Hochschule für Angewandte Wissenschaften Hamburg
Hamburg University of Applied Sciences

CCNx meassurement testbed implementation

Markus Vahlenkamp

Master project report

Fakultät Technik und Informatik
Studiendepartment Informatik

Faculty of Engineering and Computer Science
Department of Computer Science

Contents

List of Figures

Listings

1 Introduction

The amount of data transferred through the Internet is steadily increasing, all the more because of the enhanced availability of wired and wireless broadband access as well as the rising amount and quality of content. It is used to deliver and distribute masses of all sorts of content these days. Compared to what the Internet was built for, namely the direct interconnection of hosts, its current architecture is assumed inefficient for the dissemination of today's web content. Websites, videos, pictures, and so on are repeatedly transferred throughout the Internet, from content source up to the content consumer.
Consumers of all these kinds of content do not matter where the actual content is acquired from, they are also not interested in communication with a particular network node, but in receiving the desired content.
Plenty of nodes are already available, spread across the globe, which host, replicate and cache the content that is of interest for consumers. To utilise these nodes for an efficient content distribution, many techniques and workarounds like load balancers, DNS- and HTTP-redirection have been introduced. But all these Content Delivery Network (CDN) related techniques are applied on top of the actual host-to-host communication concept of the Internet. At the present researchers hold the opinion that because of the fundamental shift towards content centricity, the basic concept of networks should reflect this through enhanced content dissemination capabilities.
Current research includes for instance routing on content names instead of hostnames, content caching directly implemented into the routing infrastructure and all this backed by the publish / subscribe paradigm for content acquisition [1].
Different projects implemented or simulated the network architecture concepts they developed. These prototypes follow the concept that is generally known as Information-Centric Networking (ICN). The systems are further used to analyse the behaviour as well as to identify the pros and cons of the different approaches.
Within this project we will take the most famous of these implementations, namely Palo Alto Research Centers (PARCs) [2] CCNx project, deploy it within our testbed and analyse the underlying concept and its implementation. The goal is to test if scalability, robustness, performance or security flaws exist within the Named Data Networking (NDN) [3] concept, the concept that represents the CCNx projects core architecture. We want to stress, that our analysis is based on analytical pre-considerations and focuses on conceptual issues instead of implementation related bugs or alike. The present work is directly related to our previous publications [4, 5] but widens the spectrum and also gives a bright inside into our measurement processes and the testbed setup.
In the remainder of this document we will give a short overview of the open issues we see in the field of ICN regarding performance, stability and security in Section 2. A survey of the CCNx components and their mode of operation is given in Section 3. Afterwards in Section 4 we will expose detailed information about the testbed components and the implemented

setup we use to gather our measurement values. We expose and illustrate out explicit measurements and their results in Section 5, followed by the conclusion that our measurements lead to in Section 6.

2 Problem statement

Different ICN proposals have been developed in the past, all implementing the general idea of ICN. Among them are for instance TRIAD [6], DONA [7], NDN [3, 8], PSIRP / LIPSIN [9, 10] and NetInf [11]. They all take slightly different approaches in one or the other design choice, but aim for the general ICN goals such as content caching, routing on names as described in [12, 1]. ICN inherently introduces an enhanced content awareness to the network. The network needs to maintain information about where particular pieces of content are available in conjunction to today's networks which just need to maintain the information where particular subnets (aggregations of individual nodes) are located.
The step of exposing knowledge about content into the network infrastructure itself places a higher burden on the network infrastructure. It has to keep track of pieces of content and their location except just knowing how to reach groups of nodes or particular nodes.
Through this shift and increase of responsibility towards the network infrastructure new challenges and threads arise. The following non-exhaustive list recaps the security and performance challenges we already elaborated in [5].

Resource Exhaustion A massive generation of content subscriptions or publications, caused by for instance malicious misuse or misconfiguration, will result in an extensive utilisation of memory and processing resources. This could likely lead to a Denial-of-Service (DoS).

State Decorrelation Through the asynchronous nature of the publish / subscribe based data transmission, a decorrelation of the distributed states may lead to service disruption and unwanted traffic flows.

Path & Name Infiltration Through the publishment of a name or name prefix within the network subscription messages are attracted. This can be used to blackhole subscription messages or to start man-in-the-middle attacks by issuing falsified publication messages. The fact that distributed cached copies need to be registered to optimally use the caching infrastructure, as posed in [13], makes the genuineness validation of publishments even harder.

Cache Pollution Through the use of content caches within the routers, the network aims to perform better while disseminating the same content again and again. By spoiling the cache relevance, the usefulness and thus the performance of the cache are vulnerable.

Further randomly filling the cache also leads to an increase in control traffic and routing table maintenance.

Cryptographic Breaches Long lived signing keys combined with large amounts of published data provide increased opportunities to compromise the cryptographic credentials used to secure the authentication of the publisher as well as the integrity of the content itself.

Further resources that elaborate the security risks and threads of ICN are [14, 15].
In this paper, we will especially focus on the Resource Exhaustion and State Decorrelation case. Our objective is to figure out how severe these threads to security and performance are in an actual testbed deployment.

3 NDN / CCNx overview

Prior to introducing our testbed in Section 4, we will give a short introduction to the NDN architecture in Section 3.1 and available software components of the actual CCNx implementation in Section 3.2.

3.1 Architecture

The CCNx protocol utilises the publish / subscribe paradigm for content dissemination. Content that should be made available needs to be published within CCNx, such that content consumers will be able to retrieve the content through issuing subscription messages, known as Interests. Every piece of content in CCNx is made available through the use of certain names. These content names are used to identify and locate the content, thus in some way they take over certain parts of IPs responsibilities of today's networks. Like in IP networks every content router needs to know where particular parts of the namespace are located. This information is distributed between name-based content routers through the use of a name-based routing protocol [16].
The name-based routing information populates the Forward Information Base (FIB)-table of the name-based router and is then used to route incoming subscriptions towards the publisher.
Figure 1 depicts the architecture of a CCNx based router. Besides the FIB-table the router holds the Pending Interest Table (PIT) and the Content Store. The PIT is used to store Interest messages, when they are passed on to the next content router or an application. Interests in fact are the representation of a subscription message of the NDN concept that is propagated from subscriber towards a content source. The main purpose of the PIT, though is to aggregate content requests. For Pending Interest messages, requesting the

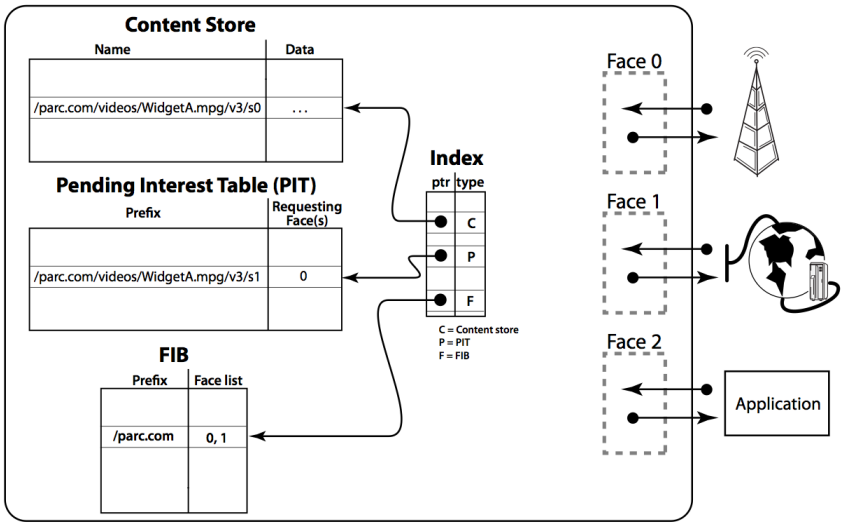


Figure 1: CCNx router overview [3]

same piece of content, at most one Interest is forwarded towards a neighbour router. Subsequent Interests that arrive at the content router, while an active Interest is pending, are noted in the PIT but their forwarding is suppressed. When the content chunks subsequently arrive at the router, following the reverse path of the Interest message, they are delivered towards every requesting consumer that previously sent an Interest for the particular chunk. This behaviour apparently results in a per chunk multicast like dissemination behaviour.
The content store is used to cache the received content to be able to deliver it to consumers that subsequently issue an Interest for the particular piece of content. It also allows the underlying mechanism to evolve from synchronous to asynchronous multicast. Without the content store the multicast like behaviour will just appear when Interests for the same content chunk arrive at the time at least one other Interest requesting the same content is already pending. The Content Store alleviates this timely coupling, by locally storing the acquired content for later requests, resulting in an asynchronous per chunk multicast dissemination.
Faces are the generalization of an interface. It may be a connection to various network nodes or to an application.
Due to the publish / subscribe approach, the communication is always driven by the receiver. Through the generation of an Interest, the client announces its willingness to receive a particular piece of content. This Interest is send to a content router that processes the Interest message in the following way [2].

1. Content store lookup is performed. If a content object matching the Interest is found within the content store, it is transmitted out the arrival interface of the Interest message. The Interest message is then dropped because it is satisfied and no further processing in needed.

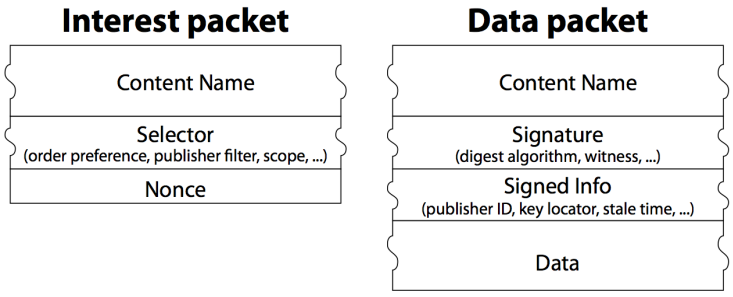


Figure 2: CCNx packet structure [3]

2. PIT lookup is performed. If a Pending Interest matching the content name is already pending, meaning that an Interest for that piece of content is already forwarded to neighbouring routers, the incoming face is just added to the corresponding PIT entry and the Interest message is discarded.

3. FIB lookup is performed. A corresponding prefix for the name of the Interest packet is looked up in the FIB-table. If a matching prefix is found, an entry is created within the PIT describing the Interest. Subsequent the Interest is forwarded out one or more faces noted within the FIB.

4. No FIB match found. The node has no chance to satisfy the Interest, thus the Interest message is discarded.

These steps are performed on every content router on the way up to a source of the name. Whenever a particular piece of requested content arrives at a content router, a PIT lookup is performed to find all faces a corresponding Interest was received on. The resulting list of faces is used to transmit the data chunks towards all subscriber that issued an Interest for that particular piece of content. Once the Pending Interest is satisfied, the PIT entry is removed and the content object is stored within the local nodes Content Store for future requests.
The PIT entries use a soft-state model. If they are not satisfied or refreshed within a certain period of time the PIT entries are dropped automatically.
Figure 2 for the sake of completeness shows the structure of an Interest message as well as a data packet.

3.2 Software components

The CCNx bundle comprises various different programs. CCNx core library implementations exist in C or Java as well as for the Android OS.

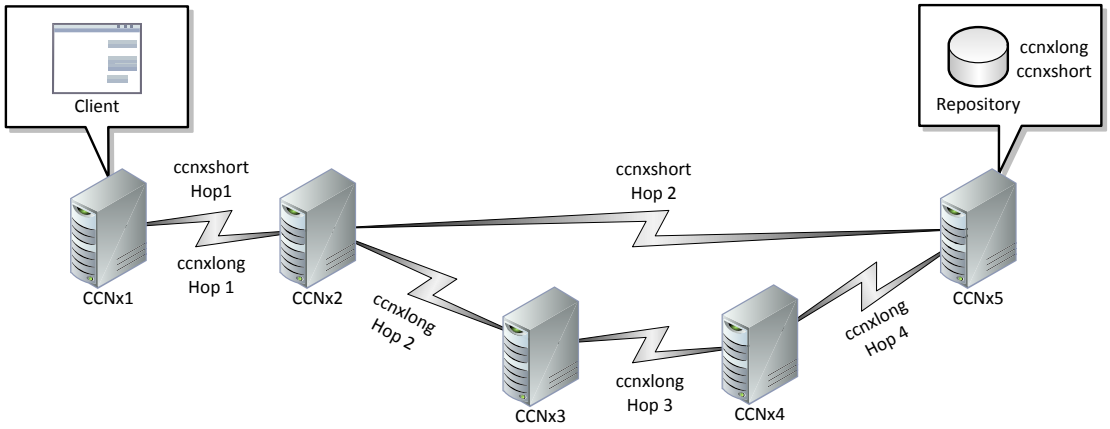


Figure 3: Testbed logical topologie

The bundle further consists of various sample applications that rely on CCNx. A chat application (ccnChat) is part of the bundle as well as a file proxy and a repository.
The file proxy is used to serve files located within the file system to CCNx clients, whereas the repository daemon serves as a persistent storage for already chunked content. Further the bundle consists of a VLC-Player[1] plugin, that allows the media player to directly interact with the routing daemon to request and playback for instance audio or video content via CCNx.
The last part of the bundle we want to mention is the wireshark[2] packet dissector. It is used to ease the retrieval of CCNx related information from packet captures and will also heavily be used in our testbed implementation.

4 Measurement

Throughout this chapter we exhibit our testbed setup in detail. The topology of the testbed is described in Section 4.1 followed by a description of the script backed workflow used for metering and post-processing. Finally the resulting measurement data is presented in subsection 4.3.

4.1 Testbed topology

Our testbed consists of five CCNx nodes named CCNx1 to CCNx5. These nodes are all virtualised linux machines running on top of a VMware ESX server. We choose such virtual-

[1] http://www.videolan.org/vlc/
[2] http://www.wireshark.org

ised environment to be able to simply change the hardware configuration of the machines, a feature we excessively rely on in our measurement cases. Through this eased configurability we are able to simulate homogeneous as well as different heterogeneous node constellations. In the basic setup all nodes are equipped with 3 GB of RAM, 2 x 2.4 GHz Cores and an virtual Ethernet interface connected to a standard VMware vSwitch. For the operating system Debian Linux in version 6.0.4 was chosen.
Regarding CCNx, we focused on the repository application as the source of content and the `ccnd` routing daemon for inter-node connectivity.
Each of the nodes depicted in Figure 3 is running a local `ccnd` process. The routing information of all these nodes is configured manually. This means static routing entries have been configured in the testbed setup phase instead of using a dynamic routing protocol. Within the setup, there exist two distinct routes, referring to the names ccnxshort and ccnxlong. We choose the names regarding the hop count from node CCNx1 to node CCNx5. While the route ccnxshort traverses just two hops, from CCNx1 through CCNx2 towards CCNx5, ccnxlong extends through all of the five nodes from CCNx1 to CCNx5.
CCNx1 in addition to the `ccnd` runs the client software that is using the CCNx Java API to request and retrieve the content used for measurement.
CCNx5 is also hosting an additional process, the content repository. The repository is prefilled with content, matching the namespaces ccnxlong and ccnxshort. Thus the client can request the files which will then be delivered on the reverse path the Interest took up to node CCNx5. Through the connection from the repository to the local `ccnd` the routing daemon running on CCNx5 knows that these namespaces are served by the repository and thus passes on the associated Interests to the repository.
Because of the distributed nature of the setup a common timereference is needed to be able to timely associate different events happening on various nodes in the testbed. Through the use of the virtualisation server we have been able to accurately synchronise the clocks of the virtual machines with the host clock of the VMware ESX server, such that the accuracy fits our requirements of below 1 second divergence. The timely resolution of our measurement setup is limited through the use of the Unix timestamps, which have a resolution of one second. The resolution of one second is considered sufficient to examine the test scenarios and answer our research questions.
To be able to remotely execute commands in batch mode, Secure Shell (SSH) [17] backed by public keys is used. This way the centrally executed control script is able to execute the necessary commands on the testbed nodes and thereby control the measurement run.

4.2 Measurement workflow

We created a script that guides through the measurement and measurement data post-processing phase. The workflow created by this script is illustrated in Figure 4. Further Figure 5 illustrates the post-processing steps, that the logging files undergo. If not explicitly

denoted, the script executes the tasks on every node included in the testbed. Within the environment, the general measurement script is executed on the first node, CCNx1. It may be worth mentioning, that this does not impact the measurement because the script is paused and waiting for user input during the time the measurement is running. When we use "control node" in the reminder of this document we refer to CCNx1.
When the script is started, it tries to restart the CCNx routing process (`ccnd`) running on every node that is involved in the measurement *(1)* . This is due to previous runs that may have influenced the running `ccnd` instance, in a way that the memory consumption or the cpu utilisation is already higher than it is in a clean state. Thus to measure just the impact created by the actual measurement run, the `ccnd` is restarted. In case it isn't already running it is started.
Some of the processes responsible for logging just append their log information to the previously created files. Thus the routing process logfile and the file transfer statistics file are erased in step *(2)* and *(3)*. Afterwards the packet capture process is started on every node *(4)*. It is configured to capture just 500 Bytes of every packet that is transmitted between each node and its successor on the way up to the repository node. Whereas the last node CCNx5 has no such successor, hence it is measuring the connection towards CCNx4 so to say the opposite direction.
In step *(5)* the logging process for the `ccnd` routing daemon is started. Subsequently the script stops and asks the user to stop the measurement run through pressing the 8 key on the keyboard. This is when the external process that utilises the CCNx infrastructure is to be executed *(6)*. After that process is done, the user has to confirm that the measurement should be stopped by pressing the 8-key *(7)*. Subsequent the routing daemon logging process as well as the packet capture process are stopped *(8, 9)*.
At this point the active measurement is completed and the post-processing of the metered data takes place. In step *(10)* the script connects to every node involved in the measurement run and prints out the statistics of the packet capture. Following this in step *(11)*, every routing process logfile is also analysed, while still residing on the particular nodes. The count of routing process logfile lines that hold invalid data is printed out. Invalid lines are those, where certain values are missing because of timeouts or alike. These invalid lines are a direct result of an overload of the `ccnd` routing daemon. The `ccndstatus` command times out while querying the daemon and thus no data is available for that request. Nevertheless the data regarding memory consumption and cpu utilisation as well as the timestamp is written into the log file. If too many lines contain those invalid logging lines, the measurement may need to be rerun. The count of lines containing invalid data are determined and displayed. On the basis of this information displayed in steps *(10)* and *(11)* it is up to the user to decide if the post-processing of the data should continue or not. It may be the case that to much packet have been dropped from the capture so the results can be looked upon as inadequate. The same may be the case with the amount of dropped routing process logfile lines. If continued though the script prompts for a name for the measurement run in step *(12)*. This name is

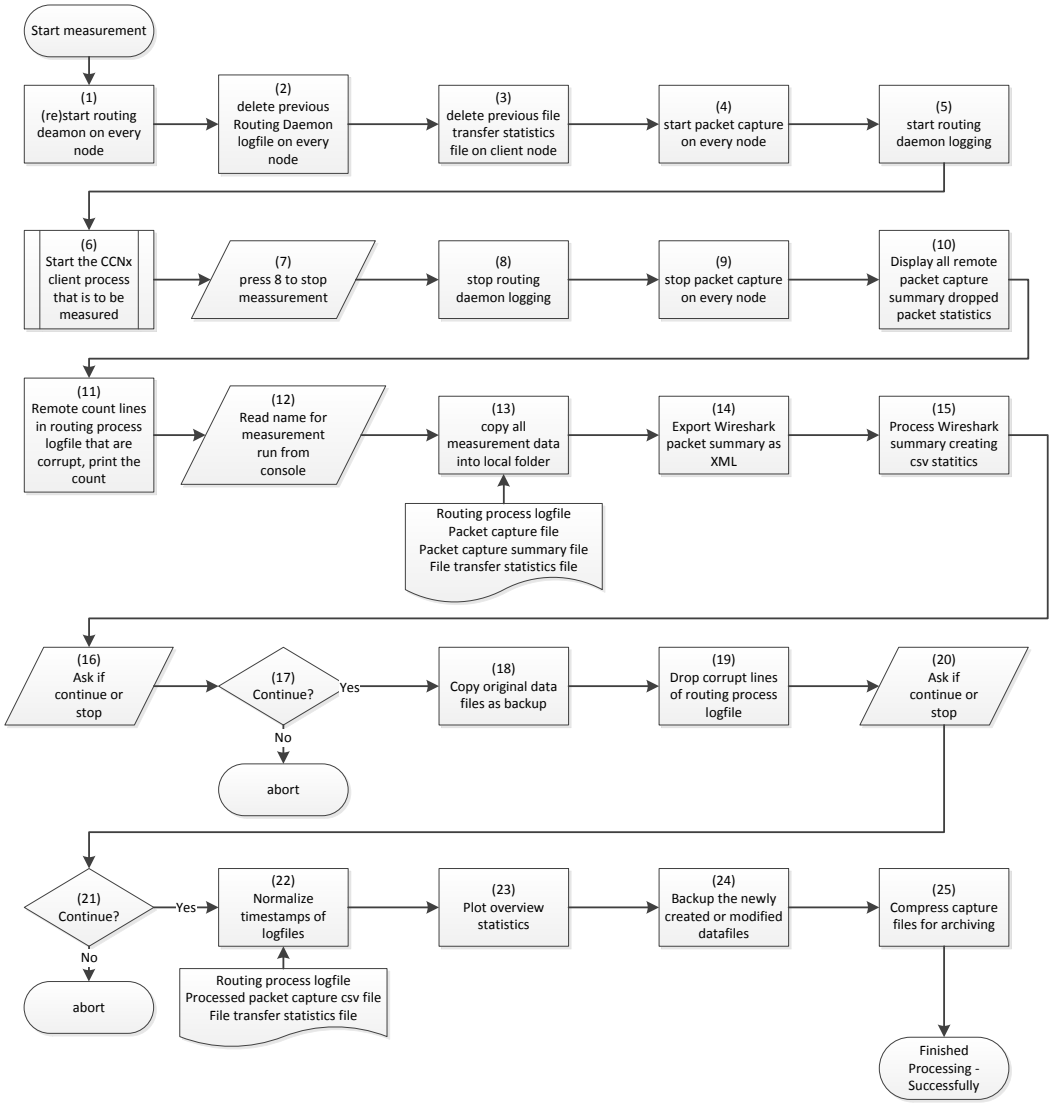


Figure 4: Workflow illustration of the semi-automated measurement process

used as a prefix for all the log files collected on the node running the measurement script, through this name it is easy to uniquely identify different runs. Subsequently the routing process logfile, capture file and the capture statistics from all nodes as well as the file transfer statistics file from the requesting node are copied into a local directory on the control node *(13)*. From this point onwards all commands and processes are executed just on the control node.
The collected capture files are processed *(14)* such that the packet capture summary xml-file is extracted. This summary file is hereafter used to gather the informations about throughput, goodput as well as the Interest retransmission count *(15)*.
Thereafter this, the script hits a breakpoint in step *(16)* and the user has the opportunity to interrupt further execution of the script if something went wrong in previous steps or just continue the processing *(17)*. Because in the following steps the gathered data will be altered, a backup of the original data is created in step *(18)*, so that in case of an error in the subsequent processing steps the original measurement data is not lost through a faulty modification of files. In step *(19)* though the invalid lines of all routing process log files are finally dropped. This is a valid step, because in step *(11)* was shown how many lines are invalid and thus has acknowledged that the amount is negligible and the processing should continue. Nevertheless is the amount of dropped lines is outputted once again and the user is asked once again *(20)* if the whole processing should be aborted or continued *(21)*. If the processing continues, the timestamps contained in the routing process log, processed packet capture statistics and the file transfer statistics file are normalized. The time-wise normalization *(22)* is performed to convert the lowest timestamp occurring in all metered logging files to value one, hence to represent the time since the start of the measurement instead of the Unix timestamps[3]. With this step the data processing is done. Following just some final clean-up tasks are performed. An overview figure for each node is created to visualise the measured data for quick review *(23)*. As depicted in Figure 14 the figure shows six graphs, containing CPU and Memory Utilization, amount of Pending Interest, Data Goodput in Interest and Data direction and the Interest Retransmission count. The second graph (Figure 15) that is created illustrates the transfer times of the files that have been transferred throughout the measurement run. Finally the plots are composed in an measurement overview pdf containing all graphs and the measurement name. A sample output can be found in the appendix (Figure 16). Then the newly created and altered files are copied *(24)* besides the backuped files from step *(18)*. At last the capture files in the backup folder are compressed *(25)* to be able to store them as efficient as possible for revision purpose. Through this, there still is the possibility to recheck or extend the analysis later on.
Further processing and analysis of the processed data is subsequently accomplished by the use of other special purpose tools.

[3] seconds since January 1st 1970 00:00:00 UTC

4.3 Measurement components

Throughout our measurement runs we create different files containing measurement data. Figure 5 shows these datafiles that are created for each run as well as the data centric workflow these files undergo. Following we describe the different measurement files in detail and how they are processed.

Routing process log The routing process measure file includes the Memory Consumption and CPU utilisation of the `ccnd`. This is to deduce the performance and scalability of the routing mechanism. Further the statistic values that the `ccnd` process provides are also collected. These informations include the Interest names the `ccnd` is aware of, pending, propagated and noted Interests. All this data is derived and aggregated along with the time of the retrieval. An example of the output is shown in Listing 1 on page 23.

Packet capture The traffic flowing between two neighbouring nodes regarding the topology depicted in Figure 3 is captured for the purpose of network level analyses, including for instance data throughput and retransmission rate. Because of performance reasons we limited the amount of captured data per packet to 500 Byte. Nevertheless we have been able to deduct the size of every packet because of the header values, reflecting the original packets length, thus we do not need the whole packet.
We post-processed the capture file by extracting the packet level summary information into an psml file (see Listing 5 page 24). That xml-file is further processed to extract the throughput and retransmission statistics listed in Listing 2 on page 23.

Capture statistics file The capture statistics file (Listing 4 page 24) is collected from all the nodes just to display it once. The important information is the included amount of dropped packets. If too much packets have been dropped the measure is useless, because the throughput calculation and maybe also the amount of retransmits are falsified. The measurement run needs to be redone. Hence the file is just used to display it and let the user decide how to continue.

File transfer statistics The file transfer statistics file (Listing 3 page 23) is created by the client program. Thus it just exists on CCNx1, the node acting as the sink of the data transfer.
Every single line contains the necessary information of the transfer of one file. The file url, the time that the file transfer started, the time it finished as well as the size of the file.

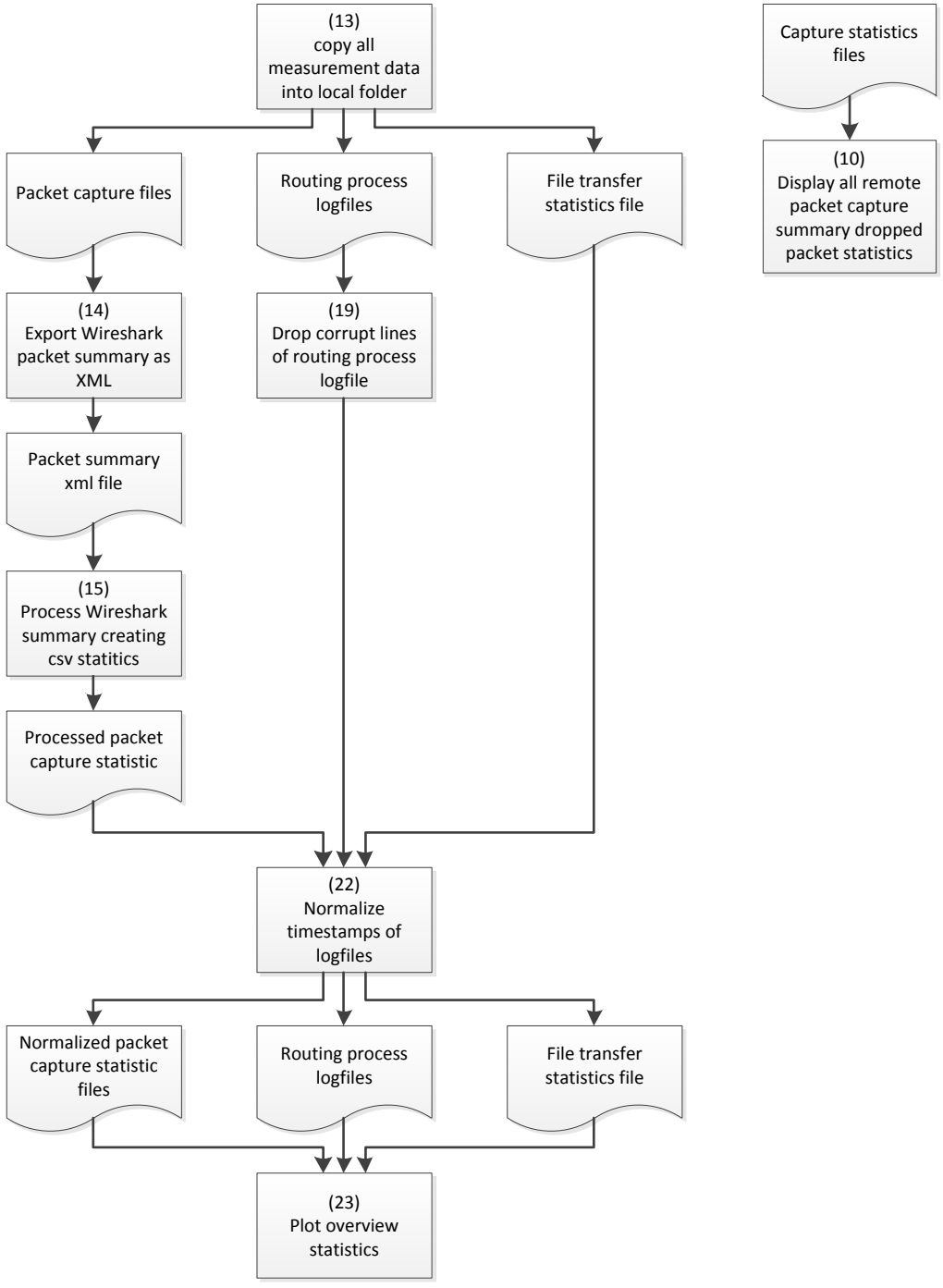


Figure 5: Datacentric workflow that the measurement results undergo

5 Results

We utilise our testbed described in Section 4 to perform various tests with the aim to examine the effects of the afore mentioned threads of Resource Exhaustion and State Decorrelation. Starting by taking a look at Resource Exhaustion through the bulk emission of Interests we continue with a deeper analysis of performance flaws in overloaded systems.
The remainder of this chapter is segmented into two subsections. Basic 5.1.1 and Extended Experiments 5.2. In the Basic Experiments we take a closer look at the Resource Exhaustion case and the behaviour of CCNx in case of rising load through an increased amount of chunk requests. All this is done through the use of the three node route ccnxshort. The Extended Experiments on the other hand utilise the five node route ccnxlong. In this part we focus our attention on content router heterogeneity and how it influences the content routing states.

5.1 Basic experiments: Resource exhaustion

For the Basic Experiments we used the short chain of content router nodes consisting of the three nodes CCNx1, 2 and 5 as illustrated in Figure 3. The connection between the content routers is established through the use of TCP connections.
At first we take a look at a case where Interests are issued for non-existing content before moving on to a real content dissemination scenario.

5.1.1 Resource exhaustion

To explore the behaviour of CCNx in cases of an overwhelming arrival of Interests, we made CCNx1 issue 2000 Interests ever 6 seconds until an overall of 150.000 Interests linger in pending state. The requested content in this case matches the prefix that follows the ccnxshort route of Figure 3 thus the Interests are forwarded towards CCNx5. Instead of requesting data that is available non-existing content is requested. This leads to the situation that PIT entries will not be cleared by the arrival of content chunks but stay alive until the soft-state timer times out. A timeout of the soft-states is also prevented by the Interest Retransmission mechanism that is used by CCNx to keep the PIT entries alive. Figure 6 depicts the resulting Pending Interest, CPU Load and Memory Consumption values of CCNx2, the first-hop router. It shows that the CPU Load is at a 100% when $\approx$ 120.000 Interest are received. The Pending Interest count is not rising any further indicating that the router operates at its performance limit. The content router is not able to handle the imposed load. This condition does not even change in time because all of the 150.000 Interests are refreshed after the refresh timer expires on the subscriber node. The load is not decreasing until the run is manually interrupted.
This behaviour illustrates a severe issue of CCNx. Through issuing masses of Interests the network infrastructure is easily overloaded in terms of local resources (CPU, Memory) result-

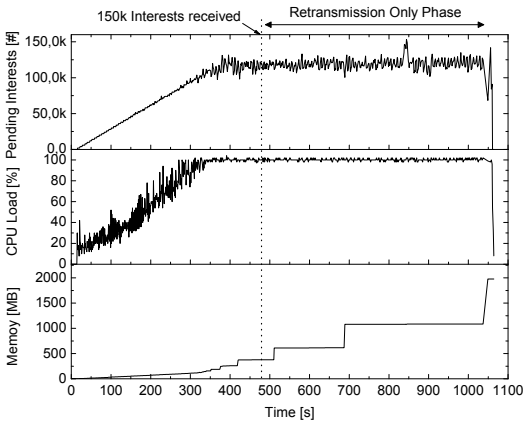
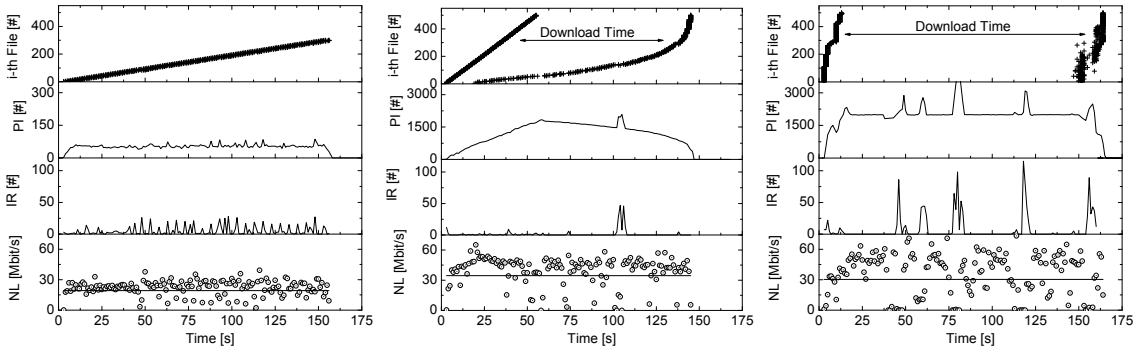


Figure 6: Load at the designated router of the receiver while requesting non-existing content [5]

ing in service degradation for all dependent nodes. This also underlines the findings in [14] that testifies the Interest state handling high resource requirements.

5.1.2 Chunk-based state multiplication

To analyse the performance of an actual content dissemination, we utilized CCNx1 to request a fixed amount of 10 Mbit files with varying frequency from CCNx5. Depicted in Figure 7 are the resulting graphs consisting of the Download Time, Pending Interests, Interest Retransmissions and the Network Load of the first hop node.
The three subfigures illustrate the measurement values for scenarios where 2 files (Figure 7(a)), 10 files (Figure 7(b)) and 100 files (Figure 7(c)) are requested per second. The topmost graph shows that at a rate of 2 files per second the time to complete the download of each file is fixed. There is no notable variation in time. Moving on to the 10 files per second case the Download Times diverges non-linear while at the same time the amount of Pending Interests increases in comparison to the 2 files per second case. This effect is even more concise when compared with Figure 7(c) where the Pending Interest rate ramps up to $\approx$ 2000 PI's and remains at that level until the end of the content dissemination. At the same time the Network Load remains at a level of roughly 30% in both cases (10 and 100 files per second). This effect arises due to the increased number of PIT entries that are generated due to the increased file count. A second factor arises in this case, the Interest Retransmission mechanism that is responsible for the other share of the increased Interest amount. Corresponding to this, the states that the CCNx router has to maintain cause an increase in CPU and Memory resource consumption (not depicted). For the reason of this state management overhead the router is not able to utilize the full link capacity resulting in the increase in Download Time. The difference in Download Time between Figure 7(b)



(a) 2 files per second (b) 10 files per second (c) 100 files per second

Figure 7: Parallel download of 10 Mbit files: Start and stop time of the download per file at the receiver & resource consumption at its designated router [Pending Interests (PI), Interest retransmits (IR), and Network Load (NL) including the mean goodput] [5]

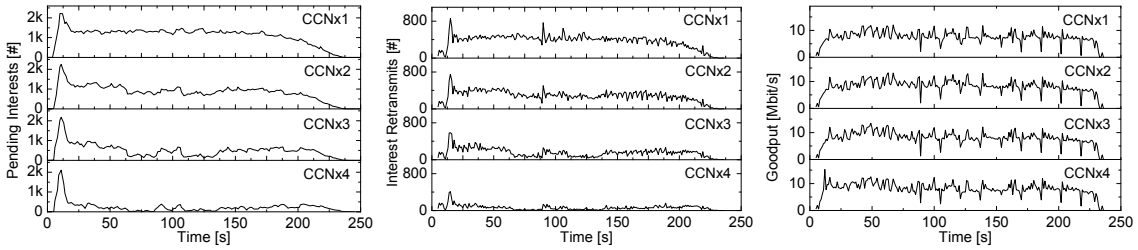
and 7(c) are the direct outcome of this overload situation. Further due to the combination of an increased rate of Interest receipts and the overload, the router randomly misses Interest packets that it is thus not able to process. This effect results in an excessively increased, nearly uniform distribution of Download Times for all files.

5.2 Extended experiments: State propagation & correlation

For our Extended Experiments, we utilize the longer chain of CCNx router nodes (ccnxlong) depicted in Figure 3. Further we changed the testbed to utilize UDP for the underlying communication protocol to connect the CCNx nodes. All forwarding nodes continue running within the virtualized environment and they are equipped with 3 GB of RAM, two cores à 2.4GHz.
The focus in this subsection resides on the long chain of CCNx nodes in our testbed. First we start again by taking a look at a homogeneous network before moving on to different heterogeneous constellations.

5.2.1 Homogeneous network

Plotted in Figure 8 are the results of a file transfer run where 500 files with a size of 10 Mbit each are requested at a rate of 100 files per second. The graphs show the Pending Interests, Interest Retransmits as well as the Network Goodput. The Pending Interest graph shows that the closer a node is located towards the source of the content the lower the amount of Pending Interests. In direct correlation to this also the Interest Retransmission-Rate declines from subscriber towards the source. This effect is caused by the fact that arriving data chunks



(a) Pending Interests (b) Interest Retransmits (c) Network Utilization

Figure 8: Routing and forwarding performance in a homogeneous five-hop network [5]

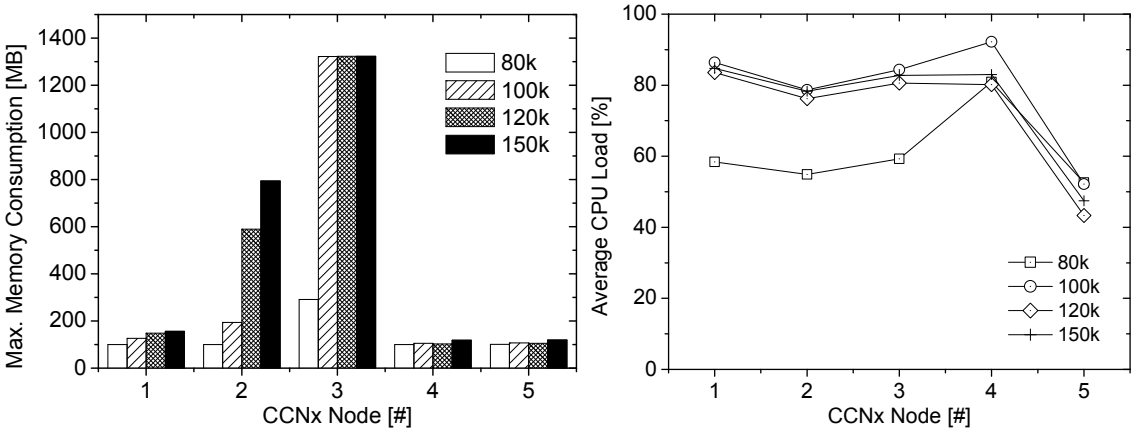
clear the PIT entries and the interval between arrival of the Interest and the arrival of the data chunk diminishes towards the content source. This conclusion is supported by the Network Utilization graph that contrary to the declining Pending Interests and Interest Retransmission graph shows an equal Network Utilization for all involved nodes.

5.2.2 Heterogeneous network

After exploring the behaviour of a homogeneous router infrastructure we now take a look at a heterogeneous infrastructure setup. Hence node CCNx4 is reconfigured to have just 600 MHz of CPU capacity (25% of the other nodes CPU capacity) available.
For the measurement we choose the same parameters that we already used in Section 5.1.1 from 80.000 up to 150.000 Interest for non-existing content. Figure 9 shows the CPU and Memory consumption. Effected by the CPU capacity reduction a peek in CPU Load accrues at CCNx4. The last nodes CPU Load drops distinctly caused by the bottleneck, CCNx4, that is just able to process and pass on a fraction of the incoming content Interests.
Also the Memory graph (Figure 9(a)) indicates that CCNx4 is lacking resources to process the stream of Interests. Hence on CCNx3 the memory consumption is magnitudes higher than on CCNx4. This effect is also reflected back towards the Interest issuer. The CPU Load off all preceding nodes is also on a very high level, instead of adjusting to the weakest nodes performance. Thus a bottleneck node within the network is negatively influencing the resource consumption and tuhus the performance of all its preceding nodes.

5.2.3 Heterogeneity magnitudes

To further identify the influence of network node heterogeneity a measurement run, including a bottleneck node, where actual files are downloaded is also performed. The CPU capacity of CCNx4 in different runs is configured to 600, 1200 and 2400 MHz. The latter case of 2400 MHz just represents the homogeneous case and is taken into account for the purpose of comparison.



(a) Memory Consumption (b) Average CPU Load

Figure 9: Load per hop for a chain of 5 routers while initiating a 80k, 100k, 120k, and 150k different Interests for non-existing content [5]

As long as the measurement environment consists of homogeneous nodes the retransmits decay the closer the content router is located towards the publisher node (Figure 10). This follows the findings we already made in Section 5.2.1.
But, as soon as the path contains a bottleneck node the whole picture shifts, the retransmission values of all preceding nodes ramp up to the highest level experienced in the homogeneous case. This effect indicates how sensitive the system behaves in case of bottlenecks. Preceding nodes continue to retransmit their Interests. Instead of getting cleared through arriving content while traversing the chain of content routers, they are propagated without any throttling mechanism up to the bottleneck, where they are finally dropped because of the CPU resource issues.

5.2.4 Increased inhomogeneities

Since ICN aims to become a leading technology in the distribution of content throughout the Internet it is even more likely that more than one bottleneck exists within the network path from subscriber towards publisher. Thus some kind of resource fluctuation over different nodes where systems are utilized and influenced in various intensities is very likely to occur. Since we identified the CPU as the core perpetrator for bottlenecks we reflect this fluctuating load behaviour through the introduction of competitive CPU consuming processes on our content routers. The competitive processes alternate in status. For a period of 10s they consumes 90% of the CPU resources before switching into a sleep state for another 20s. The interleaving of these processes we choose for the nodes of the ccnxlong route is depicted in Figure 11.

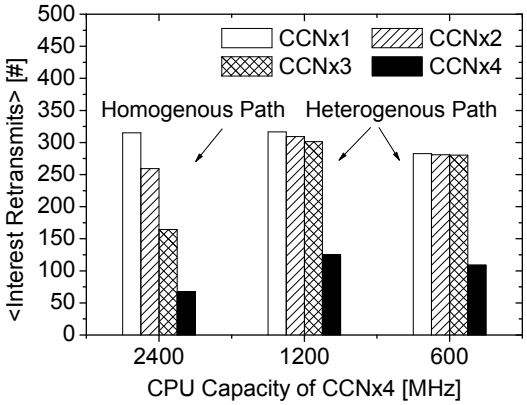


Figure 10: The effect of router strengths on Interest trading [5]

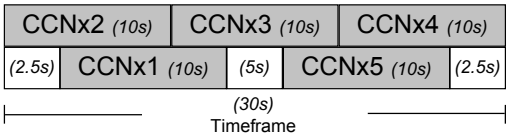
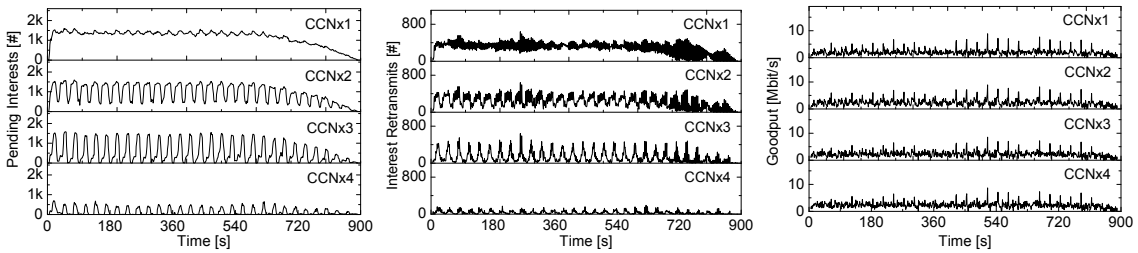


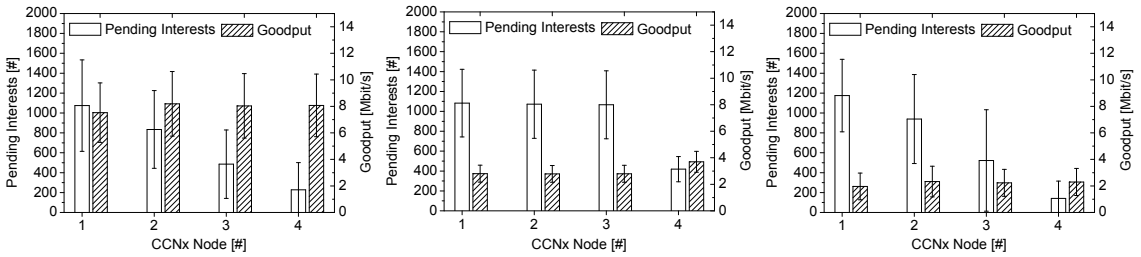
Figure 11: Interleaving of the competitive processes

Figure 12 illustrates the results of this measurement scenario. When taking a look at the Network Utilization graph it is obvious that the overall Network Performance is degraded by magnitudes. The Pending Interests and Interest Retransmits are influenced by the competitive process in a way that the spikes get thinner the closer a node is located towards the publisher. This effect is an outcome of the behaviour that the subscription node issues Interests and Interest Retransmissions, meaning that these Pending Interests can just decline from subscriber towards publisher. These findings illustrate once again that the performance is highly dependent on cross traffic traversing a content router and the basic resource configuration of the content router.



(a) Pending Interests (b) Interest Retransmits (c) Network Utilization

Figure 12: Routing and forwarding performance in a five-hop network with alternating CPU reductions [5]



(a) Homogeneous Network (b) Single Point of Weakness (c) Alternating Resources

Figure 13: Comparison of state management and forwarding performance in different network scenarios (mean and standard variation) [5]

5.2.5 Summarising retrospection

The results of our experiments show consistently that the performance of CCNx is strongly dependent on the performance of the weakest node that is traversed on the path between publisher and subscriber. Figure 13 illustrates this finding once again. It directly compares the results of our Extended Experiment results. One can see that as long as an equal amount of resources is available on the nodes (Figure 13(a)), the Pending Interests which can be seen as the indicator to load are steadily declining from subscriber to publisher while the throughput is on a high level in relation to (Figure 13(b) or 13(c)). Compared to this the Single Point of Weakness case demonstrates that all preceding nodes of a bottleneck suffer high load in terms of Pending Interests and thus CPU resources. This may lead to bottlenecks for cross traffic paths in larger deployments.
The identified performance flaw is getting even worse when multiple bottlenecks appear on the path. Figure 13(a) points out that the Network Goodput is further reduced. The Pending Interest match the shape of the Homogeneous Scenario which might be misleading to the conclusion that its caused by the same mechanisms. However the distribution of Pending Interests in Figure 13(a) is caused by the accumulation of the Pending Interest drop effect of the preceding bottlenecks of each node whereas the Pending Interest decay in Figure 13 is caused by the data that is received more recent to the receipt of the Interest.

6 Conclusion

In this paper, we started by examining issues we see in actual ICN concepts and their implementations. We focused on the two threads of Resource Exhaustion and State Decorrelation. To verify their existence and to analyse the behaviour in case of exploitation of this threads, we implemented a testbed consisting of NDN / CCNx content routers.

Our measurements illustrated that the actual design is vulnerable to those threads. Through a bulky generation of Interests, an end-node is able to overload the routing infrastructure resulting in an decreased service availability. The overload of the network infrastructure is shown to be even easier if bottlenecks (with respect to CPU processing capacity) exist within the network.
All this is caused by the data-driven state that the content routers have to maintain. One approach to mitigate the issues would be an Interest transmission rate limitation. As far as the amount of Pending Interests is limited no further data can be requested in that very moment. This might lead to situations where due to the amount of allowed Pending Interests, it is not possible to request an appropriate amount of data to fully utilize the available bandwidth of a link. Further the time that a Pending Interest, that requests available content, resides on a router varies depending on the RTT between content source and subscriber. If the RTT is increased the amount of Pending Interests also needs to increase to keep the link utilization at the same level. Hence an Interest transmission rate limitation would result in some sort of traffic shaping and rate-limitation and is thus not considered as an appropriate solution to the problem of router resource exhaustion.
Hence future work needs to be dedicated to the research of state management in case of ICN to resolve the issues discussed in this work.

References

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman, "A Survey of Information-Centric Networking," *IEEE Communications Magazine*, vol. 50, no. 7, pp. 26–36, July 2012.

[2] PARC, "The CCNx Homepage," http://www.ccnx.org/, 2012.

[3] L. Zhang, D. Estrin, J. Burke, V. Jacobson, and J. D. Thornton, "Named Data Networking (NDN) Project," PARC, Tech.report ndn-0001, 2010.

[4] M. Wählisch, T. C. Schmidt, and M. Vahlenkamp, "Bulk of Interest: Performance Measurement of Content-Centric Routing," in *Proc. of ACM SIGCOMM, Poster Session (SIGCOMM'12)*. New York: ACM, August 2012, pp. 99–100. [Online]. Available: http://conferences.sigcomm.org/sigcomm/2012/paper/sigcomm/p99.pdf

[5] ——, "Backscatter from the Data Plane — Threats to Stability and Security in Information-Centric Networking," Open Archive: arXiv.org, Technical Report arXivarXiv:1205.4778, 2012. [Online]. Available: http://arxiv.org/abs/1205.4778

[6] M. Gritter and D. R. Cheriton, "An Architecture for Content Routing Support in the Internet," in *Proc. USITS'01*. Berkeley, CA, USA: USENIX Association, 2001, pp. 4–4.

[7] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica, "A Data-Oriented (and beyond) Network Architecture," *SIGCOMM Computer Communications Review*, vol. 37, no. 4, pp. 181–192, 2007.

[8] V. Jacobson, D. K. Smetters, J. D. Thornton, and M. F. Plass, "Networking Named Content," in *Proc. of the 5th Int. Conf. on emerging Networking EXperiments and Technologies (ACM CoNEXT'09)*. New York, NY, USA: ACM, Dec. 2009, pp. 1–12.

[9] "The PSIRP Homepage," http://www.psirp.org, 2012.

[10] P. Jokela, A. Zahemszky, C. E. Rothenberg, S. Arianfar, and P. Nikander, "LIPSIN: Line Speed Publish/Subscribe Inter-networking," in *Proc. of the ACM SIGCOMM 2009*. New York, NY, USA: ACM, 2009, pp. 195–206.

[11] B. Ahlgren *et al.*, "Second NetInf Architecture Description," 4Ward EU FP7 Project, Tech.report D-6.2 v2.0, 2010.

[12] M. Vahlenkamp, "Information-centric networking - a related work survey," HAW Hamburg, Tech. Rep., 2012. [Online]. Available: http://inet.cpt.haw-hamburg.de/teaching/ss-2012/master-projects/markus_vahlenkamp_aw2.pdf

[13] A. Ghodsi, S. Shenker, T. Koponen, A. Singla, B. Raghavan, and J. Wilcox, "Information-Centric networking: Seeing the Forest for the Trees," in *Proc. of the 10th ACM HotNets Workshop*, ser. HotNets-X. New York, NY, USA: ACM, 2011.

[14] T. Lauinger, "Security & Scalability of Content-Centric Networking," Master's thesis, TU Darmstadt, Darmstadt, Germany, 2010.

[15] P. Gasti, G. Tsudik, E. Uzun, and L. Zhang, "DoS and DDoS in Named-Data Networking," ArXiv e-prints, Tech. Rep. 1208.0952, August 2012.

[16] L. Wang, A. K. M. M. Hoque, C. Yi, A. Alyyan, and B. Zhang, "Ospfn: An ospf based routing protocol for named data networking," Tech. Rep., July 2012. [Online]. Available: http://www.named-data.net/techreport/TR003-OSPFN.pdf

[17] T. Ylonen and C. Lonvick, "The Secure Shell (SSH) Protocol Architecture," IETF, RFC 4251, January 2006.

A Appendix

A.1 Measurement files

```
#Time, INames, IPend, IProp, INoted, CPU, Mem
1335953368,9,0,0,0,0.0,1868
1335953368,13,0,0,4,0.0,1868
1335953369,152,68,68,43,33.2,2820
1335953370,596,70,70,476,53.0,5372
1335953370,961,60,60,846,19.9,6540
1335953371,1422,139,139,1220,72.8,9440
1335953371,2001,111,111,1814,66.3,13136
1335953372,2619,231,231,2303,53.0,16768
1335953372,3164,185,185,2882,59.6,18352
```

Listing 1: Excerpt of the routing process measure file

```
#timestamp, data direction goodput(byte/s), interest direction goodput(byte/s), data direction throughput(byte/s), interest direction throughput(byte/s), duplicates
5,123609,8724,131397,12948,4
6,2993117,73151,3147953,129911,3
7,3364644,84604,3533670,139846,2
8,2375390,59386,2497820,111658,1
9,3536326,84621,3711424,139995,4
10,4267783,105339,4472977,171603,3
11,4306645,106587,4510189,170805,4
12,4604574,110608,4823100,179644,5
13,3912582,73946,4103190,144104,1
14,3291866,99610,3472376,194518,0
15,4089995,109502,4300205,189164,3
```

Listing 2: Excerpt of the finally deduced capture information

```
#no,ccnxName,transferStart,tansferStop,readTotal
1,ccnx:/ccnxlong/10Mbit1.txt,5,81,1281250
2,ccnx:/ccnxlong/10Mbit8.txt,5,81,1281250
3,ccnx:/ccnxlong/10Mbit4.txt,5,85,1281250
4,ccnx:/ccnxlong/10Mbit2.txt,5,88,1281250
5,ccnx:/ccnxlong/10Mbit7.txt,5,91,1281250
6,ccnx:/ccnxlong/10Mbit6.txt,5,92,1281250
7,ccnx:/ccnxlong/10Mbit0.txt,5,94,1281250
8,ccnx:/ccnxlong/10Mbit9.txt,5,96,1281250
```

Listing 3: Excerpt of the file transfer statistics

```
File: /home/ccnx/tmpfs/capture
Packets captured: 410916
Packets received/dropped on interface eth0: 410917/0
```

Listing 4: Sample content of the capture summary file

```
<?xml version="1.0"?>
<psml version="0" creator="wireshark/1.6.6">
<structure>
  <section>No.</section>
  <section>Time</section>
  <section>Source</section>
  <section>Destination</section>
  <section>Protocol</section>
  <section>Length</section>
  <section>Info</section>
</structure>

<packet>
  <section>1</section>
  <section>1337867906.522304</section>
  <section>141.22.28.230</section>
  <section>141.22.28.231</section>
  <section>CCN</section>
  <section>154</section>
  <section>Interest, ccnx:/ccnxlong/10Mbit4.txt</section>
</packet>

<packet>
  <section>2</section>
  <section>1337867906.522701</section>
  <section>141.22.28.231</section>
  <section>141.22.28.230</section>
  <section>TCP</section>
  <section>66</section>
  <section>ccnx > 51074 [ACK] Seq=1 Ack=89 Win=227 Len=0 ↘
    →TSval=36547264 TSecr=3
6544396</section>
</packet>
```

Listing 5: Excerpt of the capture extracted psml file

A.2 Measurement run graphs

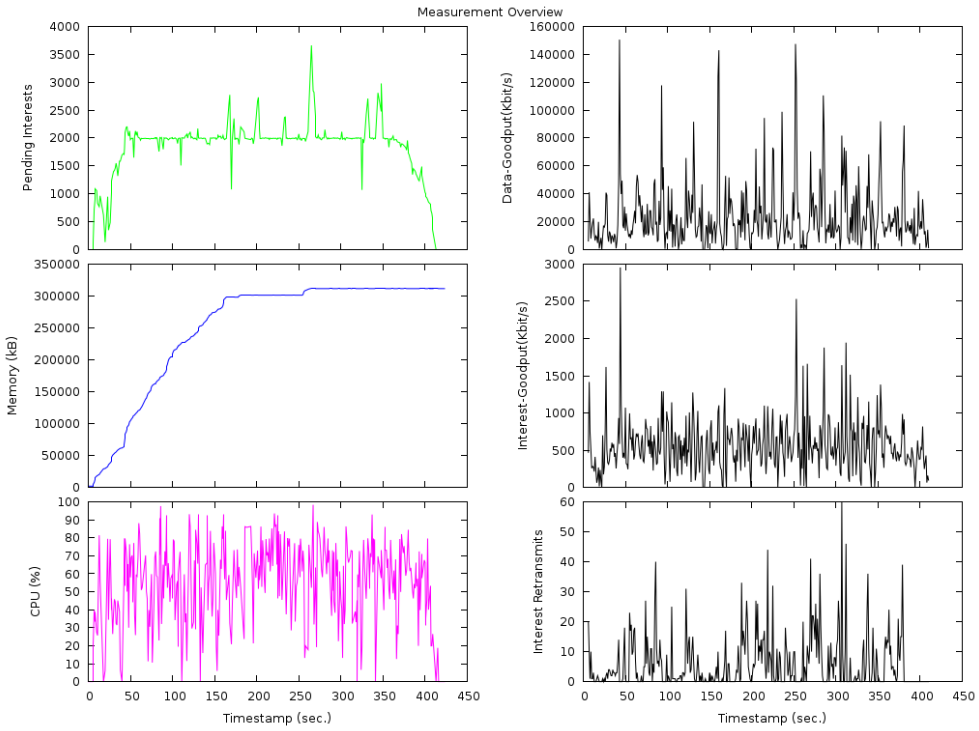


Figure 14: Per node statistics gathered throughout measurement

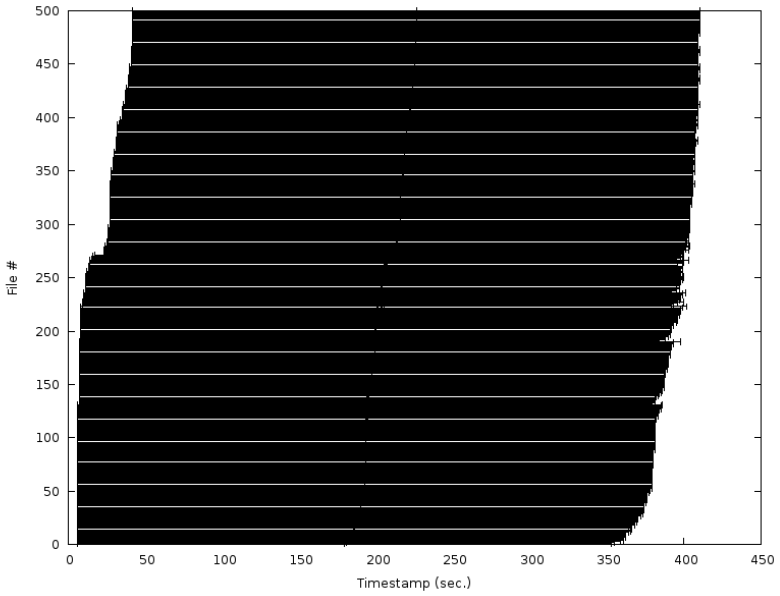
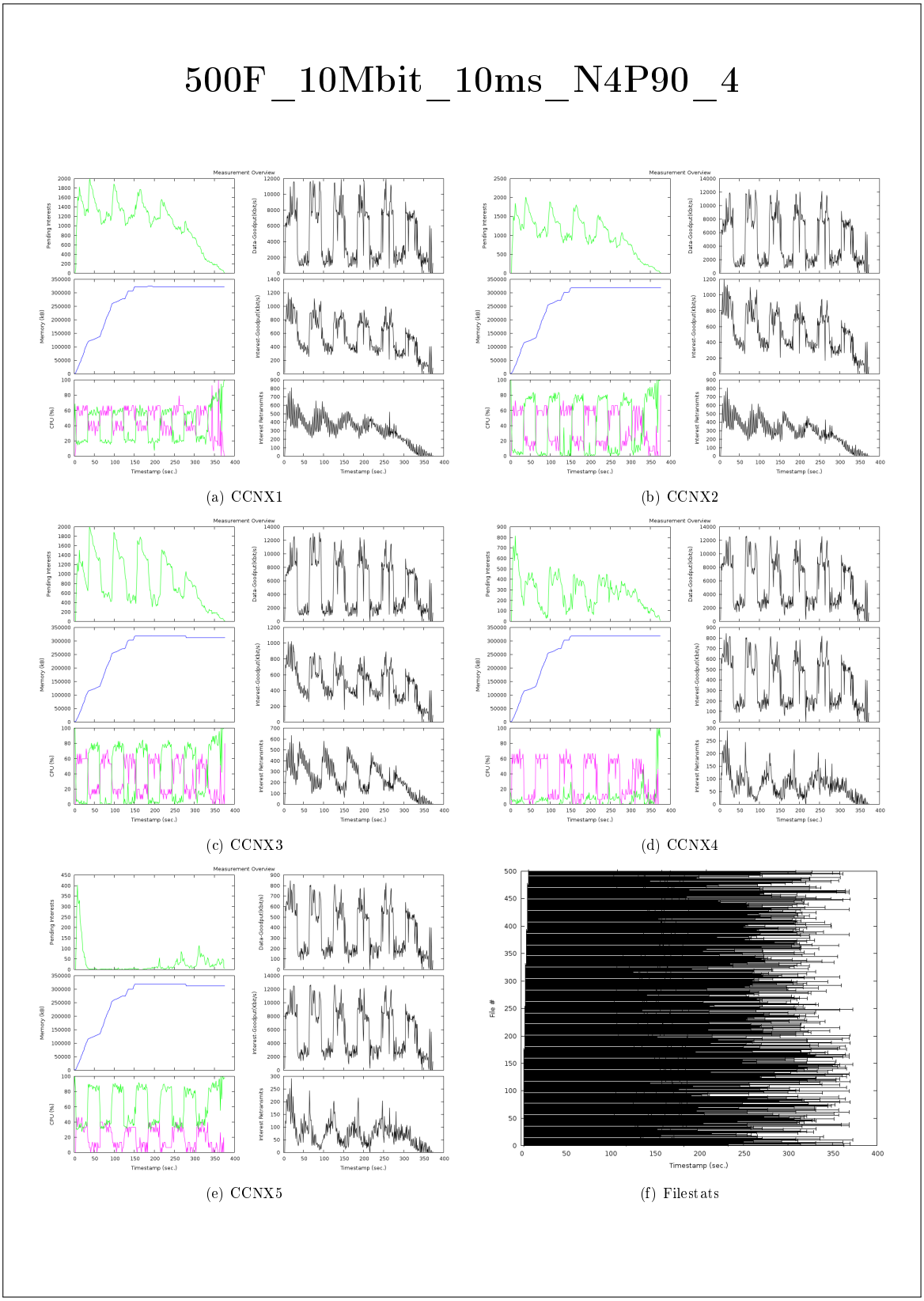


Figure 15: Filetransfer statistics



(a) CCNX1 (b) CCNX2

(c) CCNX3 (d) CCNX4

(e) CCNX5 (f) Filestats

Figure 16: Measurement run overview page