High Fidelity Content-Centric Experiments with Mini-CCNx

Carlos Cabral, Christian Esteve Rothenberg, Mauricio Ferreira Magalhães
School of Electrical and Computer Engineering
University of Campinas - UNICAMP
Campinas, Sao Paulo, Brazil
{cabral,chesteve}@dca.fee.unicamp.br

***Abstract*—Experimentally-driven research is crucial to the evaluation of any networking technology. In this paper, we focus on research experiments with Mini-CCNx, a convenient yet high-fidelity prototyping tool to evaluate Information-Centric Networking (ICN) proposals that bring named pieces of content as the main element of networks. More specifically, Mini-CCNx is tailored for the Named Data Networking (NDN) model and fills 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 in support of NDN research. To evaluate Mini-CCNx as a valid research platform, we first reproduce a series of experiments from the NDN literature. We then run routing experiments on an emulated version of the real NDN testbed and provide insights into open research topics such as behavior of the OSPFN routing protocol, forwarding strategies, and network resiliency.**

I. Introduction

With the motivation of shifting the networking paradigm to a model that fits better the current content-oriented usage of the Internet, a number of Information-Centric Networking (ICN) proposals have appeared characterized by putting *named content* as the main element of networks [1]. ICN introduces several new concepts and approaches towards realizing the paradigm shift from a host-centric communication model to a design centered around information access, irrespective of its physical location. Like in every new networking proposal, experimentally-driven research is crucial to evaluation of new ideas. Especially in the field of networking, requirements such as scalability and experimental fidelity are highly desirable when attempting to move ideas to real field trials.

Despite their similarities and differences, one common issue of ICN proposals is the amount of unresolved research challenges in this area of networking research [15]. During our own research journey on the applicability of new forwarding strategies and probabilistic state reduction techniques for scalable multicast [13] to the Content-Centric Networking (CCN) [6] model, we faced a gap in feature-rich, generally available experimental tools. The lack of a low-cost, scalable, high-fidelity, and sufficiently flexible experimental platform to evaluate diverse and customizable CCN scenarios motivated the development of Mini-CCNx [2].

Inspired by well-succeeded practices in fast prototyping for Software-Defined Networks (SDN) with Mininet [8], Mini-CCNx basically introduces a number of extensions to the Mininet network emulator to seamlessly support CCNx, the official code base of the Named Data Networking (NDN) project [10] –an outgrowth from the CCN proposal. The extensions are transparent to both CCNx and Mininet, and include software additions to allow and ease NDN experiments while achieving scalability, coherence, fidelity, and isolation [3]. With the same goals of Mininet but applied to NDN (instead of OpenFlow networks), Mini-CCNx aims at supporting research, development, learning, prototyping, testing, debugging, and any other tasks that could benefit from having a complete NDN experimental network on a commodity PC.

In this paper, we describe Mini-CCNx from an architectural and experimenter's point of view and then carry a series of experiments, firstly to validate the Mini-CCNx by reproducing results from the NDN literature [6], [11], and secondly to provide insights into content-centric routing protocols [7]. We observe that the obtained results match those from previously published work on content distribution efficiency, strategy/forwarding layer, and performance of the NDNVideo streaming application. The routing experiments on an emulated version of the real NDN testbed shed light on the operations and convergence time of the OSPFN protocol [7].

The remainder of the paper is organized as follows. Section 2 presents the architectural details of Mini-CCNx in additional to a typical experiment workflow. Section 3 describes the experimental evaluation of using Mini-CCNx to run routing experiments and reproduce NDN literature results. Finally, Section 4 concludes the paper with final remarks and an outlook on the future work to improve Mini-CCNx and attend our research agenda on ICN.

II. Mini-CCNx Emulation Tool

In a nutshell, Mini-CCNx is a fork[1] of Mininet-HiFi [4] –originally proposed to emulate OpenFlow networks– augmented with several classes and mechanisms to build NDN environments using the project's official code base [9].

A. Architectural Overview

At the heart of Mini-CCNx is container-based emulation (CBE),[2] 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,

[1]Available at https://github.com/carlosmscabral/mn-ccnx
[2]http://lxc.sourceforge.net/

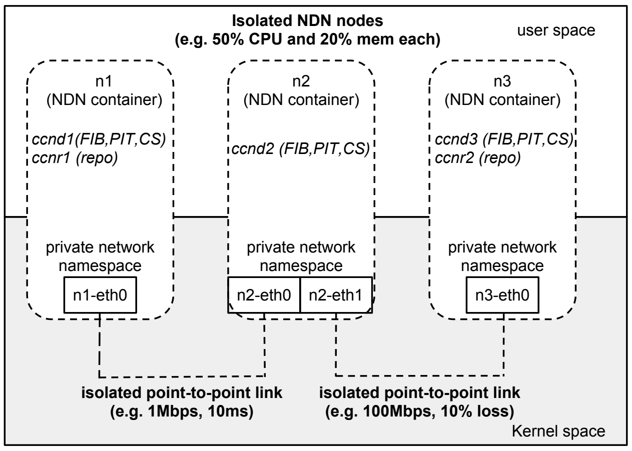
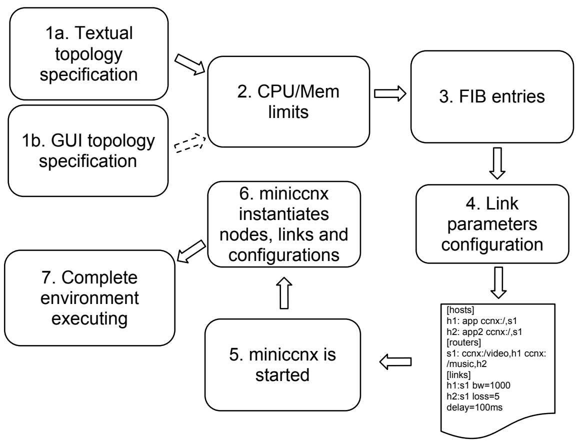


Fig. 1. Three NDN nodes connected linearly using Mini-CCNx containers.


1a. Textual topology specification
1b. GUI topology specification
2. CPU/Mem limits
3. FIB entries
4. Link parameters configuration
5. miniccnx is started
6. miniccnx instantiates nodes, links and configurations
7. Complete environment executing

```
[hosts]
h1: app ccnx:/,s1
h2: app2 ccnx:/,s1
[routers]
s1: ccnx:/video,h1 ccnx:
/music,h2
[links]
h1:s1 bw=1000
h2:s1 loss=5
delay=100ms
```

Fig. 2. Typical workflow using Mini-CCNx.

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. At this point, Mini-CCNx introduces a core difference to the OpenFlow-centric Mininet networking environment. Instead of connecting virtual hosts with software switches, in adherence to the NDN model, Mini-CCNx connects NDN nodes (containers) in a point-to-point fashion. Figure 1 illustrates the CBE and the isolation features used within Mini-CCNx to be explained in Sec. II-D.

Every software component of Mini-CCNx is built as a standalone script or as an extension (Python class) of Mininet, using standard APIs to interact with all entities, including the `ccnx` modules which do not require any modification at all. Therefore, Mini-CCNx transparently uses the newest `ccnx` version installed in the system, without requiring further updates or recompilation.

B. Fast Experimentation

The main goal Mini-CCNx is to be an convenient tool that allows experimenters to create and collect results from different NDN topologies and scenarios. To this end, all the initial configuration is done either by editing a simple text file or using a provided GUI –no extra coding is required. The typical Mini-CCNx workflow is as follows (see Fig. 2).

1. First, the user specifies the desired topology by editing the configuration text file to define the NDN nodes and their link connectivity (step *1a* in Fig. 2). Optionally, a companion GUI (`miniccnxedit`) can be used to generate a configuration file template (step *1b*).[3]

2. Next, still in the configuration file, the user specifies the CPU and memory limits for each node.

3. The user can also add default name-based FIB entries (name prefix and next hop tuples) for each node, effectively acting as a static omniscient routing protocol.

[3]As usual, the GUI is recommended for smaller topologies, when one wants a visual concept of the scenario. For scenarios with hundreds of nodes, the configuration file can be edited directly or via scripts (e.g., Bash, Python).

4. The user can specify link parameters such as bandwidth (1-1000Mbps), packet loss (in percentage), and delay (in milliseconds) for each link in the topology. At the end of this step, the configuration file is complete.

5. The user runs the `miniccnx` tool, using the created configuration file as a parameter.

6. The tool parses the configuration file, instantiates the nodes, automatically inserts FIB entries using the `ccndc` application, and applies the specified link parameters.

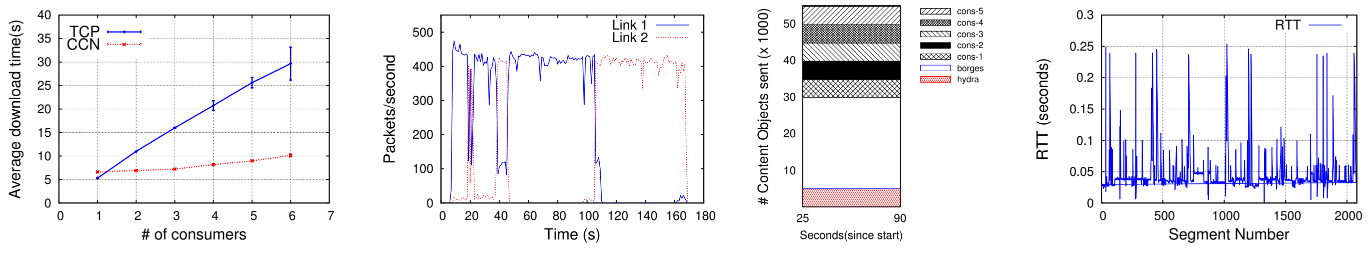
7. Finally, the environment is up and running and users can now dynamically interact with the nodes, start applications, activate dynamic routing, etc. They can also collect any needed metrics or inspect the log files.

Currently, `ccnx` connections run over a TCP or UDP. Mini-CCNx automatically and transparently adds such connectivity so the researcher only needs to be concerned about the name-based FIB entries, which can be defined statically in the configuration file (step *3*) and changed dynamically at experiment run time.

C. Complete Prototyping Environment

Being an emulated environment that runs real code, Mini-CCNx can benefit from using the same software tools used in real deployments. For instance, Mini-CCNx users can readily use custom Interest and Data packets generators (`ccnpeek`, `ccnpoke`), binary packet encoding/decoding (`xmltoccnb`, `ccntoxml`), fetching file tools (`ccnputfile`, `ccngefile`), traffic generators (`ccndelphi`, `ccntraffic`), specific ping tool (`ccnping`), routing daemons (`OSPFN`), packet dumping (`ndndump`), among other publicly available NDN code [9].

Furthermore, Mini-CCNx also includes our own developed tools to generate NDN experiments, namely `miniccnxedit` (a GUI for generating NDN topologies), `generate-linear` (a tool for automatic creation of linear topologies) and `generate-mesh` (automatic full mesh topology creation). In addition, a link-annotated replica of the 17-node NDN testbed topology is readily available. Altogether, Mini-CCNx offers a complete, fast and high productivity prototyping environment that facilitates all sorts of NDN experiments.



(a) Distribution Efficiency (b) Automatic Failover (c) Content Tx. (d) RTT w/ interval pipeline

Fig. 3. Results of replicated NDN experiments with Mini-CCNx

D. Performance and Fidelity

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

Inherited from 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.

For a performance evaluation, we refer to the technical report [3] where we discuss plentiful experiments to analyze Mini-CCNx performance dimensions such as: (*i*) scalability (number, (*ii*) coherence (sensitivity analysis to parameter changes), (*iii*) fidelity (comparison to non-emulated experiments), and (*iv*) isolation (controlled interference of the different system components under stress loads).

III. Experimental Evaluation

We now evaluate Mini-CCNx by carrying two types of experiments. First, we reproduce using Mini-CCNx experiments published in a number of papers and technical reports. We aim at showing that the experiments using Mini-CCNx exhibit great fidelity when compared to the original publications, including recent NDN project reports [11] featuring experiments with NDN applications under real testbed conditions. Second, we carry a number of routing experiments to showcase how Mini-CCNx eases the task of analyzing the content-oriented routing behavior on top of a non-trivial topology. Hopefully these examples might give researchers ideas of how Mini-CCNx can be used in order to deploy and evaluate new routing protocols or forwarding strategies for the CCN model.

A. Reproducing experiments from published work

Content Distribution Efficiency. The original paper by Van Jacobson *et al.* [6] shows how CCN shines when it comes to content distribution. The data sharing performance between TCP and CCN was compared by measuring the total time taken to simultaneously retrieve a 6MB data file over a network bottleneck. The experiment uses a source node over a 10 Mbps shared (bottleneck) link connected to a cluster of consumers with 1Gbps links simultaneously pulling the data file. We used Mini-CCNx to reproduce such behavior. For TCP, the file was retrieved with `wget` from a HTTP server in the source. For CCN, the file was retrieved with the `ccnx`'s built-in `ccngetfile`. Figure 3(a) shows the results expected from the original paper: CCN outperforms TCP with two consumers or more due to caching alleviating the bottleneck link.

Strategy/Forwarding Layer. The original paper shows how a CCN node behaves when multiple FIB entries are available for the same prefix. The strategy layer chooses the best face at the time, and if the current link fails, the CCN stack automatically sends packets over the second face. We set up Mini-CCNx with two directly connected hosts and equal link configuration (100Mbps, 1ms delay), for link 1 and link 2. The first host constantly sends Interest packets while the second replies with Data packets. Figure 3(b) shows the bandwidth in each of these links according to time. We can observe how the strategy layer chooses mostly link 1 but constantly sends probe packets through link 2. Approximately at 40s, when the strategy seems to choose mostly link 2, we disconnect it at 45s until about 70s. We can verify later how the strategy layer automatically chooses the best link again and hence successfully reproduce with Mini-CCNx the automatic failover capabilities.

NDNVideo. The video streaming application (TR-NDN-0007 [11]) allows to play live and pre-recorded videos with random access and no session negotiation. During the tests, the developers detected an interesting behavior in a deployment with multiple consumers, a central content router (*borges*) and a video producer (*hydra*): each consumer got slightly more data than the *hydra* provides. Using Mini-CCNx with the same topology and 5 consumers, we can reproduce this behavior (Fig. 3(c)). In addition, the authors detected large spikes in RTT measurements and concluded this was caused by an Interest reordering done by CCNx. Authors made several client-side improvements (e.g., timeout estimation, Interest re-issue, usage of interval-based pipeline) to address this issue. Figure 3(d) shows the observed RTT behavior of the latter approach using Mini-CCNx. The results are really close to the published ones (cf. Fig. 14 in TR-NDN-0007 [11]).

B. Routing Experiments

To further illustrate the types of routing experiments that can be done with Mini-CCNx, we now will bring some

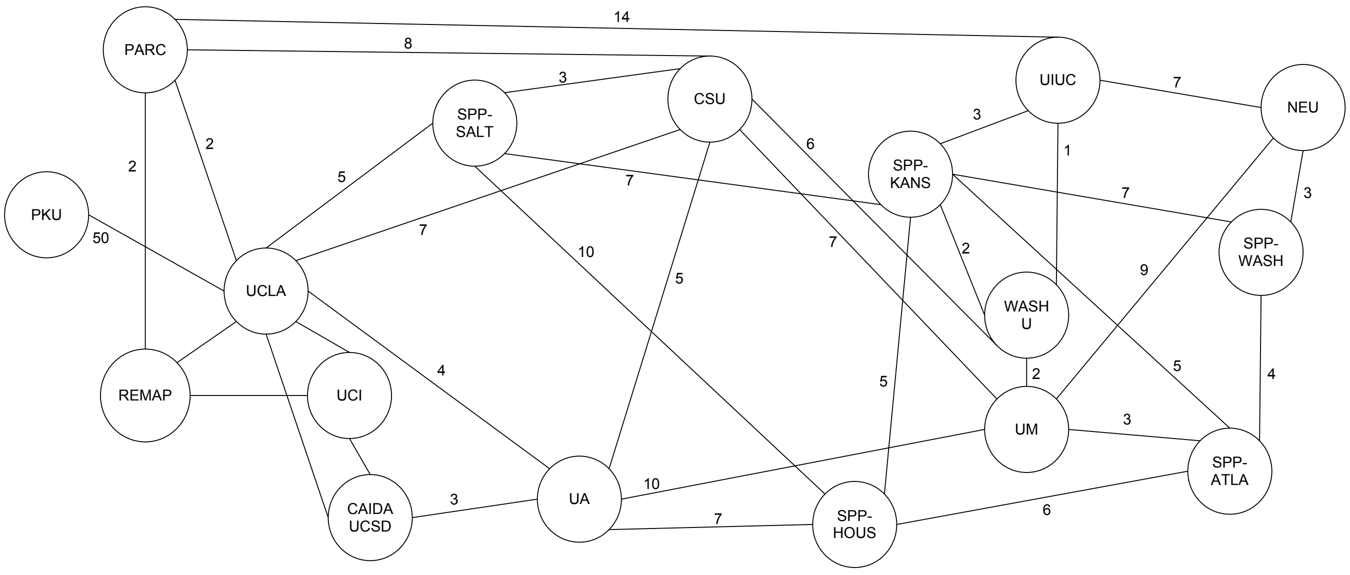
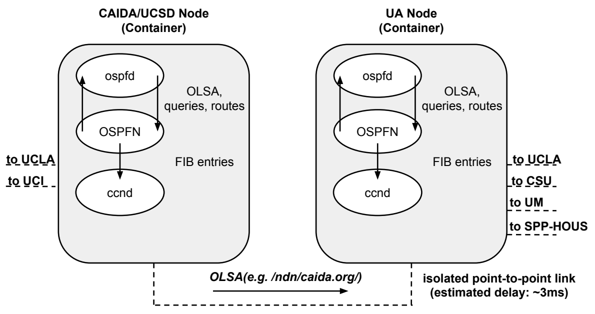


Fig. 4. NDN Testbed and link delays (ms).



(a) Overall view of the OSPFN routing scheme in the NDN testbed.

PKU
/ndn/pku.edu.cn/
UCLA
/ndn/ucla.edu/
NEU
/ndn/neu.edu/northpole/
/ndn/opschat/
CSU
/ndn/colostate.edu/netsec/
/ndn/opschat/

(b) Sample of prefixes announced by NDN nodes.

UCLA OLSA	
LS Type	10
Opaque Type	236
Announcing Router	0.0.0.14 (UCLA)
Opaque Info	
14 (prefix size in bytes)	
/ndn/ucla.edu/	

(c) OLSA format.

Fig. 5. NDN Testbed and OSPFN protocol.

routing scenarios and their respective analysis. All scenarios use the NDN Testbed topology shown in Fig. III-B. A Mini-CCNx user can simply run the command `sudo miniccnx -testbed` to automatically load 17 nodes representing the NDN testbed deployment including link delay estimates.[4]

Each NDN node runs its own instance of the `ospfn` [11] daemon in addition to the the required `ospfd` and `zebra` daemons from the Quagga routing suite. The nodes are configured to announce exactly the same name prefixes outlined in the NDN testbed website [12]. For instance, as shown in Fig. 5(b), the CAIDA/UCSD router announces the `/ndn/caida.org/` prefix, and the PARC router announces the `/ndn/parc.com` and `/ndn/opschat` prefixes.

We will show through a series of experiments how Mini-CCNx allows users to easily bring any link up or down and observe the dynamic behavior of OSPFN, including convergence times and multipath support. This brings greater agility and flexibility compared to trying these operations on the real testbed, allowing to study new software versions and experiments before going live.

[4]As seen on August 8th 2013 [12]. The propagation delays were estimated using straight geographical node distances.

Understanding the OSPFN Protocol

OSPFN [7] is an extension to OSPF (Open Shortest Path First) for routing in NDN based on a new type of opaque link state announcement (LSA) to carry name prefixes in routing messages. Best next-hops are installed to each name prefix in the FIB and operators are allowed to manually configure a list of alternative next-hops in addition to the best one. Each node in the topology announces the prefixes they are responsible for. Figure 5(b) shows some prefixes and the respective OLSA format (Fig. 5(c)) of the OSPFN prefix announcement.

The OSPFN protocol processes all the received messages (updates, insertions or deletions) and dynamically updates an internal table that maps prefixes into their respective origin routers. To illustrate the basic operation, Figure 6 shows a piece of the UCLA node's FIB, as given by the output of the `ccndstatus` command.

The `/ndn/colostate.edu/netsec/` and `/ndn/pku.edu.cn/` prefixes are announced by unique origin routers (CSU and PKU, respectively) that are directly connected to UCLA (1 hop cost). The `/ndn/neu.edu/northpole/` prefix is also announced

```
...
ccnx:/ndn/colostate.edu/netsec face: 8 flags: 0x3 expires: 214748301   (face 8 = CSU)
ccnx:/ndn/pku.edu.cn face: 10 flags: 0x3 expires: 2147483012   (face 10 = PKU)
ccnx:/ndn/neu.edu/northpole face: 13 flags: 0x3 expires: 2147483012   (face 13 = UA)
ccnx:/ndn/neu.edu/northpole face: 9 flags: 0x3 expires: 2147483012   (face 9 = PARC)
ccnx:/ndn/neu.edu/northpole face: 8 flags: 0x3 expires: 2147483012   (face 8 = CSU)
ccnx:/ndn/opschat face: 13 flags: 0x3 expires: 2147483167   (face 13 = UA)
ccnx:/ndn/opschat face: 9 flags: 0x3 expires: 2147483167   (face 9 = PARC)
ccnx:/ndn/opschat face: 8 flags: 0x3 expires: 2147483167   (face 8 = CSU)
...
```

Fig. 6. UCLA's FIB (partial)

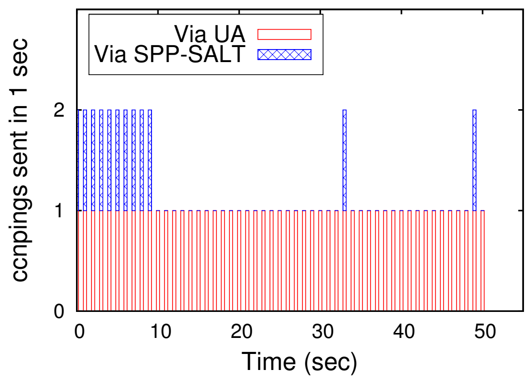


Fig. 7. *ccnpings* leaving CSU towards SPP-HOUS via UA and via SPP-SALT

by only one router (NEU) but there are 3 entries for such prefix in UCLA's FIB. This occurs because there are 3 equal-cost paths (3 hops) between UCLA and NEU, as shown in Fig. III-B. The order of such insertions, in this case, is arbitrary. Finally, the `/ndn/opschat` is announced by two routers. Therefore, the respective entries to the origin routers are created: to CSU (route (UCLA)-(CSU) with only one hop) and to NEU (routes (UCLA)-(PARC)-(UIUC)-(NEU) and (UCLA)-(UA)-(UM)-(NEU), both with a 3-hop cost). In this last case, the insertion order is important: the route to CSU (smaller cost) must be the last one to be inserted to give this route a higher priority (`ccnd` implementation convention).

Relationship between Routing and Forwarding

The next experiment shows how the NDN model exhibits a closer relationship between the routing and forwarding planes when compared to the traditional TCP/IP model. Suppose that the CSU node wants to communicate with the SPP-HOUS node, which announces `/ndn/spphous1`. There are two 2-hop cost routes between these nodes: (CSU)-(UA)-(SPP-HOUS) and (CSU)-(SPP-SALT)-(SPP-HOUS). This can be verified by the presence of the 2 FIB entries at CSU (see Fig. 8).

```
...
ccnx:/ndn/spphous1 face: 7 flags: 0x3 expires: 2147483612   (face 7 = UA)
ccnx:/ndn/spphous1 face: 11 flags: 0x3 expires: 2147483612   (face 11 = SPP-SALT)
...
```

Fig. 8. CSU's FIB (partial)

Despite the paths' equal costs (2 hops), the propagation delay of the first path (12 ms) is slightly smaller than the second path delay (13 ms). We then activate the `ccnping` in CSU towards SPP-HOUS. Using the `ndndump` sniffer, we can see that the first sent Interest uses the second path (via SPP-SALT). Note however that the node also sends another Interest (which is exactly the same as the first one) through the first path (via UA). This prospective behavior continues for about 10 seconds, as can be seen in Fig. 7.

For the first 10 seconds, the average RTT for such packets was measured in both paths. Via UA (first path), a 29,5 ms average RTT (with 0,41 ms deviation) was measured. In the second path (via SPP-SALT), the average RTT was 30ms with 0,46ms deviation. Therefore the forwarding plane chose the first path as it detected a better network condition in such route. We can see that later the forwarding plane tries to send some packets through the second path again, but this did not change the prior option of using the first path most of the time.

Topology Changes and Routing Convergence

The next experiment focuses on analyzing the OSPFN behavior with regards to topology changes and the protocol convergence time. Starting with the same routing topology of Fig. III-B, we can observe in Figure 5(b), we can observe that the UCLA node has an entry for the `/ndn/colostate.edu/netsec/` prefix pointing directly to the CSU node, which is the origin router for such prefix and which is directly connected to the UCLA node. With the command `link ucla csu down`, Mini-CCNx removes this link between the nodes at 13:23:20. Figure 9 shows the `ospfd` daemon log and UCLA's FIB just after this link removal.

```
2013/03/26 13:23:20 OSPF: nsm_change_state(0.0.0.2, Full -> Deleted): scheduling new router-LSA
origination
2013/03/26 13:23:20 OSPF: ospf_apiserver_nsm_change
2013/03/26 13:23:20 OSPF: DR-Election[1st]: Backup 0.0.0.0
2013/03/26 13:23:20 OSPF: DR-Election[1st]: DR     1.0.0.26
2013/03/26 13:23:20 OSPF: interface 1.0.0.26 [298] leave AllSPFRouters Multicast group.
2013/03/26 13:23:20 OSPF: interface 1.0.0.26 [298] leave AllDRouters Multicast group.
2013/03/26 13:23:20 OSPF: ospf_apiserver_ism_change
2013/03/26 13:23:20 OSPF: oi->ifp->name=ucla-eth6
2013/03/26 13:23:20 OSPF: old_state=7
2013/03/26 13:23:20 OSPF: oi->state=1
2013/03/26 13:23:20 OSPF: Router routing table for      0.0.0.2 updated  (0.0.0.2 = CSU)
2013/03/26 13:23:20 OSPF: Router routing table for      0.0.0.3 updated  (0.0.0.3 = NEU)
2013/03/26 13:23:20 OSPF: Router routing table for      0.0.0.7 updated  (0.0.0.7 = SPP-ATLA)
2013/03/26 13:23:20 OSPF: Router routing table for      0.0.0.8 updated  (0.0.0.8 = SPP-HOUS)
2013/03/26 13:23:20 OSPF: Router routing table for     0.0.0.16 updated  (0.0.0.16 = UM)
2013/03/26 13:23:20 OSPF: Router routing table for     0.0.0.17 updated  (0.0.0.17 = WASHU)
```

(a) `ospfd` log of the UCLA node.

```
...
ccnx:/ndn/colostate.edu/netsec face: 13 flags: 0x3 expires: 2147483207   (face 13 = UA)
ccnx:/ndn/colostate.edu/netsec face: 12 flags: 0x3 expires: 2147483207   (face 12 = SPP-SALT)
ccnx:/ndn/colostate.edu/netsec face: 9 flags: 0x3 expires: 2147483207   (face 9 = PARC)
...
```

(b) UCLA's FIB

Fig. 9. `ospfd` behavior and UCLA's FIB after link down event.

The first observed message in the log shows the link removal detection and the respective update LSA is sent. Next, the **ucla-eth6** interface is removed from shortest-path multicast group in order to trigger a new route calculation. Finally, all the affected entries are updated in the OSPF routing table. Note that all this operation was done during the same second (13:23:20). Also, we can note that all the other updates on the other node's FIB also occur during this same second and thus we can conclude that the answer was very efficient. OSPFN also detects such connectivity updates and reinserts new FIB entries to the affected prefixes using the supplied information from the `ospfd` daemon. Figure 9(b) shows a section of the FIB which now has 3 new 2-hop paths between UCLA and CSU - namely (UCLA)-(PARC)-(CSU), (UCLA-SPP-SALT)-(CSU) and (UCLA)-(UA)-(CSU).

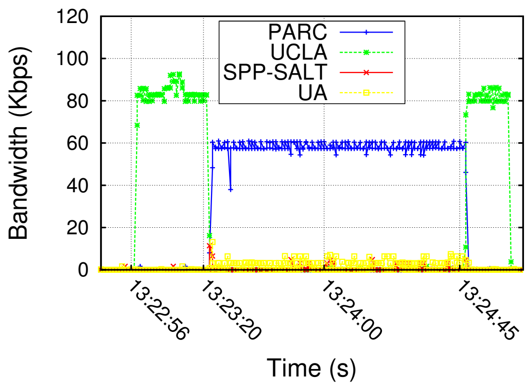


Fig. 10. Bandwidth behavior on the different links relevant to the experiment.

To restore the link we run the `link ucla csu up` command and observe the routing behavior after the link is reestablished.We can observe from the logs (not shown) that `ospfd` notices the re-connection at 13:24:00. Next, the daemon operation continues until 13:24:45, when the update LSA is finally sent and the node's table is updated. At the same time, we see that UCLA's FIB is immediately updated and the `/ndn/colostate.edu/netsec/` FIB entry is back. Therefore, the total routing reestablishment took 45 seconds, taken mostly by the OSPF routing process to re-converge.[5]

During all this experiment, the traffic generator `ccntraffic` in UCLA generated a constant and low traffic towards CSU which replied with random 1024-byte Content Objects. The bandwidth was measured in 4 points directly attached to CSU: *(i)* PARC-CSU, *(ii)* UCLA-CSU, *(iii)* SPP-SALT-CSU and *(iv)* UA-CSU. Figure 10 shows the bandwidth behavior. Furthermore, the markings on the x axis point to the events cited earlier in this experiment.

Note that at 13:22:56 the traffic generator was turned on and averaged about 80 Kbps using the direct link between CSU and UCLA. At 13:23:20, when the link is removed, we observe the fast response from the forwarding plane, which chooses the new best path through PARC but also results in lower average bandwidth (about 60 Kbps). During this time, we can note some `ccnd` daemon attempts to find new best links (smaller 5Kbps peaks) but the forwarding plane keeps choosing the PARC path. At 13:24:00, the link between CSU and UCLA is reconnected but now this link is not immediately used. This confirms the detected behavior shown on the logs. At 13:24:45 we note that the UCLA-CSU link is reestablished.

IV. Conclusion and Future Work

Inspired by the well-succeeded experience in fast prototyping for SDN, Mini-CCNx appears as a useful tool for NDN experimentation platforms. Mini-CCNx is realistic, low-cost and scalable: a whole content-centric network, with hundreds of nodes, can be run in commodity hardware, with easy configuration and high-fidelity results. The latter has been verified by a series of experiments where we were able to replicate results from published work. In addition, routing experiments on an emulated version of the real NDN testbed has shed light on the a first attempt towards a content-centric routing protocol. Altogether, we conclude that the Mini-CCNx emulator is a helpful tool for NDN researchers, a fact that we are verifying by the increasing user activity in the public open-source repository.

Our future work can be divided into two main threads. Firstly, we are working on enhancements to the Mini-CCNx tool, including (*i*) user-defined CCN-specific metrics (e.g., cache efficiency, path performance, Interest/Data ratio), (*ii*) advanced GUI to allow link annotations, FIB additions, etc., and (*iii*) support of distributed environments to further scale the experiments. Secondly, we will continue our research efforts in questions such as performance comparison of recent routing protocols (i.e. NLSR [5]), efficiency of strategy layers, caching techniques, and so on.

[5]The Quagga daemon used the default configuration and was not optimized for the experiment, which explains the high convergence time.

References

[1] B. Ahlgren, C. Dannewitz, C. Imbrenda, D. Kutscher, and B. Ohlman. A survey of Information-Centricentric Networking. *Communications Magazine, IEEE*, 50(7):26–36, 2012.

[2] C. M. Cabral, C. E. Rothenberg, and M. F. Magalhães. Mini-CCNx: Fast Prototyping for Named Data Networking. In *Proceedings of the 3rd ACM SIGCOMM Workshop on Information-centric Networking*, ICN '13, pages 33–34, New York, NY, USA, 2013. ACM.

[3] C. M. Cabral, C. E. Rothenberg, and M. F. Magalhães. Mini-CCNx Tech Report TR001. https://github.com/carlosmscabral/mn-ccnx/wiki/Publications, March 2013.

[4] N. Handigol, B. Heller, V. Jeyakumar, B. Lantz, and N. McKeown. Reproducible network experiments using container-based emulation. *CoNEXT '12*, page 253, 2012.

[5] A. K. M. M. Hoque, S. O. Amin, A. Alyyan, B. Zhang, L. Zhang, and L. Wang. Nisr: 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.

[6] V. Jacobson, D. K. Smetters, J. D. Thornton, M. F. Plass, N. H. Briggs, and R. L. Braynard. Networking named content. In *CoNEXT '09*, Dec. 2009.

[7] L. Wang, A. M. Hoque, C. Yi, A. Alyyan, and B. Zhang. OSPFN: An OSPF based routing protocol for Named Data Networking. Technical Report NDN-0003, July 2012.

[8] B. Lantz, B. Heller, and N. McKeown. A network in a laptop. In *ACM SIGCOMM Workshop - Hotnets '10*, pages 1–6, Oct. 2010.

[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] NDN Project. NDN Technical Reports. http://www.named-data.net/techreports.html.

[12] NDN Testbed. NDN Routing Topology. http://netlab.cs.memphis.edu/script/htm/topology.html.

[13] M. Sarela, C. Rothenberg, T. Aura, A. Zahemszky, P. Nikander, and J. Ott. Forwarding anomalies in bloom filter-based multicast. In *INFOCOM, 2011 Proceedings IEEE*, pages 2399–2407, 2011.

[14] S. Soltesz, H. Pötzl, M. E. Fiuczynski, A. Bavier, and L. Peterson. Container-based operating system virtualization. *ACM SIGOPS Operating Systems Review*, 41(3):275, June 2007.

[15] G. Xylomenos, C. Ververidis, V. Siris, N. Fotiou, C. Tsilopoulos, X. Vasilakos, K. Katsaros, and G. Polyzos. A survey of information-centric networking research. *Communications Surveys Tutorials, IEEE*, PP(99):1–26, 2013.