Understanding Content Dissemination using Content Centric Networking

Distributed Systems and Networking Project Report

António Rodrigues (up200400437@fe.up.pt)

June 30, 2014

Contents

Chapter 1

Introduction

Information-Centric Networking (ICN) is a field of research which advocates for a new communication paradigm for the Internet: the departure from its host-centric model and the adoption of a content-centric model, enabling the direct addressing of content, independently of its location. This position is motivated by increasing primary use of the Internet for content dissemination, rather than for explicit communication between hosts [1].

This work was prepared as an 'hands-on' introduction to the field of ICN by taking advantage of the software packages offered by Project CCNx [2], an implementation of the Content Centric Networking (CCN) architecture [3]. We run a series of experiments using a simple and exemplifying testbed, evaluating the performance of the CCNx implementation over several network parameters and its suitability for use in constrained devices.

1.1 Content Centric Networking

Content-Centric Networking (CCN) [3] presents itself as a novel networking paradigm which moves away from the host-centric communication model: instead of retrieving content by first determining its location, CCN proposes accessing content directly, independently of its location in the network, by using content names as addresses.

Communication in CCNs is similar to a publish/subscribe model, in the sense that it is also driven by the consumers of data — the subscribers — which release Interest packets into the network, eventually received by other CCN nodes. These packets announce a subscriber's desire to fetch particular content via the specification of a content name field. Holders of content which matches the content name — the publishers — respond with Data packets, also referred to as Content Objects.

1.2 Objectives

The main objectives of this work are the following:

1) Acquire detailed knowledge about a particular ICN approach, by planning and deploying a simple ICN testbed based on Project CCNx [2], an implementation of the Content Centric Networking (CCN) approach [3].

2) Validate some of the benefits claimed by ICN approaches and in some cases compare the network performance of CCNx against equivalent 'host centric' solutions. The measurement parameters may vary with each test. Since the objective of CCNs is to reduce latency experienced by clients and network traffic load towards the origins of content [3], most of the evaluations are made by comparing values of throughput, latency and network traffic load.

3) We have the clear intention to test the performance of CCNx on 'real-world' constrained devices, differing our work from other studies (of different nature), performed in the near-past by Vahlenkamp

et al. [4, 5]. The idea is to evaluate how suitable is the CCNx implementation for direct application in constrained devices such those applied to e.g. Wireless Sensor Networks (WSNs) or Vehicular Networks (VANETS) [6, 7].

Chapter 2

Content Centric Networking

We now briefly introduce relevant aspects about the Content Centric Networking (CCN) paradigm, introduced in Section 1.1.

2.1 CCN Packet Types and Content Names

CCN uses two fundamental types of packets: Interest and Data packets (also referred to as Content Objects in the scope of Project CCNx [2], a software implementation of the CCN architecture).

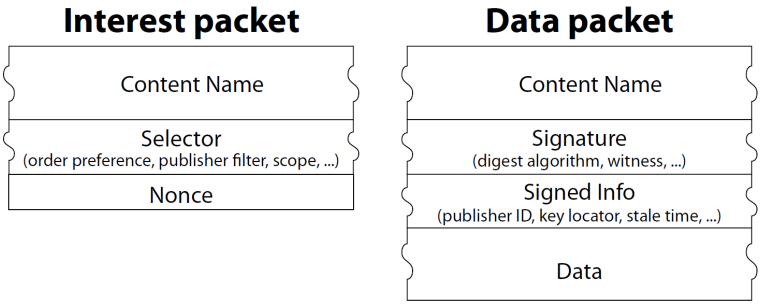


Figure 2.1: Overview of a CCN Interest and Data packet types [3].

Interest packets are originally released into the network by end nodes willing to access a particular content, addressing it via its content name. The original paper by Jacobson et al. [3] specifies the use of hierarchical content names, e.g. `/sports/football/march.schedule` (these are also used in practice in implementations of CCN, see Section 3.1.2), in order to allow publishers to release related content using a single prefix and make prefix matching actions intuitive. An Interest packet may comprise other fields (e.g. specifying time-limits, maximum number of CCN forwarding hops, etc.) and a 'nonce' field in order to discard Interest packets that may have been duplicated due to network loops (see Section 2.2 about CCN forwarding).

Data packets include the content itself with the addition of a cryptographic signature. The latter is created by using the data as well as a set of other fields such as a timestamp, the publisher's public key (required by other nodes to verify signatures), enabling the use of self-certifying names [3] Nodes are supposed to check the signatures and discard any content that fails verification.

2.2 Forwarding in Content Centric Networks

We now introduce the basic forwarding mechanics of a CCN node, starting with a description of its inner components.

A CCN node is composed by three main elements: (1) a Forward Information Base, (2) a Pending Interest Table (PIT) and (3) a Content Store (CS) [3]:

- **Forward Information Base (FIB):** Table holding entries which relate a name prefix and a list of interfaces to which Interest packets matching that content name prefix should be forwarded to.
- **Pending Interest Table (PIT):** A table for keeping track of the mapping between arriving Interest packets and the interfaces these have been received from, in order to save a reverse path for Data packets towards one or more subscribers (this may be a 1:N mapping, as an Interest packet matching the same content may be received in multiple interfaces).
- **Content Store (CS):** A cache for content, indexed by Content Name. This is a novel element, allowing for content storage at the Network level. In-network caching allows an Interest to be satisfied by a matching Data packet in any location other than the original producer of the static content, constituting one of the main content-oriented characteristics of CCN.

These elements are depicted in Figure 3.2.

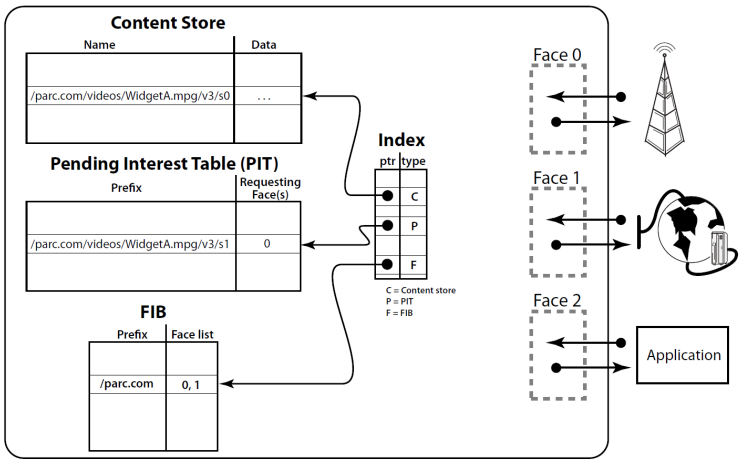


Figure 2.2: Overview of a CCN node's forwarding engine [3].

In CCN, communication is receiver-driven, i.e. having the desire to fetch a particular content, an end node releases an Interest packet into the network so that it is forwarded towards an appropriate content holder. More precisely, an intermediate CCN node performs the following sequence of operations upon the reception of an Interest packet [3]:

1. An Interest packet arrives on interface 0 of a given CCN node.
2. A longest prefix match on the content name specified in the Interest is performed. The CCN node will now look in its CS, PIT and FIB, in that order, in order to resume the forwarding action:
   (a) If there's a match on the node's CS, a copy of the respective CS entry will be sent back via interface 0, the Interest packet is dropped. **End.**
   (b) Else if there is an (exact) match on the PIT, interface 0 is added to the mapping list on the respective entry. The Interest packet is dropped (as a previous one has already been sent upstream). **End.**
   (c) Else if only a matching FIB entry is found, the Interest packet is forwarded upstream, via all remaining interfaces on the list (other than 0), towards an eventual content holder. A PIT entry <content name, interface 0> is added. **End.**
   (d) Else if there is no match at all, the Interest packet is simply discarded. **End.**

Note that in CCN only Interest packets are routed: as the intermediate CCN nodes forward the Interests, their respective PIT tables are updated with Interest-to-interface mappings, pre-establishing a reverse path for Data packets to follow as a content holder is found. When the reverse path is started by a CCN node holding a particular content name, each intermediate CCN node receiving a Data packet looks in its PIT for <content name, interface n> entries, and forwards the Data packet through all matching interfaces. In addition, a CS entry is created to cache the content locally at the node. If a Data packet with no matching PIT entries arrives, it is treated as unsolicited and discarded.

Chapter 3

Methodology

We evaluate the performance of an available ICN implementation — Project CCNx [2] [1] — by setting up a simple testbed, composed by commercial off-the-shelf (COTS) equipment. Besides validating some of the claims made by CCN, we attempt to compare CCNx applications to equivalent 'host centric' solutions.

This section (1) describes the CCNx software package to be used in the tests, (2) provides an overview of the testbed's structure and (3) details the protocol of the tests to be conducted.

3.1 Project CCNx

Project CCNx [2] provides an open source software reference implementation PARC's CCN architecture [3], with the objective of enabling experimentation in the network research community. The project provides APIs written in multiple programming languages (C, Java and Android), with extensive documentation.

CCNx is designed to run on top of existing transport protocols and addressing schemes — TCP, UDP, etc. — in an 'overlay' approach. This means that CCNx packets are tunneled within TCP or UDP flows running over IP. Although Project CCNx developers present this as a short-term approach for developers to start trying CCN, it is also presented as a feature, implying CCN's advantage over similar solutions in the current networking paradigms (e.g. web caching) [2, 3, 8, 9].

3.1.1 CCNx Applications

Project CCNx provides the following ensemble of example applications [10]:

- `ccnchat`: Simple application for real-time transmission of text messages from sender to receiver.
- `ccnfileproxy`: Allows one to share files within the node's file system to other CCNx nodes[7] (used as an alternative to setting up a CCNx repository [10]).
- `ccnputfile` / `ccngetfile`: Pair of applications to write/read files to/from CCNx nodes, e.g. to store files into a CCNx repository.
- `ccnsendchunks`: Simple application which divides a given piece of content in 'chunks' upon the reception of Interest packets directed at that same content, and sends the 'chunks' as Data packets towards the requester. Interest packets are generated via the complementing `ccncatchunks2` application.
- VLC CCNx plugin: Plugin to the VLC media player [11], which allowing a CCNx node to remotely playback a video stored on a CCNx repository, identified by its 'content name'.
- Others (see Section 'What To Look At' in [10]).

[1] We henceforth address the CCN networking concept as 'CCN' and its implementation as 'CCNx'.

The diversity of application types listed above may translate into a reasonably sized set of test possibilities. However, their specific CCNx implementations may significantly differ from those one would use in the non-CCNx case (e.g. `ccnchat` vs. an established instant messaging application or protocol e.g. XMPP). The influence of implementations should be taken into account when analyzing network performance results.

3.1.2 Specificities of CCNx Applications

Here we provide CCNx application-specific information, useful for understanding the tests and results shown in later sections.

CCNx Forwarding Tables

Since CCNx works as an 'overlay', CCN's routing scheme based on content must be translated into IP routing at a given point. In CCNx this is accomplished by the routing daemon, `ccnd`, the main process running in CCNx nodes [12].

As dynamic routing in CCNs is still an area of current research and not well supported by current CCNx releases [13, 14], we adopt the same strategy as that applied in previous work [4, 5] and manually added static routes to CCNx's Forwarding Table (FIB), using the `ccndc` utility [12]. We provide examples of the `ccndc` command in Appendix A.

Disseminating Content with CCNx

In CCNx, content sources use a specific repository application for storing content and make it available in CCNx networks by responding to matching Interests, `ccnr` [12]. The `ccngetfile` application [12] can then be used to release Interest packets to the CCNx network, querying for particular content. Appendix A presents some concrete examples of the usage of these applications.

CCNx also provides `ccnsendchunks` and `ccncatchunks2`, another duet of applications for content exchange. `ccnsendchunks` takes content (e.g. a file) as input, and produces Data packets containing 'chunks' of the content (blocks of a given size, in byte) as it receives Interests, or at a rate of one-per-second, whichever is faster [2]. The complementary Interest-producing `ccncatchunks2` also uses pipelining of Interests, varying its 'window size' according to the rate of arrival of Data packets. This behavior is also analyzed in some of the tests specified in Section 3.3. We provide examples of the use of both commands in Appendix A.

Video content stored in a CCNx repository may be directly played back in VLC media player [11], using a special CCNx plugin which generates Interests for that content (any type of content playable by VLC, e.g. a `.avi` file), receives the respective Data packets, decodes them and progressively plays them back. Again, we provide examples of its usage in Appendix A.

3.2 Testbed Description

The basic testbed structure used for evaluation of CCNx is depicted in Figure 3.1. Some tests require alterations to this scheme, nevertheless such alterations are promptly indicated in the individual test descriptions, in Section 3.3. Table 3.1 provides a list with more details of the testbed equipment.

The basic scenario depicted above includes 3 COTS residential routers, labeled as 'CCNx1', 'CCNx2' and 'CCNx3' (Linksys WRT160NL) running OpenWRT [3] [15], a Linux-based operating system. These represent the Inner Nodes (INs) of a network, serving as CCNx/IP routers (see Section 3.2.1 for details on the installation of CCNx software on embedded Linux systems). Table 3.1 summarizes the hardware specifications of these devices.

[2] Although no MAN page exists for this command, the following resource may be used as an unofficial source of information: https://www.ccnx.org/pipermail/ccnx-dev/2010-April/000189.html.

[3] 'Barrier Breaker' version, SVN revision 35323, available at https://dev.openwrt.org/browser/trunk?rev=35323.

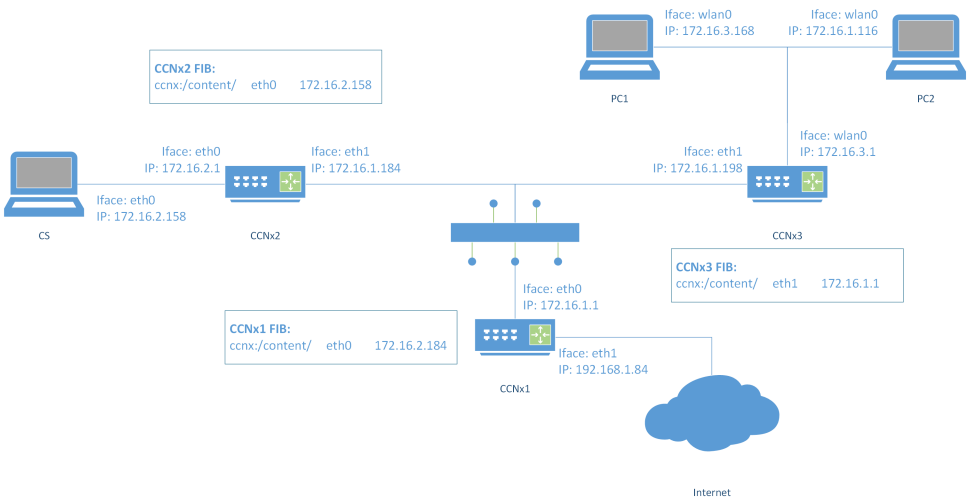


Figure 3.1: Basic arrangement for the proposed testbed.

The arrangement separates the End Nodes (ENs) — consumers of content — and the Content Sources (CSs) — generators of content — in different IP subnets (172.16.x.x), so that routing is performed in both CCNx and 'host centric' scenarios [4].

Test Element	Label(s)	Type	Relevant Specs.	Qty.
Content Source(s)	CS	Laptop (w/Ubuntu 12.04)	Wi-Fi: 802.11 b/g Ethernet: 10/100 Mbps	1
Inner Node(s)	CCNx1 CCNx2 CCNx3	Linksys WRT160NL [16]	CPU: Atheros 9130-BC1E 400 Mhz RAM: 32 MB Wi-Fi: 802.11 b/g/n Ethernet: 10/100 Mbps	3
End Node(s)	PC1 PC2	Laptop (w/Ubuntu 12.04)	Wi-Fi: 802.11 b/g/n Ethernet: 10/100 Mbps	2

Table 3.1: Table of hardware for the testbed depicted in Figure 3.1.

3.2.1 Testbed Setup Details

Setting up CCNx at End Nodes (ENs)

We have used the latest release of CCNx (0.8.1) [5] at the EN nodes. In addition to the basic CCNx package, the VLC plugin (available with CCNx 0.8.1 package) was separately compiled and installed at the ENs, in order test dissemination of video content in CCNx networks. The VLC version 2.0.8 TwoFlower was used for testing at the ENs.

As the objective of the work is to monitor the performance of CCNx according to network parameters such as network load, throughput, latency, etc. a patched version of Wireshark (version 1.8.6) was re-compiled to include a CCN packet dissector [17].

[4] As CCNx works as an overlay, IP routing will be performed in both CCNx and 'host centric' test versions. Nevertheless, as this work may potentially origin subsequent research (e.g. testing dynamic routing in CCNx), we believe it is good practice to test this scenario from the start

[5] Compiled from source, available at http://www.ccnx.org/software-download-information-request/download-releases/

Setting up CCNx at Inner Nodes (INs)

Testing CCNx on 'real-world' constrained devices was a clear intention of this work, making it different from other studies (e.g. Vahlenkamp et al. [4,5]). Although no CCNx packages were directly available to OpenWRT, a custom OpenWRT build with a CCNx package [18] (version 0.7.2) designed for a different distribution — CeroWRT [19] — was successfully accomplished.

3.3 Test Specification

The following subsections describe the set of tests to perform over the base testbed shown in Section 3.2. A summary of the test setups is given in Table 3.2.

Number	Applications	Parameters	Testbed Configs.	Descr.
1	File transfer	Throughput Qty. packets/byte Other	Fig. 3.2	Simple file transfer between two ENs.
2.1	File transfer	Network load Qty. packets/byte CPU and RAM usage	Fig. 3.1	File transfer with CCNx multihop forwarding.
2.2	Video Streaming	Network load Qty. packets/byte CPU and RAM usage	Fig. 3.1	Video streaming under multihop forwarding.
2.3	File Transfer	Network load Qty. packets/byte CPU and RAM usage	Fig. 3.3	CCNx multihop forwarding with multiple paths.

Table 3.2: Summary of the tests to be ran on the testbeds depicted in Figures 3.1, 3.2, and 3.3.

3.3.1 Test 1 - CCNx Throughput and Overhead

The objective of this test is to assess CCNx throughput and overhead by monitoring the exchange of content between two ENs (PC1 and CS) using `ccnsendchunks` and `ccncatchunks2`. In this case, a simpler scheme than that shown in Figure 3.1 is used, depicted in Figure 3.2.

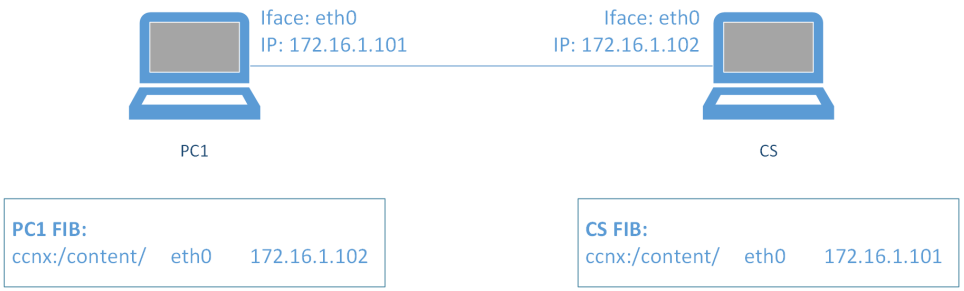


Figure 3.2: Example of the configuration for Test 1. PC1 shall execute `ccncatchunks`, while CS executes `ccnsendchunks`.

As `ccncatchunks2` implements its own flow control mechanism, the FIBs of both CCNx nodes are configured to route packets over UDP (see Section 3.1.2 for details), in order to avoid interference from TCP's own flow control. The content used for exchange in the tests consists in randomly generated data files of 500 kB, 5 MB and 50 MB. The chunk size parameter of `ccnsendchunks` is tested under three different values: 1024 byte, 4096 byte and 8192 byte.

The following parameters are measured:

1. Throughput;

2. Number of packets exchanged between ENs, discriminated by type;

3. Bytes exchanged between ENs;

4. Other parameters specific to `ccncatchunks2` flow control mechanism.

Parameters 1, 2 and 3 from the above list shall be measured on a non-CCNx setting, by transferring data over FTP. The same setup shown in Figure 3.2 is used, with CS running an FTP server (`proftpd` [6]) and PC1 fetching content using an FTP client. Again, the content to be exchanged in the tests consists in the same randomly generated data files used in the CCNx tests.

3.3.2 Test 2.1 - CCNx Multihop Forwarding (File Transfer)

Here we test CCNx on the testbed scenario depicted in Figure 3.1. The objective is to monitor network parameters as CCNx routers (INs) forward Interest and Data packets to/from a content source (CS) and two content consumers (PC1 and PC2), both sending Interests for the same content.

The same type of content as that used in Test 1 is used, but limited to a 5 MB file [7]. In this case, the `ccnr` application is used at CS to host a CCNx file repository, while PC1 and PC2 retrieve the file using `ccngetfile`. Two subtypes of test are conducted: (1) PC1 and PC2 start the file transfer separately in time, i.e. PC2 starts releasing Interest packets after PC1 receives its last Data packet; and (2) PC1 and PC2 content requests overlap. The integrity of the transmitted files is verified via an MD5 checksum, for all transfers.

The following parameters are measured:

1. Network load (packets/sec) at the different interfaces of INs, over time;

2. Number of packets exchanged between INs, discriminated by type;

3. Bytes exchanged between INs;

4. CPU and memory usage in each IN, over the transfer time.

As packet sniffing tasks are now being conducted at constrained nodes, we use `tcpdump` with the `-s 500` option, i.e. set to capture the initial 500 byte of each packet. Despite the fact that not all packet information is saved, 500 byte are enough to capture the CCN's headers, along with the headers of any underlying protocols, allowing us to correctly assess packets sizes [8].

Parameters 1 to 4 from the above list shall be measured on a non-CCNx setting, by transferring data over FTP. The same testbed setup as that used for the CCNx case (shown in Figure 3.1) is used, with CS running an FTP server (`proftpd`) and both PC1 and PC2 fetching content using an FTP client, in a non-overlapping fashion. Again, the content to be exchanged in the tests consists in the same randomly generated data files used in the CCNx tests.

3.3.3 Test 2.2 - CCNx Multihop Forwarding (Video Streaming)

The scenario for this test is similar to that of Test 2.1, now using CCNx's VLC plugin to playback a video stored at CS instead of directly transferring a file.

PC1 and PC2 both start the playback of video content using CCNx's VLC plugin, in the manner shown in Section 3.1.2. Just as in Test 2.1, both non-overlapping and non-overlapping test subtypes are ran. The same parameters as those measured in Test 2.1 are evaluated in this case.

[6] Available at http://www.proftpd.org/

[7] Preliminary tests with larger files (e.g. 50 MB) have been conducted, however since `ccnd` keeps its cache in memory and due to issues in the cache replacement mechanism of OpenWRT's `ccnd`, the RAM of the Linksys WRT160NL would exhaust quickly forcing a system restart.

[8] In addition to capture file size reductions, the number of packets dropped by the kernel has consistently been reduced to 0 in all experiments.

A video of size 6.3 MB [7] and duration of 8 seconds is used in the test: although not a quantitative measurement, the qualitative assessment of the video playback is also given (e.g. occurrence of playback gaps, frame 'freezing', etc.).

3.3.4 Test 2.3 - CCNx Multihop Forwarding (Multiple Paths)

The objective of this test is to assess how CCNx forwards packets when multiple paths exist between content sources and consumers. To do so, we use a slight variation of the previous testbed setup, shown in Figure 3.3.

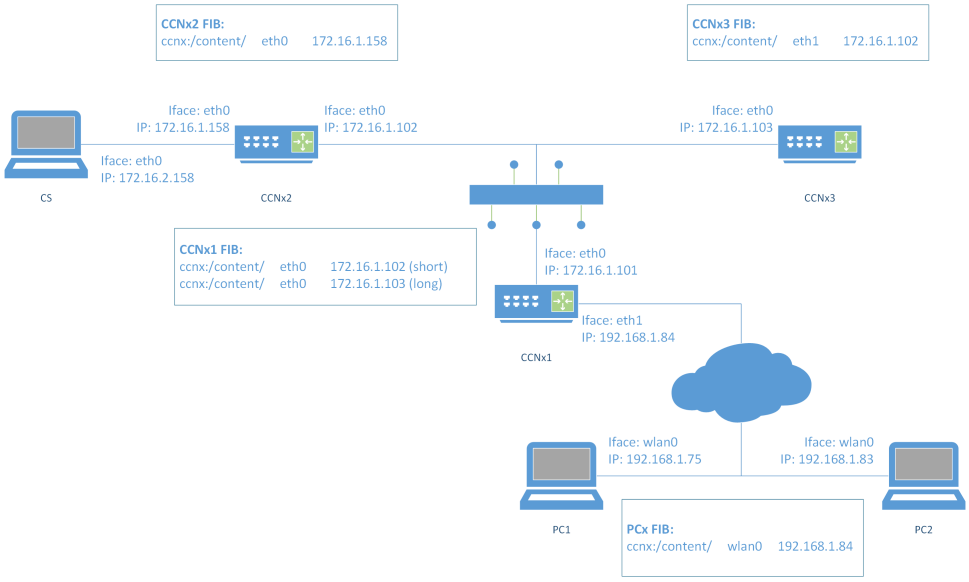


Figure 3.3: Testbed arrangement for Test 2.3.

The same type of content as that used in Test 2.1 (see Section 3.3.2) is used, limited to a 5 MB file. Again, the `ccnr` application is used at CS to host a CCNx file repository, while PC1 and PC2 retrieve the file using the `ccngetfile` application.

Three subtypes of test are conducted:

1. PC1 and PC2 retrieve the content from CS via a short path, via CCNx1 and CCNx2, in a non-overlapping fashion.
2. PC1 and PC2 retrieve the content from CS via a long path, via CCNx1, CCNx3 and CCNx2, in that order.
3. PC1 and PC2 retrieve the content from CS with the INs configured to forward packets via both short and long paths.

The different paths are specified at the INs by adding manual entries to the FIB, using the `ccndc` command (see Section 3.1.2 for details). To test the influence of the order by which `ccndc` commands are ran, i.e. the order by which routing entries are put into the FIB, we first test a setting for which the 'short path' is included first, followed by another setting for which the 'long path' is included first. The same parameters as those measured in Tests 2.1 and 2.2 are evaluated in this case.

Chapter 4

Experimental Results

Here we present the results from the tests specified in Section 3.3.

We present only part of the results in this chapter, as the complete setting of measurements is rather large considering the required length for this assignment. The complete collection of results can be consulted in Appendix B.

4.1 Test 1 - CCNx Throughput and Overhead

We choose the results for a representative case and elaborate on its discussion. The same reasoning and arguments can then be used to understand the remaining results shown in Appendix B.

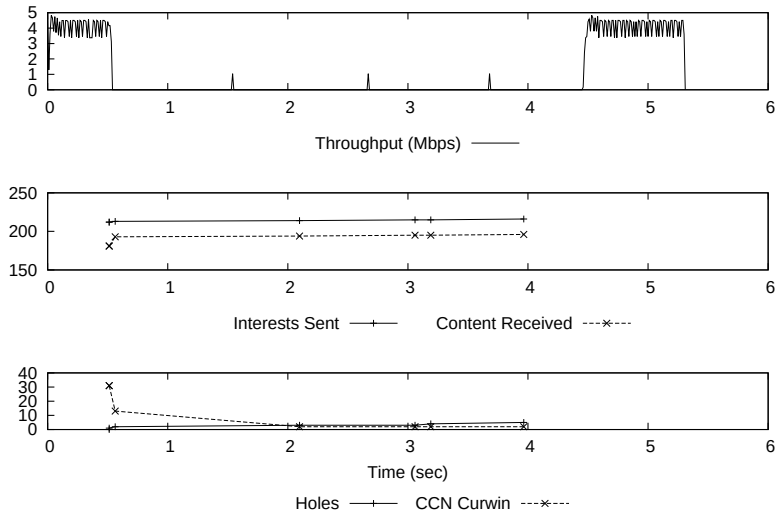


Figure 4.1: Results for Test 1: Throughput in Mbps, as perceived by PC1, when retrieving a file of size 500 kB with a 1024 byte 'chunk' size.

Figure 4.1 shows the results for the throughput and other CCNx specific statistics, as perceived by PC1, when retrieving a file of size 500 kB with a 1024 byte 'chunk' size. The action of the joint flow control mechanism applied by both `ccncatchunks2` and `ccnsendchunks` applications (introduced in Section 3.1.2) is visible: note how after 0.5 seconds of transfer the throughput drops as well as the 'Curwin' value (i.e. the size of the window/pipeline of Interests `ccncatchunks2` sends to `ccnsendchunks` at a time), which drops from 31 (the maximum size) to 1. Note that such an event is coincident with

the occurrence of 'holes', i.e. a lack of Interest-to-Data packet correspondence. After this point, small transferring peaks are visible after approx. 1 second intervals, once again consistent with the expected behavior of the `ccnsendchunks` application. This flow control mechanism has a severe impact on the overall goodput, i.e. the number of useful information bits over the total amount of time required for transfer (approx. 5.35 seconds).

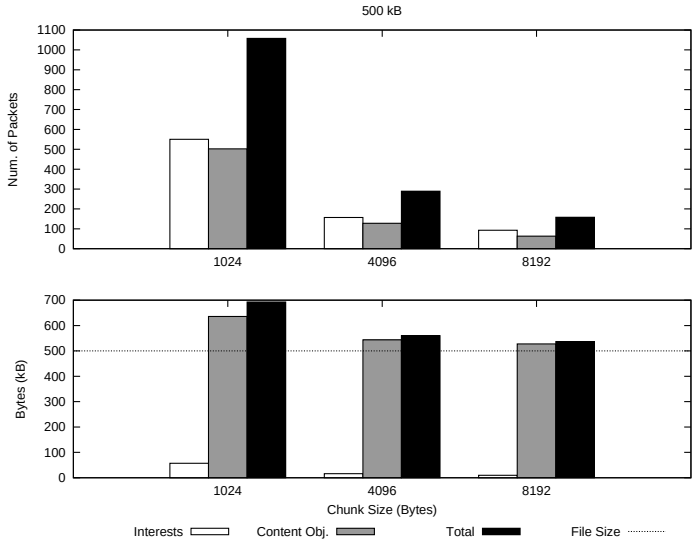


Figure 4.2: Results for Test 1: Packet and byte quantities, measured at PC1, for a file size of 500 kB.

Figure 4.2 presents the results for the number of packets and bytes, as measured by the capture filters in PC1. Again, we include the results for a 500 kB file size and a 'chunk' size of 1024 byte. The amount of Interest packets consistently surpasses the amount of Data packets, for all 'chunk' sizes. This is explained by the 'loss' of a batch of N pipelined Interest packets, due to overloading of the `ccnsendchunks` application. In the case of 500 kB, e.g. for a 'chunk' size of 1024 byte, one can situate the mismatch between Interest and Content Objects (or Data packets) within the interval $]0; 100]$. This is consistent with the results shown in Figure 4.1, with the mismatch being equal to the sum of values of 'Curwin', at the time of occurrence of 'hole' events.

We consider the results for the file size of 5 MB, depicted in Figures 4.4 and 4.3, to further elaborate on CCNx throughput and overhead. In order to easily visualize the impact of the variation of 'chunk' size in overhead, we have added an horizontal line on the byte quantity charts in Figure 4.3, equal to the respective file size.

The difference between the quantity of Content Object bytes and the file size decreases as the 'chunk' size increases, i.e. the overhead decreases with the increase of the chunk size. This can be explained by several factors, identified by examining the packet capture logs:

- The header of a CCNx Content Object, is composed by the following fields (consistent with the Data packet descriptions given in Section 2.1), resulting in a total max. of 215 byte:

  1. Type of packet code (2 byte)
  2. CCN signature (128 byte)
  3. Content name codified as ASCII (32 to 35 byte)
  4. Signed information (50 byte)

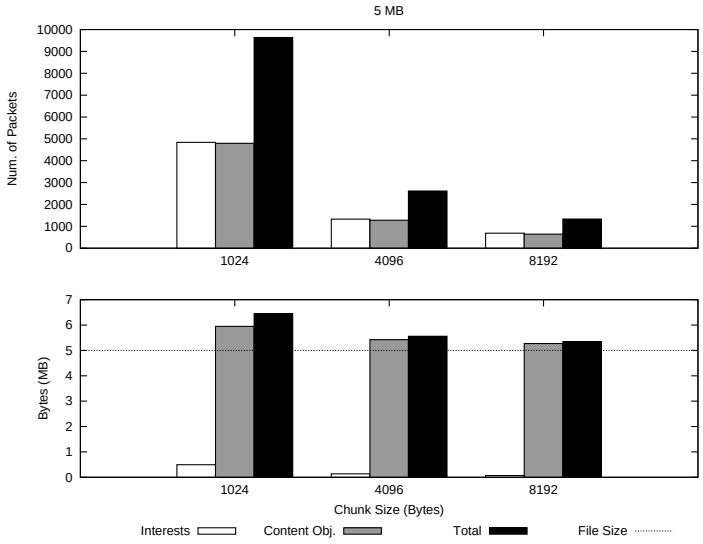


Figure 4.3: Results for Test 1: Packet and byte quantities, measured at PC1, for a file size of 5 MB.

- CCNx (at least `ccnsendchunks` in particular) relies on IP fragmentation to transport Content Object packets with data fields (i.e. payloads) larger than 1480 byte, which is in turn related with the maximum size allowed for a standard Ethernet frame, 1518 byte.

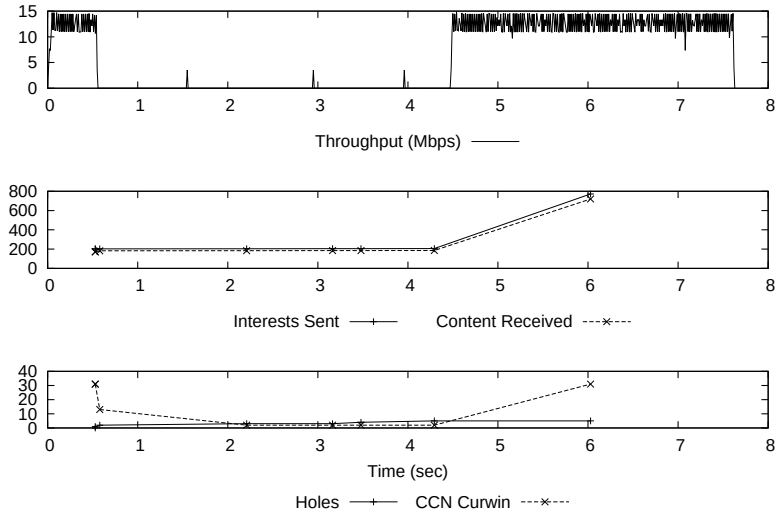
So in the case of 'chunk' sizes C of 1024 byte, for every 1024 byte of data there is an additional 220 byte [1], while in the case of $C = 4096$ or 8192 byte that header is 'spread' over larger payload sizes, reducing the overhead. Nevertheless, the overhead of CCNx may then be effectively set at 188 byte plus the size of the content name string.

Regarding Figure 4.4, the throughput increases with the 'chunk' size as well. By verification of the packet capture logs, the throughput limitations are imposed by the rate of generation of Content Objects and not by the rate at which Interest packets are sent (maybe due to the signature generations for each content block by part of the `ccnsendchunks` application [2]), as for all file sizes and 'chunk' sizes, the average inter-packet arrival times for Content Objects (in periods where the throughput is maximum) are approximately the same, ~ 2.5 milliseconds. Considering the sizes of Content Objects for each 'chunk' size (including CCNx header), we have the following estimates for throughput values, consistent with the values shown in Figures 4.1 and 4.4:

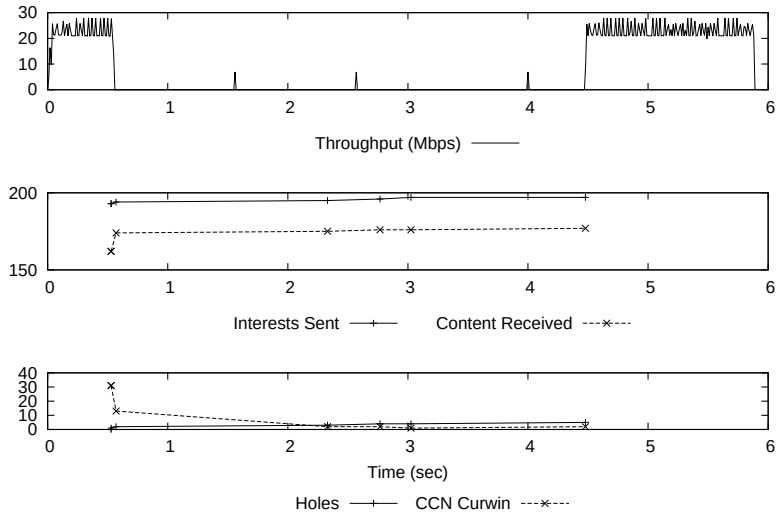
- 1024 byte: $\frac{1}{0.0025} \times (1024 + 220) \times 8 \approx 3.98\,\text{Mbps}$
- 4096 byte: $\frac{1}{0.0025} \times (4096 + 220) \times 8 \approx 13.8\,\text{Mbps}$
- 8192 byte: $\frac{1}{0.0025} \times (8192 + 220) \times 8 \approx 26.9\,\text{Mbps}$

The results for the non-CCN case, obtained by fetching the same files via FTP are shown in Appendix B, in Section B.1. As expected, the use of channel bandwidth is much more efficient (approaching throughput values of Fast Ethernet's 100 Mbps) as well as lower values of overhead. In terms of packet numbers, as FTP avoids payload sizes larger than 1416 byte (so that frames do not exceed Ethernet's MTU of 1500 byte), the packet counts are (compared with the number of Content Objects) larger.

[1] By verification of the capture logs, the only variable field is the content name, regardless of the 'chunk' size.
[2] See the discussion in https://www.ccnx.org/pipermail/ccnx-dev/2010-April/000188.html.



(a)



(b)

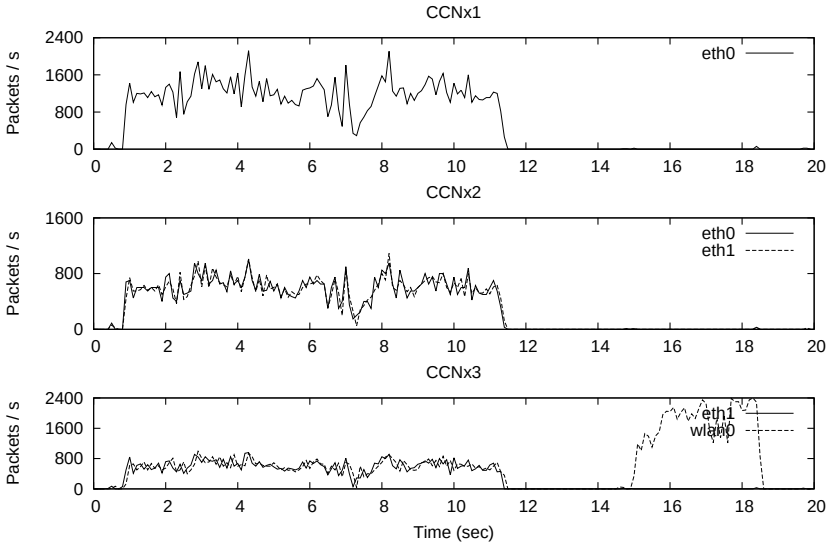
Figure 4.4: Results for Test 1: Throughput in Mbps, as perceived by PC1, when retrieving a file of size 5 MB with 4096 and 8192 byte 'chunk' sizes.

4.2 Test 2 - CCNx Multihop Forwarding

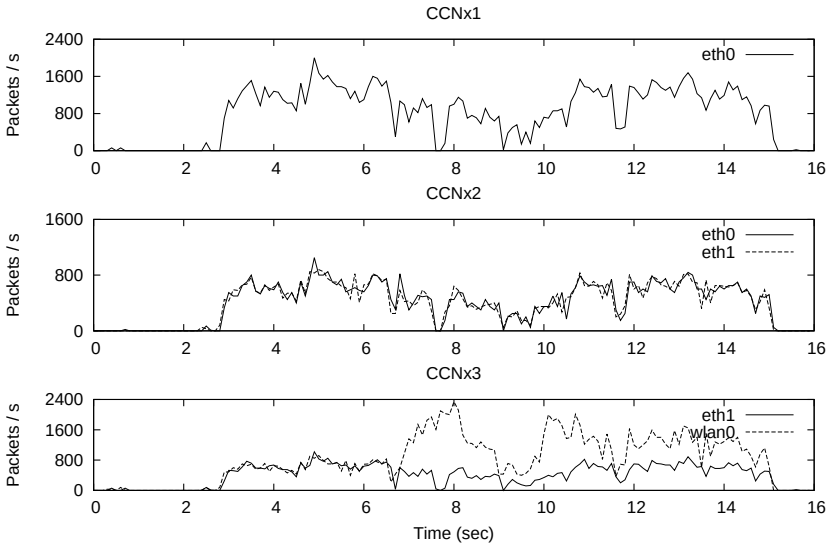
We now present the results for the CCNx multihop forwarding tests, described in Section 3.3. We divide the presentation of the results by subtype, i.e. the results for Tests 2.1, 2.2 and 2.3 are presented individually.

4.2.1 Test 2.1 - CCNx Multihop Forwarding (File Transfer)

Figure 4.5 depicts the network load (in packets per second) at each of the CCNx nodes in the testbed (see Figure 3.1), for both non-overlapping and overlapping file retrieval cases. A 5 MB file is retrieved via a wireless link between CCNx node (IN) CCNx3 and ENs PC1 and PC2. The CCNx routes established for this test follow those specified in Figure 3.1.



(a)



(b)

Figure 4.5: Results for Test 2.1: Network load, in packets per second, as perceived by each of the CCNx nodes in the testbed, while PC1 and PC2 retrieve a file of size 5 MB via a wireless link, through CCNx3. The graph in (a) shows the results for the non-overlapping file retrieval, while (b) shows the overlapping case.

In Figure 4.5(a) the effect of CCN's in-network caching characteristic is perfectly noticeable: the load

on CCNx3 shows two clear activity clusters, one for the initial retrieval of the file by PC1, which extends for approx. 11 seconds and another for the file retrieval by part of PC2, which takes approx. 3 seconds. During the initial period, the remaining CCNx nodes display a similar activity pattern, corresponding to the forwarding of Interest and Data packets between the several CCNx hops, up to the CS node. The values for CCNx1 seem doubled when compared to the other nodes, since the forwarding is made via a single interface, `eth0`. The second period only shows activity on CCNx1, as expected, as PC2 is retrieving the content cached in CCNx1's Content Store. The transfer is also clearly faster than in the initial retrieval, as no intermediate CCNx forwarders exist between content publisher and subscriber.

In Figure 4.5(b), the load values on CCNx3 are similar on both interfaces `eth1` and `wlan0`, up until approx. 7 seconds from the start of the test. At this point, a spike on the load for interface `wlan0` is verified, corresponding to the entry of PC2. For an initial period (7 to 8 seconds) PC2 retrieves the content already cached at CCNx3, while for the remaining time the Data packets arriving at CCNx3 from interface `eth1` are multicasted to both PC1 and PC2. Figure 4.6 shows that in terms of packet counts — Interests and Data packets — there is no difference between the overlapping and non-overlapping cases.

The remaining test cases and results for Test 2.1 can be consulted in Appendix B, in Section B.2.1.

4.2.2 Test 2.2 - CCNx Multihop Forwarding (Video Streaming)

We now present the results for Test 2.2 in Figure 4.7. We limit the presented results to the network load values at each of the CCNx nodes, leaving the remaining measurements to Appendix B.

Similarly to the file transfer results shown in Section 4.2.1, Figure 4.7(a) shows two separate network activity clusters: (1) up until approx. 14 seconds for the streaming to PC1 and (2) from approx. 17 till 25.5 seconds for the streaming to PC2. In terms of time length, (1) clearly exceeds the time duration of the video used as sample, which was reflected by several gaps in the video playback. In the case of (2) such gaps were still noticed, but not as frequent or as long.

In the case of Figure 4.7(b), again similar results to that of Test 2.1 were verified, including a spike in the values of the load at CCNx3's `wlan0` interface upon the initialization of PC2's streaming activity, at around 5 seconds after the initialization of the test. Again, the time length for the video download is extended, now reflected by frequent gaps in the video playback.

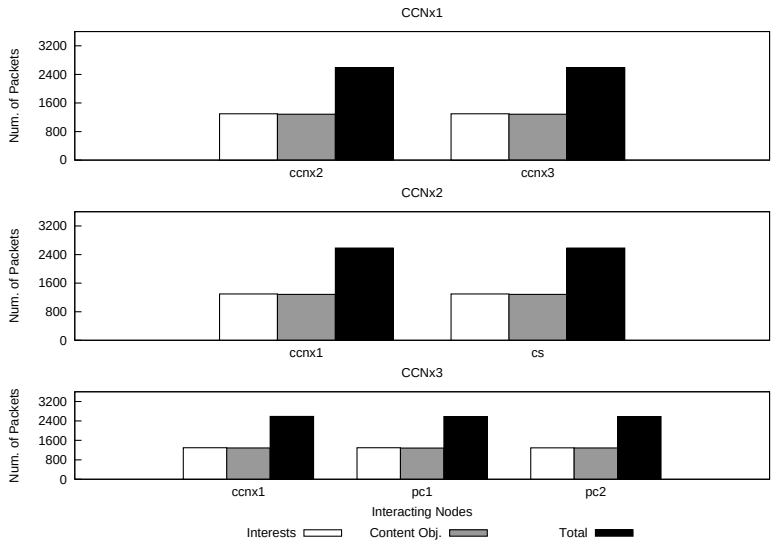
Figure 4.8 shows the values of CPU and memory consumption for the non-overlapping test case. The 'memory' line corresponds to the amount of RAM used by the CCNx's main process, `ccnd`. It is clear how the memory usage increases as the content of the video file is forwarded among the CCNx nodes, stabilizing at approx. 14 seconds after the test initialization, at the same time the transfer with PC1 is finished. Even though these measurements evaluate more the implementation of CCNx than the CCN concept itself, these highlight a need for efficient cache replacement policies and CCNx applications for constrained devices such as the routers used in this experiment. Although for general use cases, CCN nodes such as home gateways may have caching disabled, for experimental applications such as CCNx applied to Vehicular Networks (VANETS) [6, 7], these concerns should be taken into account.

4.2.3 Test 2.3 - CCNx Multihop Forwarding (Multiple Paths)

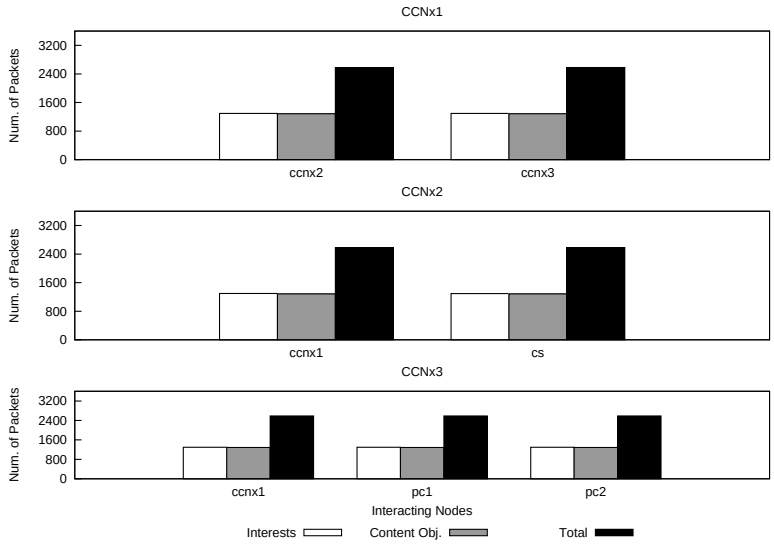
In the case of the multiple path scenario presented in Section 3.3.4, we show the results for the packet counts registered at each CCNx forwarding node, as the load values are similar to those of the non-overlapping version of Test 2.1. Other measurements not shown here may be consulted in Appendix B, Section B.2.3.

Figure 4.9(a) shows the packet counts (Interest and Content Objects) registered at each CCNx forwarding node, exchanged between the respective peer elements, in this case for the 'long' route only.

A note should be made to help distinguish the origin of Interest and Content Object packets in the chart, which is related to the way the routes are established among nodes (see Figure 3.3): e.g. for CCNx1 the value of Interest packets associated with 'PC1' and 'PC2' represents those *received* from PC1 and PC2, while the quantity associated with 'CCNx3' represents the number of Interests *forwarded*



(a)

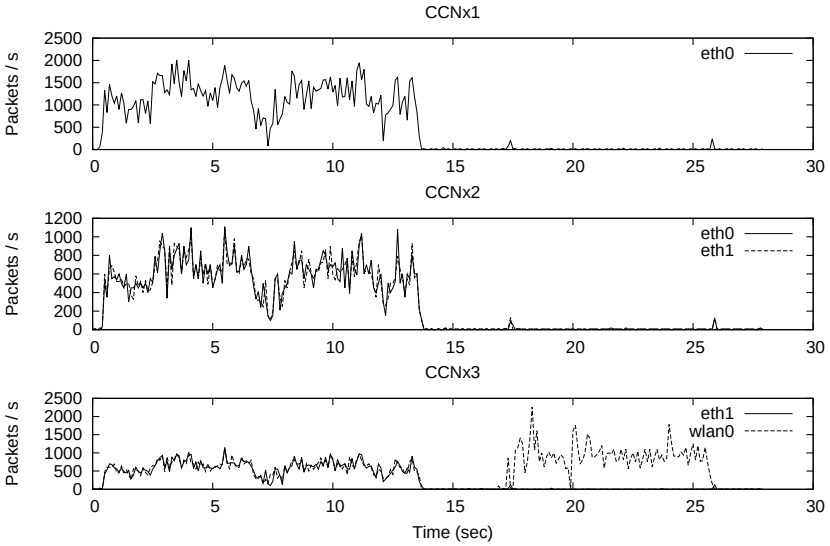


(b)

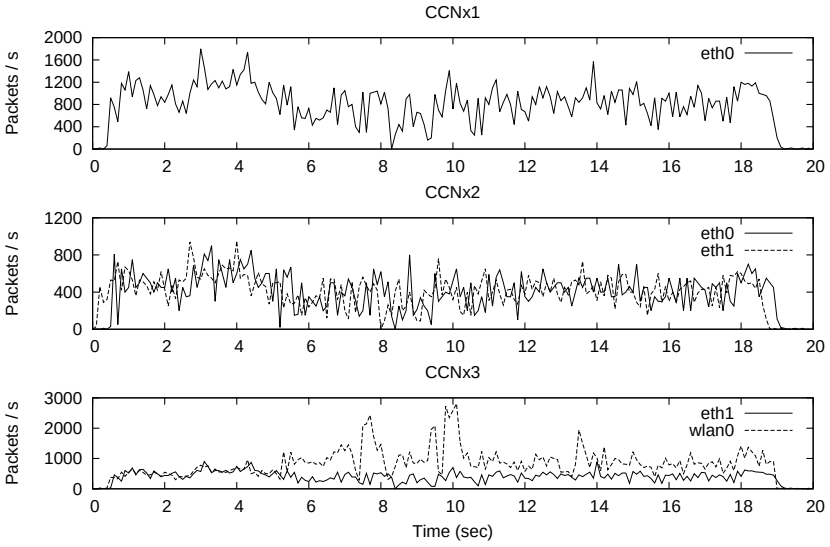
Figure 4.6: Results for Test 2.1: Packet counts (Interest and Content Objects) registered at each CCNx forwarding node, exchanged between the respective peer elements. Case (a) presents the results for the non-overlapping case, while (b) shows the results for the overlapping case.

to CCNx3. The same reasoning can be applied to the Content Objects: those associated with 'PC1' and 'PC2' are *forwarded* to these nodes, while the values associated with 'CCNx3' correspond to the Content Objects *received* from CCNx3.

The values in Figure 4.9(a) are expected, with no divergences between the number of forwarded packets between each of the INs and ENs. Furthermore, the existence of a single path (or conversely the absence of the 'short' path) is noticed by the absence of values associated with CCNx2 and CCNx1 for



(a)



(b)

Figure 4.7: Results for Test 2.2: Network load, in packets per second, as perceived by each of the CCNx nodes in the testbed, while PC1 and PC2 playback a streamed video file of size 6.3 MB and 8 second duration, via a wireless link, through CCNx3. The graph in (a) shows the results for the non-overlapping streaming case, while (b) shows the overlapping case.

nodes CCNx1 and CCNx2, respectively.

In the case of Figure 4.9(b), the presence of multiple paths is visible by the amount of Interest packets which are forwarded along the 'long' path, i.e. from CCNx1 to CCNx3 and then from CCNx3 to CCNx2. From consultation of the capture logs, one verifies that a small quantity of Interest packets (~ 90) is forwarded by CCNx1 to both CCNx2 and CCNx3. Only a residual number of Content Objects is forwarded by CCNx2 to CCNx3, corresponding to special CCNx packets exchanged among peers, not

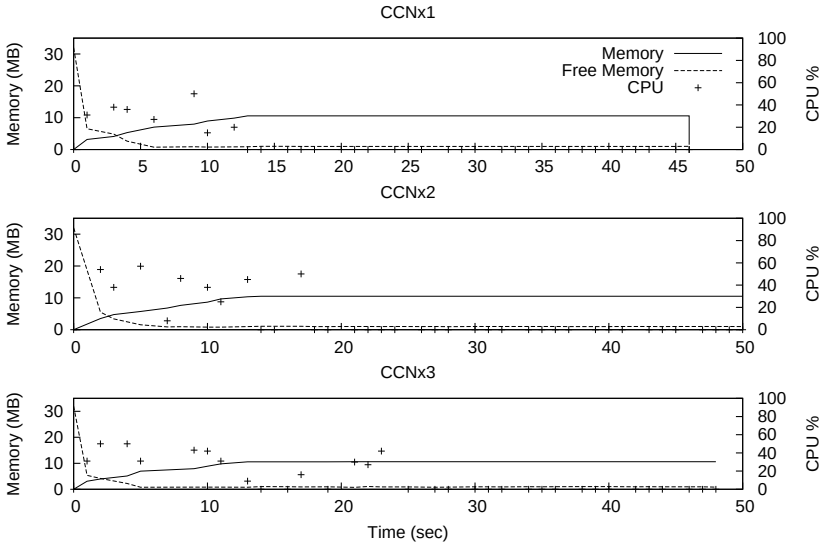
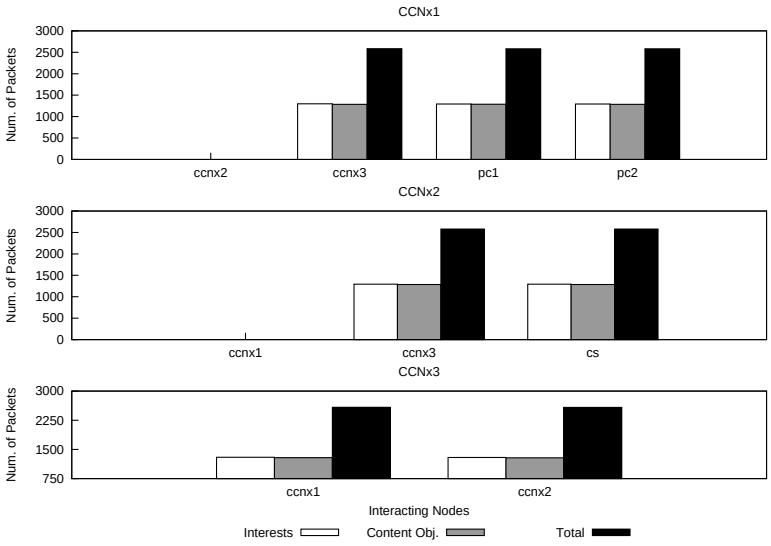
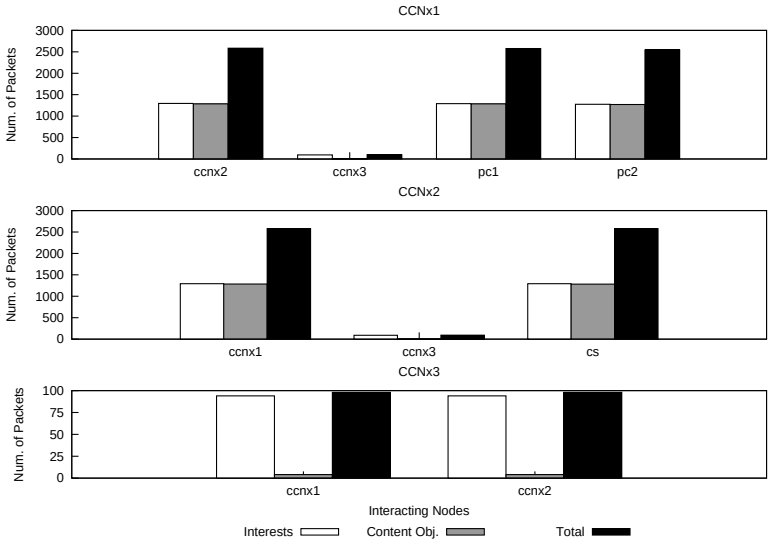


Figure 4.8: Results for Test 2.2: CPU and memory utilization at all CCNx nodes, during the streaming of a video file of size 6.3 MB and 8 second duration, non-overlapping case. Regarding the memory values, the 'memory' line corresponds to the amount of RAM occupied by the `ccnd` process, while the 'free memory' corresponds to the amount of free memory in the system.

directly related to the content being requested by PC1 and PC2 (these pertain to 'probing' Interest packets, periodically sent along alternative interfaces in order to detect better paths [20]). As the Interests forwarded by CCNx3 to CCNx2 are duplicated (arriving after those issued by CCNx1), these are discarded by CCNx2.



(a)



(b)

Figure 4.9: Results for Test 2.3: Packet counts (Interest and Content Objects) registered at each CCNx forwarding node, exchanged between the respective peer elements. Case (a) presents the results for the 'long' route only, while (b) shows the results for multiple paths, i.e. with the junction of both 'long' and 'short' routes.

Chapter 5

Conclusions

This work provided an introduction to the inner workings of a particular ICN approach via experimentation with CCN's open source implementation Project CCNx.

We have tested CCNx by mounting and running several experiments over a simple testbed of constrained devices (low CPU power and memory capabilities). We verify that a direct application of CCNx to such device types is not feasible, considering the test cases presented here.

As future work, since Web Caches and CCNx both at rather high layers of the network stack (implementing a similar functionality) it could be interesting to compare the performance of CCNx against the use of intermediate Web Caches. Tests with a modified version of the `ccnchat` application may prove useful for evaluating exchange patterns with small and sporadic messages, the scenario often found in M2M, WSNs or VANETS.

The results and insights provided by this work can prove useful for alterations of CCNx at both conceptual (e.g. alternatives to the transport mechanisms provided by `ccncatchunks2`) and implementation (e.g. throughput limits imposed by `ccnsendchunks`) levels, specially when regarding the application of CCNx to applications using constrained devices.

Bibliography

[1] George Xylomenos, Christopher N. Ververidis, Vasilios a. Siris, Nikos Fotiou, Christos Tsilopoulos, Xenofon Vasilakos, Konstantinos V. Katsaros, and George C. Polyzos. A Survey of Information-Centric Networking Research. *IEEE Communications Surveys & Tutorials*, PP(99):1–26, 2013.

[2] Palo Alto Research Center, Inc. Project CCNx. Available as http://www.ccnx.org/about/, October 2013.

[3] Van Jacobson, Diana K Smetters, James D Thornton, Michael F Plass, Nicholas H Briggs, and Rebecca L Braynard. Networking Named Content. *Proceedings of the 5th international conference on Emerging networking experiments and technologies CoNEXT 09*, 178(1):1–12, 2009.

[4] Matthias Wählisch, Thomas C Schmidt, and Markus Vahlenkamp. Bulk of Interest : Performance Measurement of Content-Centric Routing. *SIGCOMM Comput. Commun. Rev.*, 42(4):99–100, 2012.

[5] Markus Vahlenkamp. CCNx Meassurement Testbed Implementation. Technical report, Hamburg University of Applied Sciences, 2012.

[6] Marica Amadeo, Claudia Campolo, and Antonella Molinaro. Enhancing Content-Centric Networking for Vehicular Environments. *Computer Networks*, 57(16):3222–3234, November 2013.

[7] Giulio Grassi, Davide Pesavento, Lucas Wang, Giovanni Pau, Rama Vuyyuru, Ryuji Wakikawa, and Lixia Zhang. Vehicular Inter-Networking via Named Data. pages 1–10, 2013.

[8] Nathan Willis. Content-Centric Networking with CCNx. Available as http://lwn.net/Articles/520670/, October 2012.

[9] N. Blefari Melazzi, M. Cancellieri, A. Detti, M. Pomposini, and S. Salsano. The CONET Solution for Information Centric Networking. Technical Report January, University of Rome "Tor Vergata", 2012.

[10] Paul Muther. CCNx Release Field Guide. Available as https://www.ccnx.org/wiki/CCNx/CCNxReleaseFieldGuide, June 2012.

[11] VideoLAN Organization. VLC Media Player. Available as http://www.videolan.org/vlc/, October 2013.

[12] Project CCNx. Project CCNx : Command/Utility Documentation. Available as http://www.ccnx.org/releases/latest/doc/manpages/index.html, December 2013.

[13] Lan Wang, A K M Mahmudul Hoque, Cheng Yi, Adam Alyyan, and Beichuan Zhang. OSPFN: An OSPF Based Routing Protocol for Named Data Networking. Technical report, 2012.

[14] A K M Mahmudul Hoque, Syed Obaid Amin, Adam Alyyan, Beichuan Zhang, Lixia Zhang, and Lan Wang. NLSR: Named-data Link State Routing Protocol. In *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, ICN '13, pages 15–20, New York, NY, USA, 2013. ACM.

[15] openwrt.org. OpenWRT - Wireless Freedom. Available as https://openwrt.org/, December 2013.

[16] openwrt.org. Linksys (by Cisco) WRT160NL. Available as http://wiki.openwrt.org/toh/linksys/wrt160nl, December 2013.

[17] Project CCNx. Project CCNx : Wireshark Plugin. Available as https://github.com/ProjectCCNx/ccnx/tree/master/apps/wireshark, December 2013.

[18] Dave Täht. Package Repository for the CeroWrt Project. Available as https://github.com/dtaht/ceropackages-3.3/tree/master/net/ccnx, December 2013.

[19] CeroWRT Project. CeroWRT Project. Available as http://www.bufferbloat.net/projects/cerowrt/wiki, December 2013.

[20] Cheng Yi, Jerald Abraham, Alexander Afanasyev, Lan Wang, Beichuan Zhang, and Lixia Zhang. On the Role of Routing in Named Data Networking. Technical report, University of Arizona, UCLA, University of Memphis, 2013.

Appendix A

CCNx Quick Reference

Here we provide a quick reference for several CCNx commands and applications used in this work. We follow a similar section structure as that of Section 3.1.2.

A.1 CCNx Forwarding Tables

To add a route allowing a node `ccnx1` to forward Interest packets directed at content domain `ccnx:/content/files/`, the following `ccndc` command shall be used:

```
user@ccnx1:-# ccndc add ccnx:/content/files/ udp 172.16.1.101
```

This command will tell `ccnd` to forward any Interests for content matching the prefix `ccnx:/content/files/` to the IP address 172.16.1.101, over UDP.

A.2 Disseminating Content with CCNx

A.2.1 Usage of `ccnr` and `ccngetfile`

In CCNx, content sources use a specific repository application for storing content and make it available in CCNx networks by responding to matching Interests, `ccnr` [12]. The following sequence of commands initializes a repository and inserts a file into it, making it addressable via the name `ccnx:/CCN_REPO_DIR/file1.dat`:

```
user@cs:-# ccnr
user@cs:-# ccnputfile ccnx:/CCN_REPO_DIR/file1.dat file1.dat
```

The `ccngetfile` application [12] can then be used to release Interest packets to the CCNx network, querying for particular content, e.g. (assuming that `CCN_REPO_DIR` = `/content/files`):

```
user@ccnx1:-# ccngetfile ccnx:/content/files/file1.dat file1.dat.local
```

The command above releases Interests for the content `ccnx:/content/files/file1.dat` and saves the file locally as `file1.dat.local`. Being a CCNx application, it follows CCN's '1 Data packet per Interest' principle [3], but with pipelining of Interests, i.e. the application immediately releases N Interest packets into the network before the actual reception of Data packets. The actual behavior of this exchange (flow control) is analyzed in some of the tests specified in Section 3.3.

A.2.2 Usage of `ccnsendchunks` and `ccncatchunks2`

We provide examples of the use of both commands below, for an exchange of content `ccnx:/content/files/file1.dat` between nodes `ccnx1` (sender) and `ccnx2` (receiver):

```
user@ccnx1:-# ccnsendchunks -b 4096 ccnx:/content/files/file1.dat < file1.dat.ccnx1

(...)

user@ccnx2:-# ccncatchunks2 -p 31 ccnx:/content/files/file1.dat > file1.dat.ccnx2
```

With the combination of commands shown above, `ccnx1` gets ready to send Data packets, each one containing blocks of 4096 bytes of the file `file1.dat.ccnx1`. This content may be addressed via the content name `ccnx:/content/files/file1.dat`. On the other end, `ccnx2` starts sending batches of (at most) 31 Interest packets for the content `ccnx:/content/files/file1.dat`, saving the file as `file1.dat.ccnx2` when all the chunks are finally transmitted.

A.2.3 Usage of VLC plugin

E.g. for reproducing some content addressed as `ccnx:/content/files/video.avi`, located somewhere in the networks, VLC should be called in the following way (note the triple slash '/'):

```
user@ccnx1:-# vlc ccnx:///content/files/video.avi
```

Appendix B

Additional Measurements

B.1 Test 1 - CCNx Throughput and Overhead

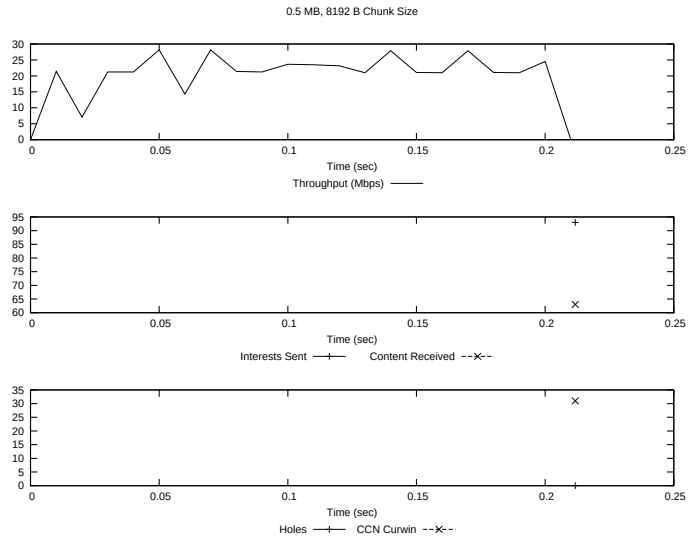


Figure B.1: Results for Test 1: Throughput in Mbps, as perceived by PC1, when retrieving a file of size 500 kB with a 8192 byte 'chunk' size. Statistics specific to the `ccncatchunks2` application are also shown.

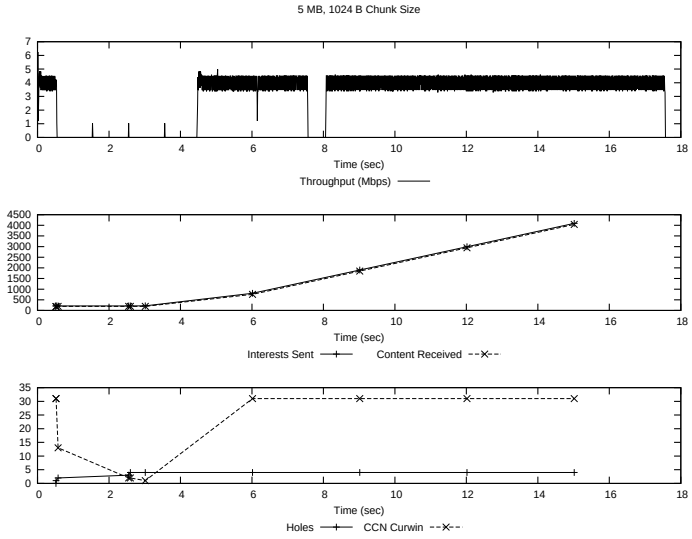


Figure B.2: Results for Test 1: Throughput in Mbps, as perceived by PC1, when retrieving a file of size 5 MB with a 1024 byte 'chunk' size. Statistics specific to the `ccncatchunks2` application are also shown.

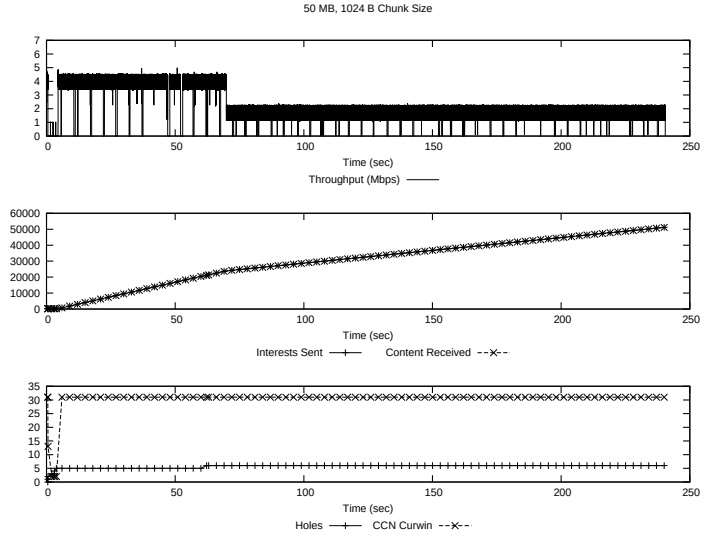


Figure B.3: Results for Test 1: Throughput in Mbps, as perceived by PC1, when retrieving a file of size 50 MB with a 1024 byte 'chunk' size. Statistics specific to the `ccncatchunks2` application are also shown.

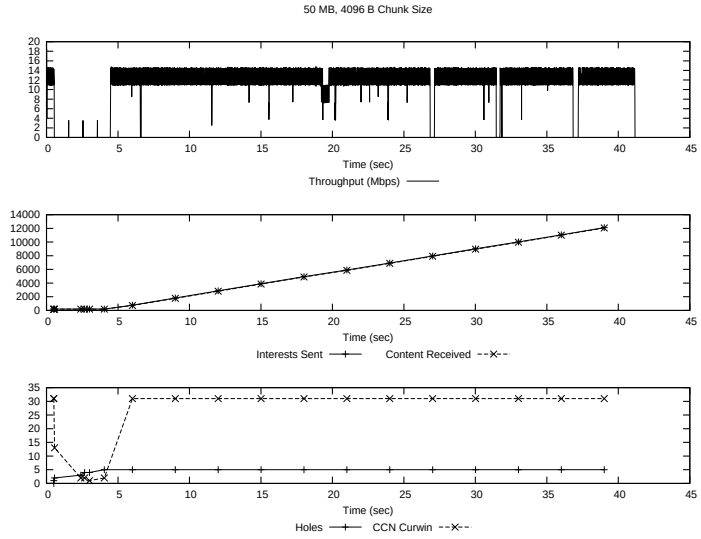


Figure B.4: Results for Test 1: Throughput in Mbps, as perceived by PC1, when retrieving a file of size 50 MB with a 4096 byte 'chunk' size. Statistics specific to the `ccncatchunks2` application are also shown.

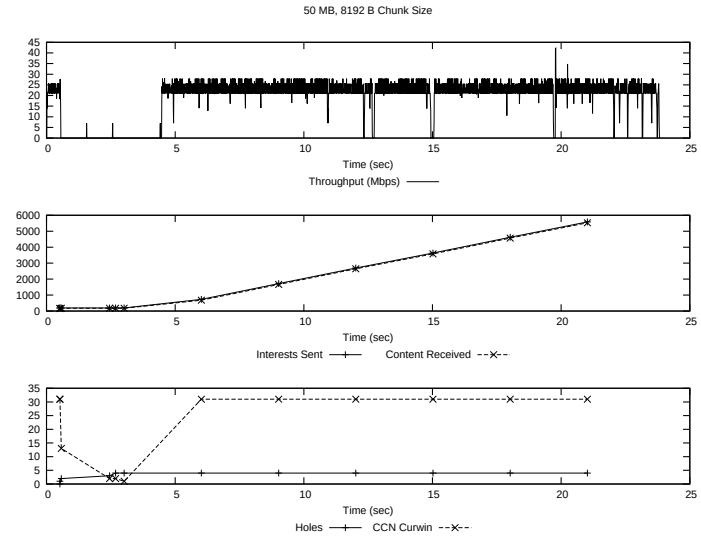


Figure B.5: Results for Test 1: Throughput in Mbps, as perceived by PC1, when retrieving a file of size 50 MB with a 8192 byte 'chunk' size. Statistics specific to the `ccncatchunks2` application are also shown.

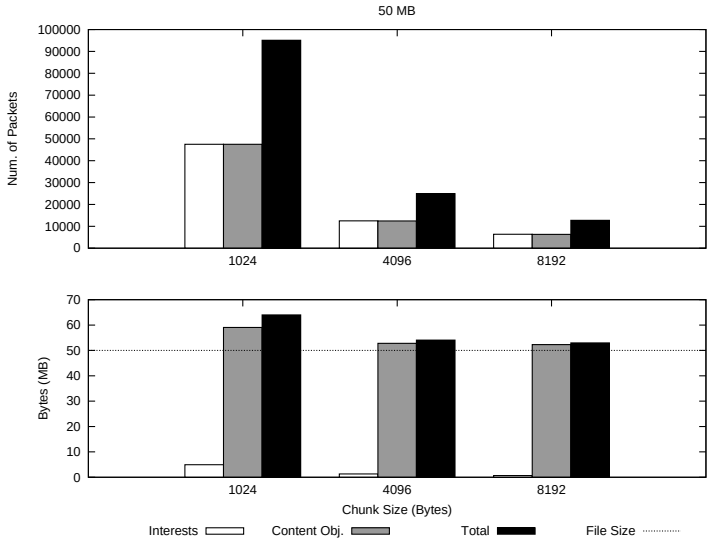


Figure B.6: Results for Test 1: Packet and byte quantities, measured at PC1, for a file size of 50 MB.

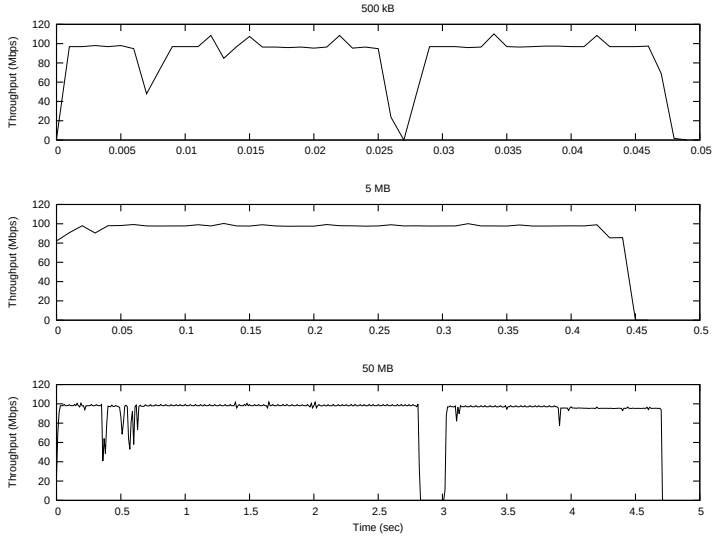


Figure B.7: Results for Test 1: Throughput in Mbps, as perceived by PC1, when retrieving files of sizes 500 kB, 5 MB and 50 MB, over FTP.

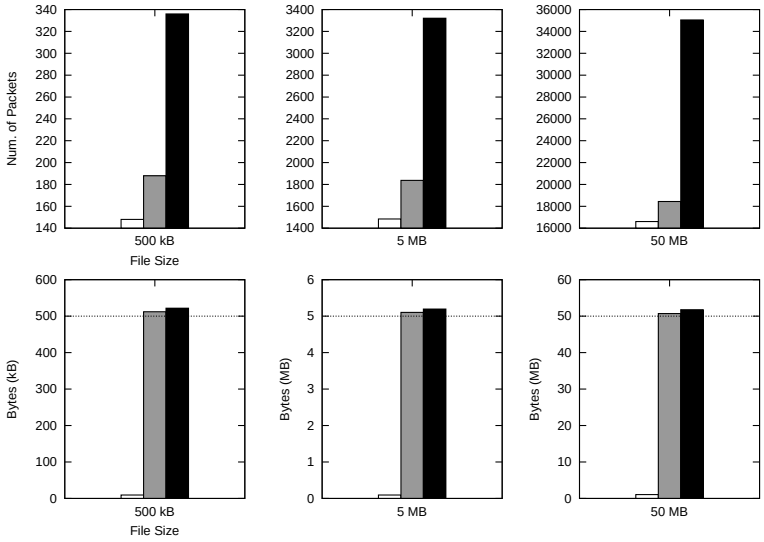
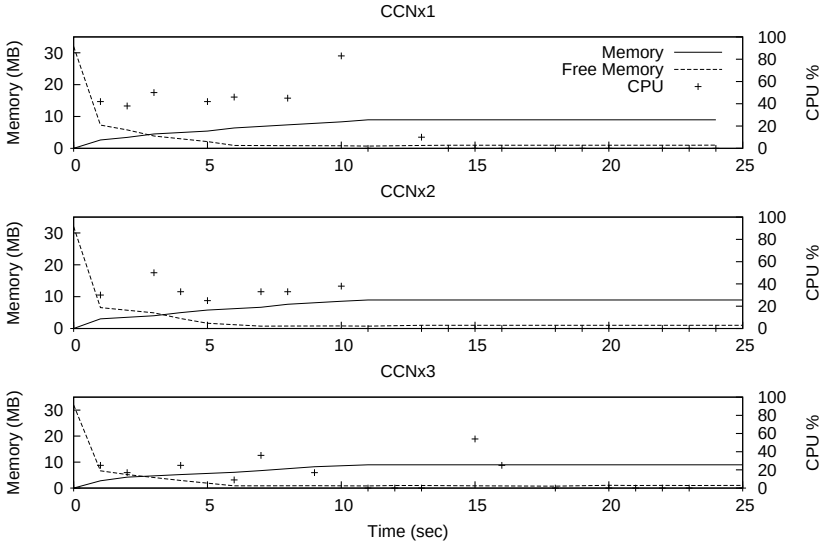


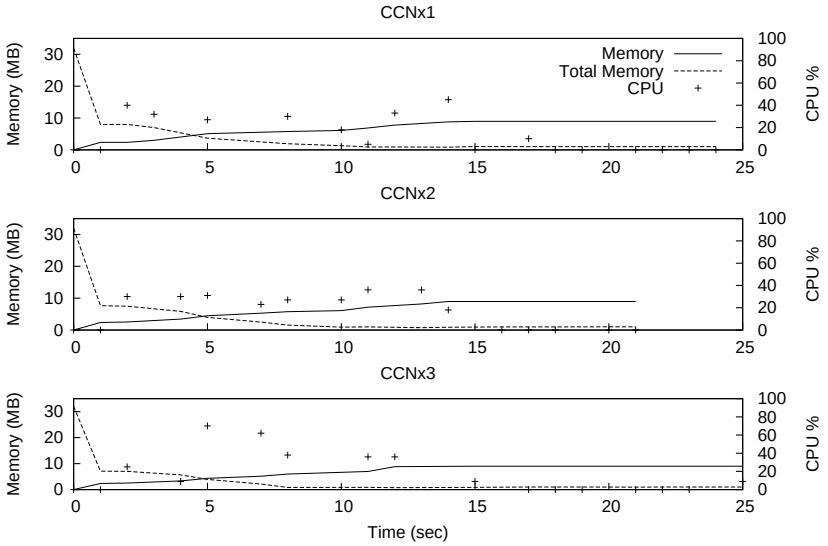
Figure B.8: Results for Test 1: Packet and byte quantities, measured at PC1, for files of sizes 500 kB, 5 MB and 50 MB, over FTP. The white bars represent packets sent from the FTP client side, gray represent packets sent from the FTP server side, while the black bars represent the total number of exchanged packets.

B.2 Test 2 - CCNx Multihop Forwarding

B.2.1 Test 2.1 - CCNx Multihop Forwarding (File Transfer)

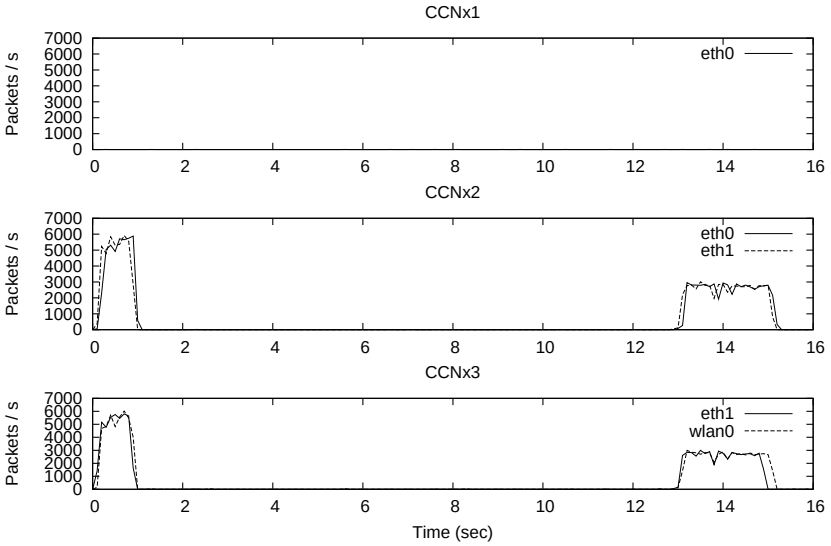


(a)

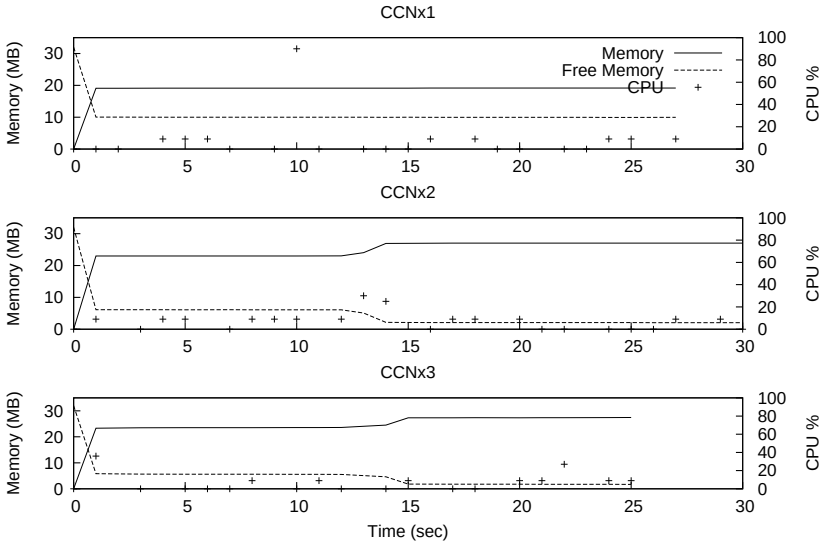


(b)

Figure B.9: Results for Test 2.1: CPU and memory utilization at all CCNx nodes, during the transfer of a file of size 5 MB, for both non-overlapping (a) and overlapping (b) cases. Regarding the memory values, the 'memory' line corresponds to the amount of RAM occupied by the `ccnd` process, while the 'free memory' corresponds to the amount of free memory in the system.



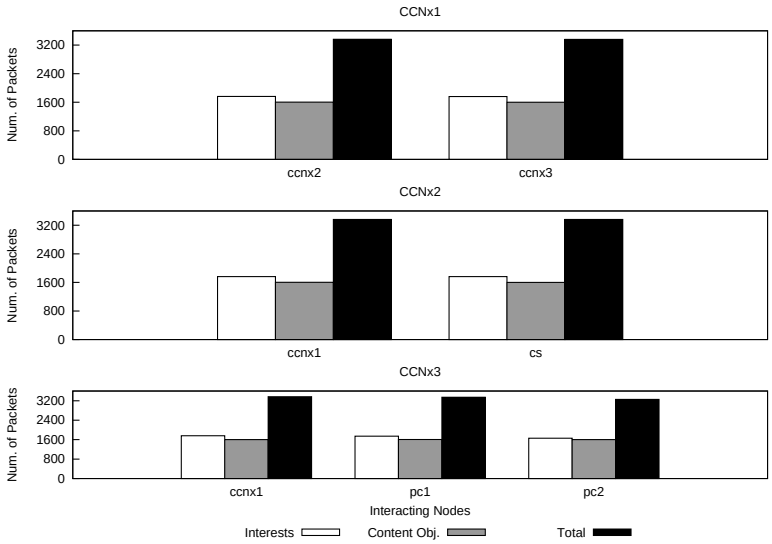
(a)



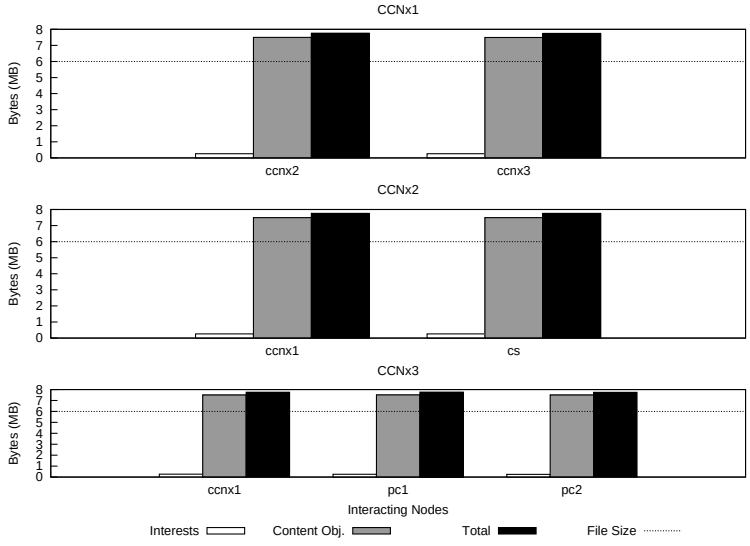
(b)

Figure B.10: Results for Test 2.1: Network load (a) and CPU and memory utilization (b) at all routing nodes, during the transfer of a file of size 5 MB over FTP, for a non-overlapping case. Regarding the memory values, the 'memory' line corresponds to the amount of RAM occupied by the `ccnd` process, while the 'free memory' corresponds to the amount of free memory in the system.

B.2.2 Test 2.2 - CCNx Multihop Forwarding (Video Streaming)

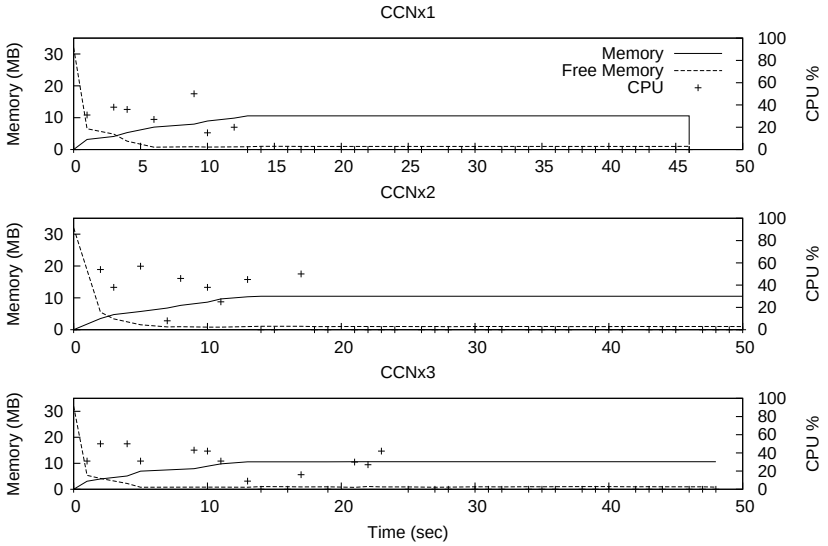


(a)

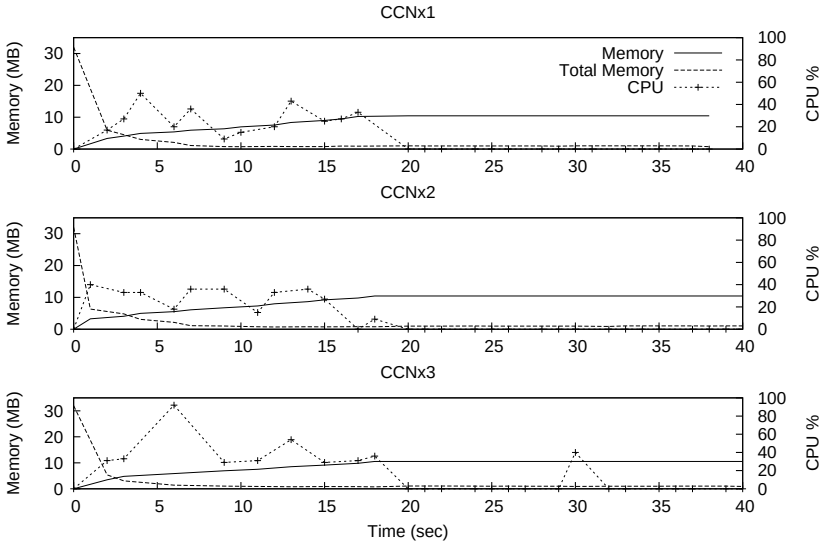


(b)

Figure B.11: Results for Test 2.2: Packet (a) and byte (b) counts (Interest and Content Objects) registered at each CCNx forwarding node, exchanged between the respective peer elements. This test involves the streaming of a video of size 6.3 MB and 8 seconds duration from the content source CS to nodes PC1 and PC2 (see Figure 3.1).



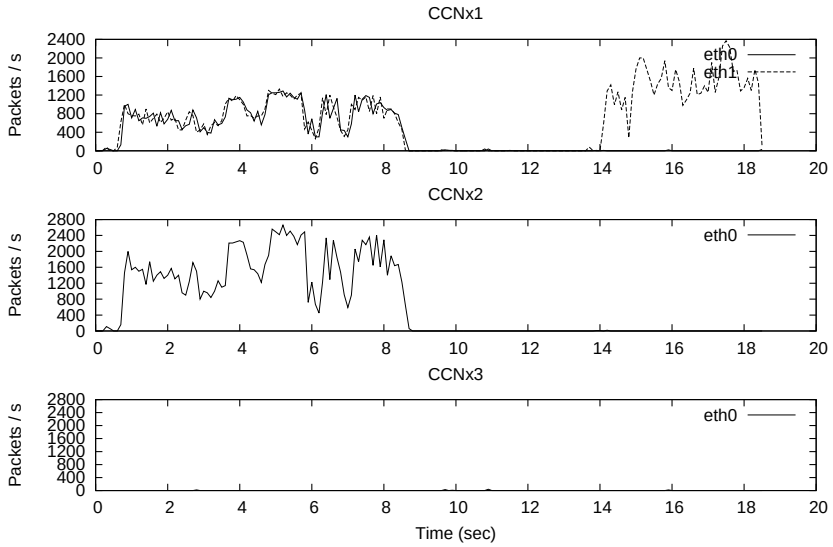
(a)



(b)

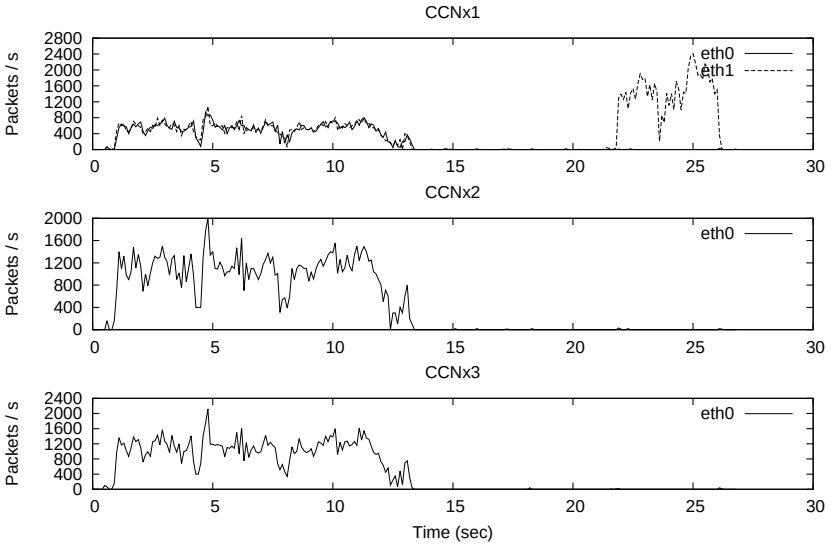
Figure B.12: Results for Test 2.2: CPU and memory utilization at all CCNx nodes, during the streaming of a video file of size 6.3 MB and 8 second duration, for both non-overlapping (a) and overlapping (b) cases. Regarding the memory values, the 'memory' line corresponds to the amount of RAM occupied by the `ccnd` process, while the 'free memory' corresponds to the amount of free memory in the system.

B.2.3 Test 2.3 - CCNx Multihop Forwarding (Multiple Paths)

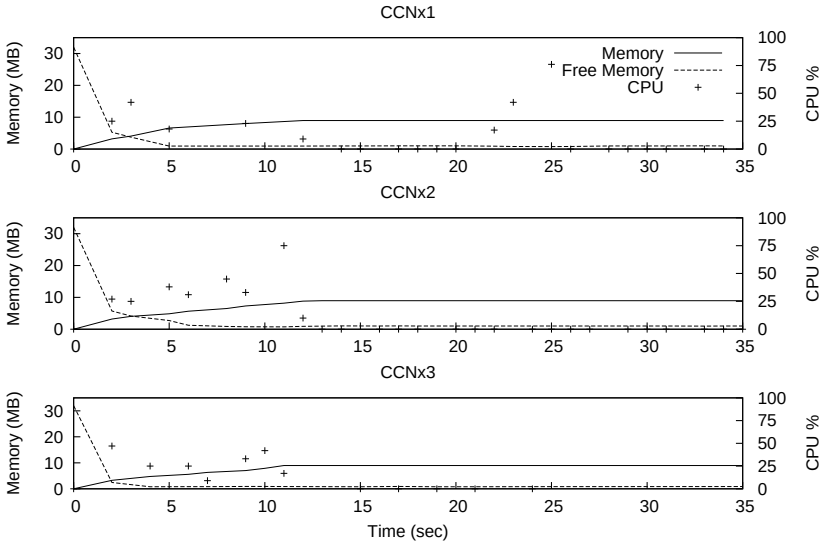


(a)

Figure B.13: Results for Test 2.3: Network load at all CCNx nodes, during the transfer of a file of size 5 MB (non-overlapping case), using the 'short' path indicated in Figure 3.3.

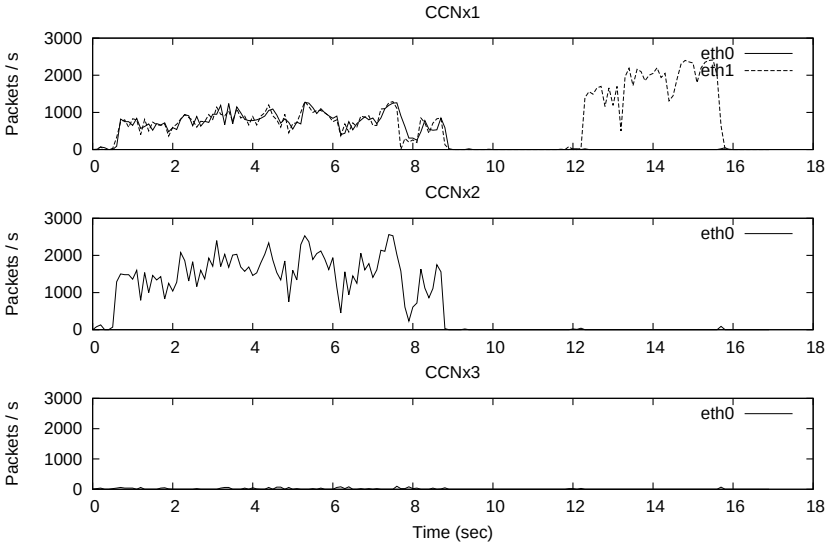


(a)

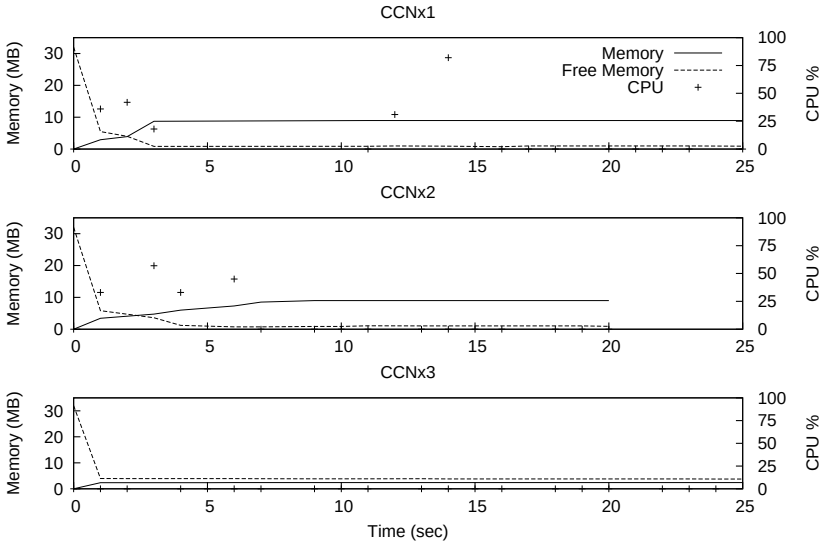


(b)

Figure B.14: Results for Test 2.3: Network load (a) and and CPU and memory utilization (b) at all CCNx nodes, during the transfer of a file of size 5 MB (non-overlapping case), using the 'long' path indicated in Figure 3.3. Regarding the memory values, the 'memory' line corresponds to the amount of RAM occupied by the `ccnd` process, while the 'free memory' corresponds to the amount of free memory in the system.



(a)



(b)

Figure B.15: Results for Test 2.3: Network load (a) and and CPU and memory utilization (b) at all CCNx nodes, during the transfer of a file of size 5 MB (non-overlapping case), using both 'short' and 'long' paths indicated in Figure 3.3. Regarding the memory values, the 'memory' line corresponds to the amount of RAM occupied by the `ccnd` process, while the 'free memory' corresponds to the amount of free memory in the system.