOUT:
            puts("Timeout!!!");
            ccn_reset_timeout(s);
            break;
          //Nothing was read -- ignore
          case CCN_FETCH_READ_ZERO:
```

```
            case CCN_FETCH_READ_NONE:
              break;
            //The download has completed
            case CCN_FETCH_READ_END:
            {
              //1) Find out how long it took to download
              //This data is not currently used
              segmentData *segdata = ccn_fetch_get_context(s)
                  ;
              struct timeval s_end,s_int;
              gettimeofday(&s_end,NULL);
              timersub(&s_end,&segdata->start,&s_int);
              double dur = s_int.tv_sec + ((double)s_int.
                  tv_usec)/1000000.;
              //TODO: use dur to calculate an average segment
                   download time
              //2) Clean up
              ccn_fetch_close(s);
              --count;
              free(segdata);
              //3) Start downloading the next file
              if (request_next(handle,&top))
                ++count;
              break;
            }
            default:
              printf("Error: unknown return from
                  ccn_fetch_read: %ld\n",res);
          }
        }
      }
      //Don't cook the CPU/freeze the system
      usleep(10);
    }
    //Get the current time at the end of the stream download,
        and calculate the duration.
    gettimeofday(&end,NULL);
    timersub(&end,&start,&interval);
    double total = interval.tv_sec + ((double)interval.
        tv_usec)/1000000;
    printf("Total download took %lf seconds\n",total);
}

int main(int argc, char **argv)
{
    unsigned long sws,cws;
    if (argc < 3 || (sws=strtoul(argv[1],NULL,0))*(cws=
        strtoul(argv[2],NULL,0)) == 0)
```

```
    return fprintf(stderr,"Usage: %s segment_window_size
        chunk_window_size\n",argv[0]),EXIT_FAILURE;
  //Remove any locally cached chunks first,
  // otherwise that would be cheating ;-)
  (void) system("ccnrm /itec1");
  chunkWindowSize = (unsigned)cws;
  //Start the download
  download_run((unsigned)sws);

  return EXIT_SUCCESS;
}
```

B.2 DASH over HTTP test application

Listing B.2: Asynchronous download application for HTTP: Downloads multiple DASH segments simultaneously, simulating a real DASH player but with adjustable number of parallel segment downloads

```
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <curl/curl.h>
#include <unistd.h>

#define MIN_SEGMENT 1
#define MAX_SEGMENT 299
#define MIN_PIPELINE_REQ 1
#define MAX_PIPELINE_REQ (MAX_SEGMENT - MIN_SEGMENT)

//Number of simultaneous (pipelined) requests
static unsigned int pipeline_req;

//Number of the next segment to download
static int top = MIN_SEGMENT;
//Output file (needed by cURL)
static FILE* bitbucket = NULL;
//Function to obtain the next segment URI to use
static char* getnexturi(void)
{
  static char uri[256];
  if (top > MAX_SEGMENT) return NULL;
  snprintf(uri,sizeof(uri),"http://192.168.1.181/bunny/
      bunny_2s_500kbit/bunny_2s%d.m4s",top++);
  return uri;
}
//Queue a file for downloading with curl_multi_perform
```

```
static CURL* addDownload(CURLM *master)
{
  CURL* handle;
  //Get the URI to download
  char *uri = getnexturi();
  if (uri == NULL) {
    puts("All segments added.");
    return NULL;
  }
  //Create a cURL handle for it
  handle  = curl_easy_init();
  if (handle == NULL)
    puts("NULL handle received!");
  //Configure the handle and
  //add it to the master ('multi') handle
  if (  handle != NULL &&
        curl_easy_setopt(handle,CURLOPT_URL,uri) ==
            CURLE_OK &&
        curl_easy_setopt(handle,CURLOPT_WRITEDATA,bitbucket
            ) == CURLE_OK &&
        curl_multi_add_handle(master,handle) == CURLM_OK)
                return handle;
  puts("Adding a handle failed!");
  return NULL;
}

int main(int argc, char **argv)
{
  unsigned i=0;
  CURLM *m_curl;
  CURLMcode res;
  struct timeval start,end,interval;
  int count=0;
  if(argc <= 1)
    return puts("Insufficient arguments"), 1;

  pipeline_req = atol(argv[1]);
  if(pipeline_req > MAX_PIPELINE_REQ || pipeline_req <
     MIN_PIPELINE_REQ)
    return puts("Invalid Pipeline Request: argument must be
        between 1 and 298"), 2;

  //CURL needs an output file, so let's use /dev/null
  //This special file throws away anything written to it
  bitbucket = fopen("/dev/null","w");

  //Set up curl, and enable pipelining
  m_curl = curl_multi_init();
  curl_multi_setopt(m_curl, CURLMOPT_PIPELINING, 1L);
```

```
//Add pipeline_req sub-handles to cURL, to download that
   many files in parallel.
for(i=0; i<pipeline_req; i++)
  (void) addDownload(m_curl);

//Before really starting the download, get the current
   time
gettimeofday(&start,NULL);

do
{
  // Perform the download,
  //   stopping when one or more of the sub-handles
     change state
  res = curl_multi_perform(m_curl, &count);

  // Check the message to see if a download has finished
  if (res == CURLM_OK && count < (int)pipeline_req)
  {
    CURLMsg *msg;
    int i;

    //Remove download handles for completed downloads
    while((msg=curl_multi_info_read(m_curl,&i))!=NULL)
    {
      char *url = NULL;
      if (msg->msg != CURLMSG_DONE)
        continue;
      curl_multi_remove_handle(m_curl, msg->easy_handle);
      curl_easy_getinfo(msg->easy_handle,
          CURLINFO_EFFECTIVE_URL,&url);
      printf("Segment '%s' has been downloaded! top=%d
          count=%d\n",url,top,count);
      curl_easy_cleanup(msg->easy_handle);
    }
    //Start downloading new files, until the counter
       reaches MAX_SEGMENT
    for (;count<(int)pipeline_req && top <= MAX_SEGMENT
       ;++count)
      (void) addDownload(m_curl);
  }
  //Do not overheat the CPU by sleeping a little (0.1ms)
  //(This does not slow down the downloads --
  // they happen asynchronously).
  usleep(100);
  // Stop when the number of active handles reaches 0 or
     there is an error
} while(count > 0 && res==CURLM_OK);
```

```
  if (res != CURLM_OK)
  {
    fprintf(stderr,"An error occurred: %s\n",
       curl_multi_strerror(res));
  }
  //Compare start and end time of the total download to get
      the duration
  gettimeofday(&end,NULL);
  timersub(&end,&start,&interval);
  printf("Total elapsed time: %lf seconds\n",interval.
     tv_sec + ((double)interval.tv_usec)/1000000.);

  //Clean up
  curl_multi_cleanup(m_curl);
  fclose(bitbucket);

  return 0;
}
```

B.3 Experiment 2 execution script

Listing B.3: Experiment 2: Sending 50 chunks of 2MB each, with different window sizes, chunk sizes and RTT

```
#!/bin/bash
# Author: Frederik Van Bogaert
# Parameters: chunk size, RTT and CCN window size (interest
    pipelining)

OTHER_HOST=192.168.1.181
CCN_NAME=ccnx:/test2/file
LOGFILE=smallchunks.log
REMOTEDIR=~/data/test/
FILESIZE=20 # x 100 000 bytes
ITERATIONS=50

rm $LOGFILE

echo "Creating files on $OTHER_HOST"
ssh $OTHER_HOST mkdir -p $REMOTEDIR
#Make 50 files of 2 MB. These are our "segments"
ssh $OTHER_HOST "for ((i=0;\$i<$ITERATIONS;i=\$i+1)); do dd
    if=/dev/urandom of=$REMOTEDIR/\$i bs=100000 count=
   $FILESIZE; done"
#Make sure the local routing table is correct
ccndc add $CCN_NAME udp $OTHER_HOST
```

```
for rtt in 0 10 40 100 200;
do
  echo "Setting RTT to ${rtt}ms"
  #Set the "queuing discipline" for interface eth0 to "
     Network Emulation"
  #   with artificial delay of rtt ms and capped to 100
     MBit/s
  ssh $OTHER_HOST tc qdisc replace dev eth0 root netem
     delay ${rtt}ms rate 100Mbit
  for chunk_size in 1024 4096 16384 65000;
  do
    for window_size in 5 20 60 100 200;
    do
      echo "Preparing to send data with sz=$chunk_size ws=
         $window_size rtt=$rtt"
      #Remove cached copies from both hosts
      #This ensures the segments will be re-chunked and re-
         sent every time
      ssh $OTHER_HOST ccnrm $CCN_NAME
      ccnrm $CCN_NAME
      #Send the chunks on the other host. Do this in the
         background.
      ssh $OTHER_HOST "for ((it=0;\$it<$ITERATIONS;it=\$it
         +1)); do ccnsendchunks -b $chunk_size $CCN_NAME/\
         $it <$REMOTEDIR/\$it; done &" &
      #We need to wait a bit to be sure the content is
         available from the other host
      sleep 5
      RESULT=0
      for ((it=0; $it<$ITERATIONS; it=$it+1));
      do
        #Download the chunk and store the debug output
           including transfer time
        RESULTLINE=$(ccncatchunks2 -d -p $window_size
           $CCN_NAME/$it 2>&1)
        if [ -z "$RESULTLINE" ]; then exit; fi
        #Filter the result to obtain just the time taken by
            the download.
        R=$(echo $RESULTLINE | egrep -o "[0-9.]+ seconds" |
            cut -d' ' -f1)
        #Floating-point addition is not supported by BASH
           -> use python (or perl,awk,...) instead
        RESULT=$(python -c "print $RESULT + $R")
      done
      #Write the parameters + result to a log file (CSV
         format)
      echo "$rtt;$chunk_size;$window_size;$RESULT" >>
         $LOGFILE
      #Terminate the server-side process
```

```
      pkill ssh
      echo "Done in $RESULT seconds"
    done
  done
done
```

B.4 Experiment 3 execution script

Listing B.4: Experiment 3: Download a DASH stream using CCN and HTTP, with various parameters

```
#!/bin/bash
#Author: Frederik Van Bogaert
#Prerequisite: The DASH CCN repository is initialized on
    the other host

OTHER_HOST=192.168.1.181
OUTFILE=experiment3.csv
#remove the output file if it already exists
[ -f "$OUTFILE" ] && rm $OUTFILE

for rtt in 0 1 10 20 40 100;
do
  #Set the RTT on OTHER_HOST, and limit the rate to 10Mbit/
      s
  ssh $OTHER_HOST "tc qdisc replace dev eth0 root netem
      delay ${rtt}ms rate 10Mbit";
  #SWS = Segment Window Size = # of simultaneous segments
  for sws in 1 2 3 4 6 8 12 16;
  do
    #Try HTTP first
    RESULT=$(./httpdl $sws | egrep -o "[0-9.] seconds" |
        awk '{print $1}')
    #We're not using the chunk size field for HTTP
    # -> put the special value 'http' there
    echo "$rtt;$sws;http;$RESULT" >> $OUTFILE
    #CWS = Chunk Window Size = Number of chunks to fetch
        simultaneously
    #       per segment
    for cws in 1 2 4 8 12 16;
    do
      #Download the stream using given SWS and CWS
      #(see dlasync.c), store output in $RESULT
      RESULT=$(./dlasync $sws $cws|egrep -o "[0-9.]+")
      echo "$rtt;$sws;$cws;$RESULT" >> $OUTFILE
    done
```

```
    done
done
```

Auteursrechtelijke overeenkomst

Ik/wij verlenen het wereldwijde auteursrecht voor de ingediende eindverhandeling:
Content-Centric Networking

Richting: **master in de informatica-multimedia**
Jaar: **2014**

in alle mogelijke mediaformaten, - bestaande en in de toekomst te ontwikkelen - , aan de Universiteit Hasselt.

Niet tegenstaand deze toekenning van het auteursrecht aan de Universiteit Hasselt behoud ik als auteur het recht om de eindverhandeling, - in zijn geheel of gedeeltelijk -, vrij te reproduceren, (her)publiceren of distribueren zonder de toelating te moeten verkrijgen van de Universiteit Hasselt.

Ik bevestig dat de eindverhandeling mijn origineel werk is, en dat ik het recht heb om de rechten te verlenen die in deze overeenkomst worden beschreven. Ik verklaar tevens dat de eindverhandeling, naar mijn weten, het auteursrecht van anderen niet overtreedt.

Ik verklaar tevens dat ik voor het materiaal in de eindverhandeling dat beschermd wordt door het auteursrecht, de nodige toelatingen heb verkregen zodat ik deze ook aan de Universiteit Hasselt kan overdragen en dat dit duidelijk in de tekst en inhoud van de eindverhandeling werd genotificeerd.

Universiteit Hasselt zal mij als auteur(s) van de eindverhandeling identificeren en zal geen wijzigingen aanbrengen aan de eindverhandeling, uitgezonderd deze toegelaten door deze overeenkomst.

Voor akkoord,

Van Bogaert, Frederik

Datum: **8/09/2014**