Mini-CCNx: Fast Prototyping for Named Data Networking

Carlos M. S. Cabral , Christian E. Rothenberg (Co-advisor) , Maurício F. Magalhães (Advisor)

Department of Computer Engineering and Industrial Automation (DCA)
Faculty of Electrical and Computer Engineering (FEEC)
University of Campinas (UNICAMP), São Paulo, Brazil

{cabral,chesteve,mauricio}@dca.fee.unicamp.br

Abstract – Experimental research in Information-Centric Networking (ICN) is crucial to the evaluation of new architectural proposals that bring named pieces of content as the main element of networks. This paper presents a new fast prototyping tool for the NDN (Named Data Networking) model, Mini-CCNx, that aims at filling an existing gap in generally available experimental platforms. Using container-based emulation and resource isolation techniques, Mini-CCNx appears as a flexible, scalable, high-fidelity, and low-cost tool that enables experiments on emulated networks with hundreds of NDN nodes in a commodity laptop. All those characteristics are highly desirable for the evaluation of open research issues in ICN, such as routing protocols, forwarding strategies, caching techniques, content-oriented application development, and so on.

Keywords – CCN, NDN, emulation, prototyping, Internet, ICN

1. Introduction

With the motivation of shifting the networking paradigm to a model that fits better the current usage of the Internet, a number of Information-Centric Networking (ICN) proposals have appeared with the goal of putting *named content* as the main element of networks. Proposals like DONA[7], NetInf [4] and CCN [6], among others, introduce several new concepts and strategies towards an ICN architecture. Like in every new design proposal, experimentally-driven research is crucial to the evaluation of new ideas. Especially in the computer networking area, requirements such as scalability and experimental fidelity are highly desirable and a matter of concern when attempting to move ideas to real field trials.

During our research journey on new forwarding strategies and probabilistic state reduction techniques for CCN core nodes, we faced an existing gap in feature-rich, generally available experimental tools. We could not find a low-cost, scalable, high-fidelity, and sufficiently flexible experimental platform to carry in-depth evaluation of diverse and customizable CCN scenarios.

This experimental issue motivated the development of Mini-CCNx presented on this paper. Inspired by well-succeeded practices in fast prototyping for Software-Defined Networks (SDN) [8], Mini-CCNx is able to specify and run a complete content-centric network of hundreds of nodes in a simple laptop, with high flexibility, agility, and configurability. Furthermore, Mini-CCNx provides a high-fidelity and experimenter-friendly prototyping environment, enabling applications and network configuration being tested before moving to real testbed environments.

Mini-CCNx is based on the Named Data Networking (NDN) [10] model and uses the official implementation, CCNx [2], what brings greater relevance for the user and developer communities. In this paper, we describe the platform from an architectural and experimenter point of view. We validate the capabilities of Mini-CCNx through extensive experimentation.

The remainder of the paper is organized as follows. Section 2 analyses existent tools for NDN experimentation and outlines the goals of our platform. Section 3 describes the architectural details of Mini-CCNx. Section 4 presents an evaluation of Mini-CCNx in terms of scalability, performance, and fidelity. Finally, Section 5 concludes the paper final remarks.

2. Motivation and Goals

Any networking experimentation environment would ideally combine (at least) the following characteristics:

Flexibility. It should be possible to rapidly create several scenarios using different configuration parameters.

Scalability. It should be possible to create topologies with a sufficiently high number of nodes.

Low-Cost. The tool can be run in a commodity laptop/desktop or in a single Amazon EC2 instance.

Table 1. Summary of tools and its characteristics

	Simulators	**Testbeds**	**Emulators**
Ex:	*ndnSIM/ccnSim*	*Testbed NDN*	*Mini-CCNx*
Flexibility	High	Low	High
Scalability	High	Low	High
Cost	Low	High	Low
Realism	No	Yes	Yes
Ease of Config.	Medium	Low	High
Link config.	Yes	With restrictions	Yes

Realism. As described in [5], three realism dimensions can be defined for an experimentation tool: (*i*) *functional* realism (the system behavior must be the same as the one of a real deployment and the code executed within the tool must be the same code run on the real hardware), (*ii*) *timing* realism (the performance must be close or indistinguishable from the real behavior), and (*iii*) *traffic* realism (the system must be able to generate and receive real traffic, coming either from the Internet or from the local network).

2.1. Existing ICN experimentation tools

We can group existing ICN research platforms at least in three main categories: simulators, testbeds, and emulators, each with its pros and cons.

Simulators. They are flexible, scalable, and, in general, have a low cost. ndnSIM [11] is a ns-3 module that implements the NDN model. ccnSim [1] extends OMNeT++ with the required structures and protocols to implement the CCN model. Both tools have the disadvantages of any simulator: (*i*) they are not realistic (the code used in the simulator is different from the one that will be executed in a real deployment), and (*ii*) the hardware, protocol stack, and traffic models used by simulators may not yield the best fidelity.

Testbeds. They are experimental infrastructures that provide real resources for realistic test scenarios. They can be shared among researchers (e.g., GENI, Planetlab)or specifically created for a local deployment. Testbeds are realistic by nature but, in general, they have (*i*) high creation and maintenance costs, and (*ii*) reduced flexibility and scalability when it comes to custom topologies creation. Furthermore, the largest international testbeds are beyond the reach of most researchers.

Emulators. In general, they may be not as scalable as simulators, but arguably they are equally flexible and have low hardware costs. Like testbeds, they are realistic because they run real code (applications, OS kernel, etc). The work on Mini-CCNx presented in this paper aims at filling the gap of being an emulator specifically tailored for NDN experimentation.

Table 1 summarizes the analysed characteristics and adds two more factors: ease of configuration, and the possibility of configuring link parameters, such as packet delay and loss. The lack of a tool combining all desired factors and the lessons learned with SDN prototyping motivated our investment on developing Mini-CCNx.

3. Mini-CCNx Architecture

Mini-CCNx is a Mininet-HiFi fork [5] (originally proposed for OpenFlow networks) augmented with several classes and mechanisms to build NDN environments based on the project's official code base [9].

3.1. Overview

Mini-CCNx uses container-based emulation (CBE) [1], a lightweight OS-level virtualization technique. Each container allows groups of processes to have independent views of system resources, such as process IDs, file systems and network interfaces while still using the same kernel. Each container is a NDN node, with its own network namespace, virtual network interface(s), NDN-specific data structures implemented by the *ccnd* daemon (PIT, FIB and CS) and repositories, as implemented by he *ccnr* daemon. These nodes are connected to each other using virtual Ethernet links in the kernel space.

CBE yields higher scalability when compared to full-virtualization by trading the ability to run multiple OS kernels for lower overall overhead using a single kernel [12]. The higher scalability comes however at a price: isolation may be compromised. The kernel resources are shared by all the NDN nodes which may interfere with each other. For example, if a misbehaving content-oriented application in a node begins to indefinitely allocate memory, the overall experiment results will be compromised due to lack of memory available for other

[1] http://lxc.sourceforge.net/

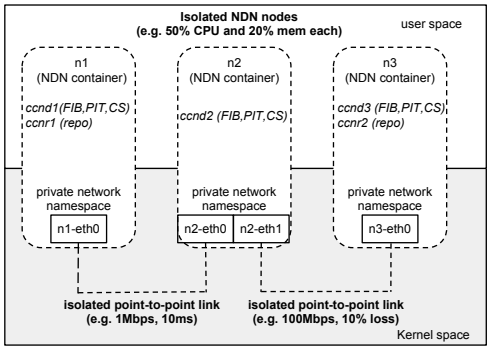


Figure 1. Three NDN nodes connected linearly using Mini-CCNx containers

nodes. Therefore, Mini-CCNx uses isolation techniques in order to limit the resources available for each node and link.

Like Mininet-HiFi, Mini-CCNx uses Linux *cgroups* to limit CPU bandwidth for each node. Mini-CCNx also extends this concept adding limits to memory utilization, an important subject for NDN when it comes to caching and content storage. Finally, using Linux traffic control (`tc`), it is possible to configure several link properties such as bandwidth, delay, and packet loss. Figure 1 illustrates the CBE and isolation features used within Mini-CCNx.

4. Performance and Fidelity

We now focus on analysing Mini-CCNx's fidelity and performance using a low-cost hardware, largely available to most researchers and students.[2] The `ccnx` version used in all cases is 0.7.0. We analyse the following performance dimensions: (*i*) scalability, (*ii*) coherence and (*iii*) fidelity

4.1. Scalability

The scalability is analysed with regards to the number of nodes and links Mini-CCNx is able to instantiate and run. Thus, two representative topologies were chosen, (1) full mesh, and (2) linear (nodes connected in a simple linear fashion). These topologies were chosen because they represent two connectivity limits –any other topology will have a connectivity level within this range.

Tables 2 and 3 show the number of instantiated NDN nodes, the memory used by Mini-CCNx only, the total memory used by the `ccnd` daemon instances running in each node, the total overall memory (Mini-CCNx + `ccnd`), the number of links, and the set up time of each scenario. Note that the largest part of memory usages comes from the `ccnd` daemons and this memory usage grows linearly with the number of nodes. We note that the set up time is strongly related to the number of instantiated links, becoming the limiting factor ($>$ 10min) for a full mesh topology of 128 nodes. When looking at the linear topology, memory establishes the bottleneck after $\sim$1,500 nodes.

[2] All the tests were done on a laptop with a typical configuration (Intel Core I5 2410M processor with 4GB RAM).

Table 2. Scalability. *Linear* topology

# of nodes	Mini-CCNx Mem.(MB)	`ccnd` Mem.(MB)	Total Mem.(MB)	# of links	Set up time(s)
4	15.0	7.2	22.2	3	$<$1
64	15.7	113.8	129.5	63	6
512	21.6	918.5	940.1	511	95
1024	27.8	1835.4	1863.2	1023	228
1536	35.3	2754.2	2789.6	1535	320

Table 3. Scalability. *Full mesh* topology

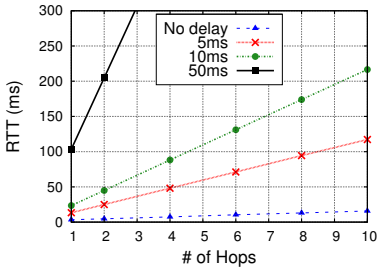
# of nodes	Mini-CCNx Mem.(MB)	`ccnd` Mem.(MB)	Total Mem.(MB)	# of links	Set up time(s)
4	15.1	7.2	22.3	6	$<$1
16	15.7	28.9	44.6	120	11
64	26.0	114.5	140.6	2016	118
128	62.0	229.8	291.8	8128	753

4.2. Coherence

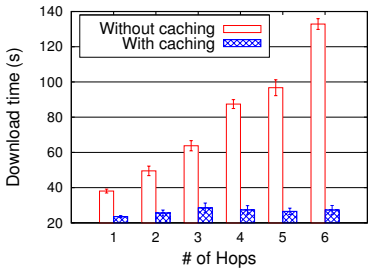
We focus now on how coherent are the results with regards to experiment parameters such as link delay, bandwidth, and number of hops. For this analysis, we investigate two simple NDN scenarios. In the first one, we measure the RTT using `ccnping` [9] for varying hop numbers and link delay values (Fig. 2(a)). As expected, the RTT increases linearly with the number of hops. For different link delay values, the RTT behavior is consistent. For example, under 10 ms link delay and for 2 hops, the *ping* takes 20 ms upstream, plus 20 ms downstream, plus some processing time per node.

In the second scenario, the average download time of a 100MB file was measured for different hop distances. Two sub-cases were analysed: (1) *no-caching*, where Interest messages go all the path until the producer, and (2) *caching*, where a first download of the file populates node caches along the path, and then a second request may retrieve pieces of content from the caching nodes (de-

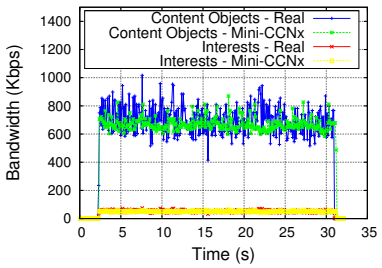
fault caching configuration of CCNx was used). As shown in Fig. 2(b) (with 95% CI), download times increase for larger distances between client and producer. We can also observe the expected effects of caching, where the download time benefits from the amount of cached content in the nodes closer to the client.



(a) RTT vs # hops for varying link delays



(b) Average download time vs # hops



(c) BW comparison with real testbed

Figure 2. Coherence and fidelity analysis

4.3. Fidelity

As a first assessment on how Mini-CCNx is able to reproduce real experiments with high-fidelity, we use a simple topology with two real desktops and native `ccnx` installations directly connected via FastEthernet interfaces. Using the `ccntraffic` [3] generator, the first desktop constantly sends Interest messages asking for different contents while the second desktop answers with 1024-bytes Content Objects. The same scenario was reproduced using Mini-CCNx, with 100Mbps and a delay of 200 μs as link parameters. Figure 2(c) shows the Interest and Content Objects (Data) traffic for both cases. Note that the bandwidth consumption in Mini-CCNx follows very closely the real deployment behavior.

5. Conclusion

Inspired by the well-succeeded experience in fast prototyping for SDN, Mini-CCNx appears as a innovative tool that aims at filling an existing gap in the currently available NDN experimental platforms. Mini-CCNx is realistic, low-cost and scalable: a whole content-centric network, with hundreds of nodes, can be run in a simple laptop, with easy configuration and high-fidelity results.

References

[1] ccnSim. Scalable chunk-level CCN simulator. `http://perso.telecom-paristech.fr/~drossi/index.php?n=Software.ccnSim`.

[2] CCNx. Official implementation of the CCN model. `https://www.ccnx.org/`.

[3] CCNx Traffic. CCNx: traffic generation - ARL ONL Wiki. `http://wiki.arl.wustl.edu/onl/index.php/CCNx:\_traffic\_generation`.

[4] Christian Dannewitz, Dirk Kutscher, Börje Ohlman, Stephen Farrell, Bengt Ahlgren, and Holger Karl. Network of Information (NetInf) - An Information-Centric Networking Architecture. *Computer Communications*, January 2013.

[5] Nikhil Handigol, Brandon Heller, Vimalkumar Jeyakumar, Bob Lantz, and Nick McKeown. Reproducible network experiments using container-based emulation. *CoNEXT '12*, page 253, 2012.

[6] Van Jacobson, Diana K. Smetters, James D. Thornton, Michael F. Plass, Nicholas H. Briggs, and Rebecca L. Braynard. Networking named content. In *CoNEXT '09*, page 1, New York, New York, USA, December 2009. ACM Press.

[7] Teemu Koponen, Mohit Chawla, Byung-Gon Chun, Andrey Ermolinskiy, Kye Hyun Kim, Scott Shenker, and Ion Stoica. A data-oriented (and beyond) network architecture. *ACM SIGCOMM CCR*, 37(4):181, October 2007.

[8] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop. In *ACM SIGCOMM Workshop - Hotnets '10*, pages 1–6, New York, New York, USA, October 2010. ACM Press.

[9] Named Data Networking. NDN Github Repository. https://github.com/named-data, April 2013.

[10] NDN Project. Named Data Networking. `http://www.named-data.net/`.

[11] ndnSIM. NS-3 based NDN simulator. `http://ndnsim.net/`.

[12] Stephen Soltesz, Herbert Pötzl, Marc E. Fiuczynski, Andy Bavier, and Larry Peterson. Container-based operating system virtualization. *ACM SIGOPS Operating Systems Review*, 41(3):275, June 2007.