Experimental Evaluation of Content Distribution with NDN and HTTP

Haowei Yuan and Patrick Crowley
Computer Science and Engineering
Washington University
St. Louis, Missouri 63130
Email: {hyuan, pcrowley}@wustl.edu

***Abstract*—Content distribution is a primary activity on the Internet. Name-centric network architectures support content distribution intrinsically. Named Data Networking (NDN), one recent such scheme, names packets rather than end-hosts, thereby enabling packets to be cached and redistributed by routers. Among alternative name-based systems, HTTP is the most significant by any measure. A majority of today's content distribution services leverage the widely deployed HTTP infrastructure, such as web servers and caching proxies. As a result, HTTP can be viewed as a practical, name-based content distribution solution. Of course, NDN and HTTP do not overlap entirely in their capabilities and design goals, but both support name-based content distribution.**

This paper presents an experimental performance evaluation of NDN-based and HTTP-based content distribution solutions. Our findings verify popular intuition, but also surprise in some ways. In wired networks with local-area transmission latencies, the HTTP-based solution dramatically outperforms NDN, with roughly 10x greater sustained throughput. In networks with lossy access links, such as wireless links with 10% drop rates, or with non-local transmission delays, due to faster link retransmission brought by architectural advantages of NDN, the situation reverses and NDN outperforms HTTP, with sustained throughput increased by roughly 4x over a range of experimental scenarios.

I. Introduction

Content distribution makes up a significant portion of today's Internet traffic [1]. The redundancy in content distribution network traffic is widely understood and has been discussed in a substantial body of literature [2][3]. Correspondingly, many new Internet and Internet-scale architectures have been proposed to address this issue [4][5][6][7].

Name-centric network architectures shift emphasis away from identifying end-hosts and toward identifying data. For content distribution, this enables the caching and reuse of content in the network. To achieve this goal, data requests need to have unique *names*, and there need to be in-network storage elements that can cache the data and respond to matching requests. Following this approach, a distributed caching system is overlaid across the network, and cross-network traffic can be reduced when there is a *cache hit*. All recent content-centric or information-centric network designs [7] [8] share the name-centric concept.

Named Data Networking (NDN) [9] is a recently proposed network architecture that follows the name-centric approach. In NDN, content is named at the granularity of individual packets, and packets are routed and forwarded based on their names. There are two types of packets in NDN networks, *Interest* packets and *Data* packets. Clients emit Interest packets containing the name of the requested content. Interest packets follow hierarchically organized, name-based routes toward data sources, which reply with Data packets containing both the name and its associated data. NDN routers cache Data packets by treating packet buffers differently. Where IP routers empty old packet buffers and enqueue them on a free list for subsequent use, NDN routers treat packet buffers like entries in a cache indexed by their names. In this way, compare to IP routers, NDN enables caching without requiring a net increase in packet buffer memory.

In addition to recently proposed name-centric network architectures, the HTTP protocol and its globally deployed infrastructure have rightly been identified as a name-centric system [10], and can arguably be considered the new narrow waist of the Internet. Indeed, URLs are the names that matter most in today's Internet. In HTTP, the requested URL in the HTTP header is the content name. Today's HTTP infrastructure, including both web servers and caching proxies, can be viewed as providing in-network storage for named HTTP data. Traditionally, HTTP has been used for transferring files, but many web-based applications have emerged in recent years that run as HTTP overlays, including video and file distribution.

The purpose of this paper is to evaluate the effectiveness of NDN and HTTP as content distribution systems over a range of experimental scenarios. Many feel that name-based approaches are well-suited to guide the future evolution of the Internet. To that end, it is important to understand the practical limits and relative merits of today's designs and implementations, in order to inform the design of more scalable future systems.

In this paper, we measure NDN-based and HTTP-based content distribution performance in a series of controlled, repeatable experiments on a testbed of real systems. The NDN-based solution uses *CCNx*, an open-source NDN software prototype developed by PARC. The HTTP-based solution employs two widely used commercial grade open-source systems: the *lighttpd* web server and the caching web proxy *Squid*. While it is in a way unfair to compare CCNx to lighttp/Squid, since the latter have enjoyed many years of developer investment in performance-tuning and evolutionary development, we feel that our observations highlight the strengths of both, and

can be used to drive future CCNx development.

We compared CCNx and lighttpd/Squid content distribution performance by distributing a file to up to 40 end-hosts in a variety of networks that feature a range of packet drop probabilities on access links (from 0% to 10%) and a range of core network packet transmission delays (from 0ms to 200ms). Our results show that lighttpd/Squid distributes files faster in ideal networks by a factor of 10x, since it is implemented more efficiently than CCNx. While the implementation of CCNx is not optimized, its NDN architectural advantages allow it to outperform Squid in networks with lossy access links by a factor of 4x.

As a small step toward improving CCNx efficiency, we made a modest change to the CCNx implementation and observed a throughput improvement of 40%.

II. Experimental Setup

In this section, we introduce the hardware platforms and software tools used in our experiments.

A. Testbed

We conducted the performance study in the *Open Network Laboratory* (ONL) [11]. We used up to 48 single-core machines from the testbed. Each machine was equipped with a 2.0 GHz AMD Opteron Processor, 512 MB memory and a 1 Gbps network interface. These machines could be connected via virtual switches and/or *Network Processor-based Routers* (NPRs) to form an isolated network.

B. CCNx Software Tools

A set of software tools, mostly provided by the CCNx software distribution, and some written on our own, were used to perform file distribution. The CCNx software evaluated in this paper is ccnx-0.4.0, released on September 15, 2011. The core component, *ccnd* daemon, implements NDN packet forwarding and caching function. In our experiments, *ccnd* is configured with all default environment variable values. The underlying transportation protocol we chose is TCP.

The CCNx built-in *ccncatchunks2* program generates a sequence of Interest packets. These Interest packets share a common prefix. The generated Interest starts with name ccnx:/filename/0, where the last portion is the chunk index. The chunk index is increased by 1 for each generated Interest. Note that each generated Interest packet requests a chunk of a file. In NDN, large files are chunked, and each chunk is transferred using one Data packet. Because there is no suitable server side application in the CCNx distribution, we built a server side program, *ccnfileserver*, which generates Data packets with content fetched from files on the server.

C. HTTP and Web-Caching Software Tools

We chose the open-source HTTP server *lighttpd*-1.4.28 and the web-caching software *Squid*-3.1.11 for evaluating HTTP and Web-Caching system performance. Since both programs are widely used, we felt comfortable using their default configuration in our study. The *wget* program was used to download files.

III. File Distribution Performance

In this section, we compare the performance of distributing files using NDN- and HTTP-based solutions. We first evaluate their performance under ideal, wired network conditions, and then study their performance when packet loss and delay exist, which essentially emulates a wireless network. The metric we measure is *Download Time* (DT), which is defined as the time from when a client application sends a request for a file until the file is downloaded completely. The factors to be studied are summarized in Table I. Note that some preliminary results were included in our ANCS 2011 short paper [12].

TABLE I
Factors to Be Studied for File Distribution Performance

Factors	Description
Levels of Cache	1, 2
Number of Clients	5, 10, 15, 20, 25, 30, 35, 40
Link Loss Rate	5%, 10%
Link Delay (ms)	0, 50, 100, 200

A. Experimental Configuration

The file distribution topology includes 40 client hosts, one server and two levels of intermediate nodes, which can be either CCNx routers or Squid proxies. Every eight clients form a *cluster* and share a common second level intermediate node. All the hosts in the testbed were connected via 1 Gbps links. Figure 1 shows the experiment topology. In the CCNx case, clients run the *ccnd* daemon and the *ccncatchunks2* application to download the file. The second level intermediate nodes, which are CCNx routers, run the *ccnd* daemon. The server runs a *ccnd* daemon and the $ccnfileserver$ program. For the HTTP case, clients run *wget* to fetch data. The second level nodes run $Squid$ proxies, and the server runs a $lighttpd$ web server. To perform file distribution, a 100 MB file is stored in the server, and the clients try to fetch the file simultaneously. The file download time DT is recorded on each client host. The name used by CCNx is ccnx:/repo/100m.txt, and the URL http://192.168.55.33/100m.txt is used in the HTTP-based scenario.

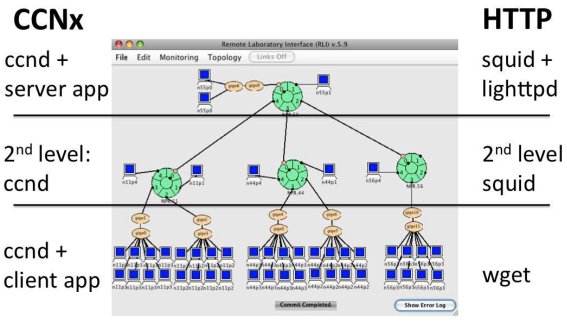

Fig. 1. File Distribution in Ideal Networks

B. CCNx vs. Lighttpd

We first evaluate the performance of downloading a 100MB file using CCNx and lighttpd directly without a caching proxy. The factor to be studied is the number of clients in the system. Every cluster has the same number of active clients. We start with five clients requesting the file, with one client in each cluster. The number of active clients is increased by one in the subsequent tests until all of the clients are active. For each configuration, we run the experiment five times and note the average DT across all the active clients.

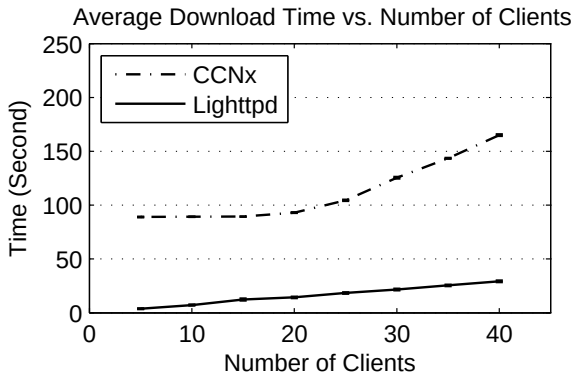

Fig. 2. CCNx vs. Lighttpd (90% Confidence Interval)

The average file download times for CCNx and lighttpd are shown in Figure 2. When the clients fetch files from a lighttpd server directly, the average file download time increases linearly as the number of clients increases. The lighttpd server's physical link (1 Gbps) is saturated and thus becomes the bottleneck. In the case of CCNx, when there are fewer than 20 clients, the file download time is stable regardless of how many clients are fetching the file. When there are more than 20 hosts, the download time increases. When there are fewer than 20 hosts, our simple CCNx server program $ccnfileserver$ cannot chunk the file and reply with Data packets efficiently, and thus it becomes the bottleneck. It is worth noting that although our $ccnfileserver$ is the bottleneck when the number of clients is less than 20, we are more interested in whether the *ccnd* daemon is scalable, which can be justified by increasing the number of clients. As shown in Figure 2, when there are more than 20 hosts, the *ccnd* daemon host is saturated and becomes the bottleneck. Figure 2 shows that due to the forwarding throughput bottleneck, CCNx content distribution performance is worse than using the HTTP server directly.

C. CCNx vs. Squid

We also compared the file distribution performance using CCNx and Squid. The factors we studied in this performance evaluation are the levels of cache and the number of clients in the system. In the single level case, all the clients connect to the server through the top level CCNx router or Squid proxy, and the second level intermediate nodes are unused. In the two level case, clients are connected via a second level cache. We varied the number of clients the same way as we did in Section III-B. For each configuration, we also ran the experiment five times and recorded the average DT across all the active clients. Figure 3 shows the results.

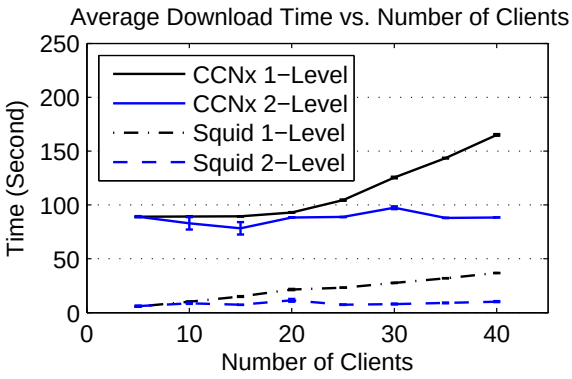

Fig. 3. CCNx vs. Squid (90% Confidence Interval)

For the single level cache case, the situation is similar to Section III-B, because Squid behaves like the lighttpd to some extent in this setup. The file download time increases linearly with the number of clients: the Squid implementation is very efficient, and the proxy host's physical outgoing link capacity (1Gbps) becomes the bottleneck. In the CCNx case, the downloading time stays relatively constant until there are more than 20 hosts, which is the same as in Section III-B. The *ccnd* daemon becomes the only bottleneck when there are more than 20 clients.

For the two-level cache case, both Squid and CCNx perform much better, as the physical link bottleneck and the *ccnd* daemon processing bottleneck disappear. Overall, the current CCNx implementation is about 10 times slower than Squid.

D. Lossy Network Condition

The architectural advantage of NDN is that every router is capable of caching data, therefore, NDN's ability to act as a

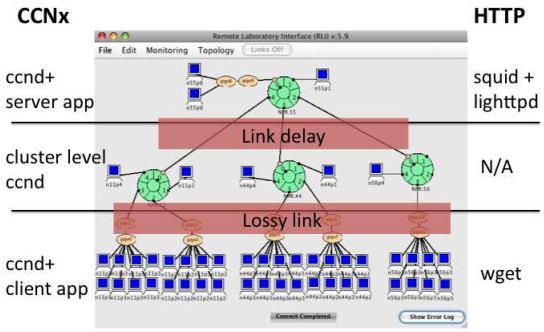

Fig. 4. File Distribution in Lossy Networks

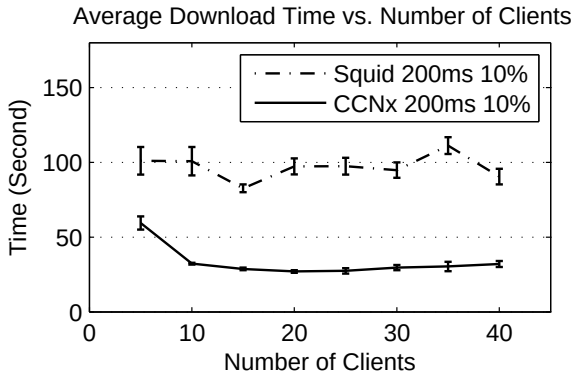

Fig. 5. CCNx vs. Squid in Lossy Networks (90% Confidence Interval)

link-level packet retransmission mechanism potentially brings many benefits. In this section, we present the performance of CCNx-based and Squid-based content distribution methods in lossy network conditions. Essentially, we emulate wireless access, where packet loss exists between the end-hosts and the access point, and there is a delay between the access point to the server or to the content cache next to the server.

To emulate a lossy link, we installed the $Rand\_drop$ plugin, which probabilistically selects and drops packets on the NPRs. To add delay to a network link, we added a $Delay$ plugin to an NPR connected with the link. At each of the NPR ports, filters were installed to direct the packets going through this port to the plugin before they are forwarded. Figure 4 shows the topology we used for this test. The topology looks the same as in the ideal network case, with the addition of filters and $Rand\_drop$ and $Delay$ plugins on the NPRs. In addition, to reflect the differences between NDN and current IP networks, there is cluster-level caching for CCNx but no Squid proxy at the cluster level. This is a reasonable design as in the NDN network every router is a NDN/CCNx router, while in IP network, it is not common for content publishers or ISPs to deploy a Squid cache close to a wireless access point.

The performance metric for these two content distribution methods in lossy networks was still the download time. The clients downloaded a 1 MB file, rather than 100MB files used in the previous experiments, because the lossy conditions resulted in much longer download times. We varied the delay times among 0ms, 50ms, 100ms, and 200ms. We set the link loss rate at either 5% and 10%. Figure 5 shows CCNx- and Squid-based content distribution performance when there is 10% link loss rate and 200ms delay. CCNx performs much better in this network condition. For the CCNx case, there is a local cache for each cluster, and the TCP retransmission can be finished quicker. In addition, the benefits of a local cache become larger when there are more clients requesting the same file concurrently. As we can see on Figure 5, when there is a single client at each cluster, CCNx outperforms Squid. As the number of clients per cluster increases, the average download time using CCNx decreases, which is what we expected. While the Squid performance is relatively stable as the number of clients increases. For instance, when there are 20 clients, i.e., 4 clients per cluster, the performance of distributing files using CCNx is about 4x faster than Squid. When there are 20 - 40 clients, the performance of CCNx file distribution stays stable.

We also studied the impacts of link delay and link loss rate on file distribution performance. Figure 6 shows that the file distribution performance degrades as the delay time is increased. For 200ms and 100ms delay, Squid performs considerably worse than CCNx, but for 50 ms delay, there is no big difference between these two methods. When there is no delay, the Squid performs slightly better. Figure 7 shows when the packet loss rate increases, the file distribution performance becomes worse for both CCNx and Squid.

When using TCP, the interaction between packet drop rates and round-trip time is important. While much could be said about this, we point out that the NDN architecture simplifies these dynamics considerably by shortening paths.

IV. Improving CCNx Performance

Our performance study in Section III shows that CCNx can potentially outperform other content distribution methods by leveraging the architectural advantage of NDN design. However, the current performance of the CCNx implementation could prevent it from being widely adopted.

CCNx employs an XML encoding scheme to encode packets to wire format. In our previous work [13], we found packet name decoding to be time consuming. The original CCNx implementation stores content with their names encoded in the Content Store (CS), as a result, when the CS is queried, several content names might need to be decoded, which slows down the program. We made a simple change to the CCNx software such that decoded content names are stored in the Content Store. This way, when a CS query happens, the decoded names can be used directly for name comparison.

We measure the peak throughput of the top-level CCNx cache shown in Figure 1. In this experiment, there is only 1 level of cache being used such that the CPU of the top-level CCNx cache is guaranteed to be saturated. We use the same metrics, *Outgoing Throughput* and *Incoming Throughput*, as in [13]. The measured peak throughput values are shown in Figure 8. The peak throughput of the modified CCNx implementation is about 40% higher than the original implementation.

It should be noted that the code change we made only reduces Content Store lookup time, and the Content Skip List query specifically. Eventually, when the number of diverse Interest names increases, the forwarding lookup and Data packet prefix match lookup will likely take more time.

V. Related Work

Previous work closely related to this study can be classified in two categories, and here we mention the most relevant works in each. The first category includes NDN forwarding plane design, CCNx performance analysis and evaluation. Detailed CCNx data structures and operational flows as well as CCNx's baseline performance were first discussed in [13]. [14]

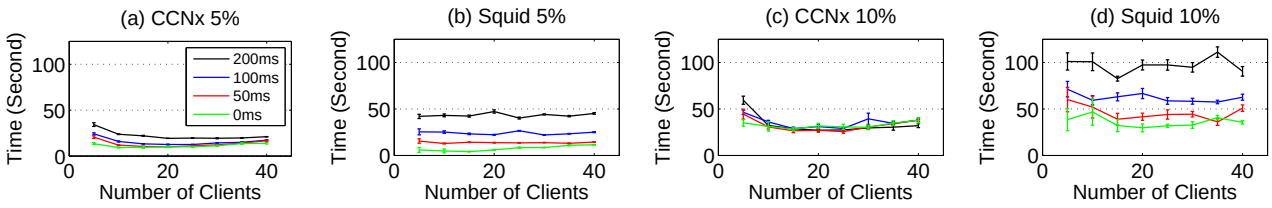

Fig. 6. Link Delay (90% Confidence Interval)

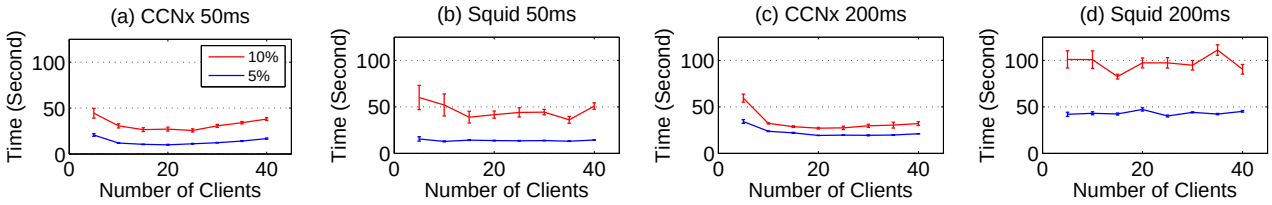

Fig. 7. Link Loss Rate (90% Confidence Interval)

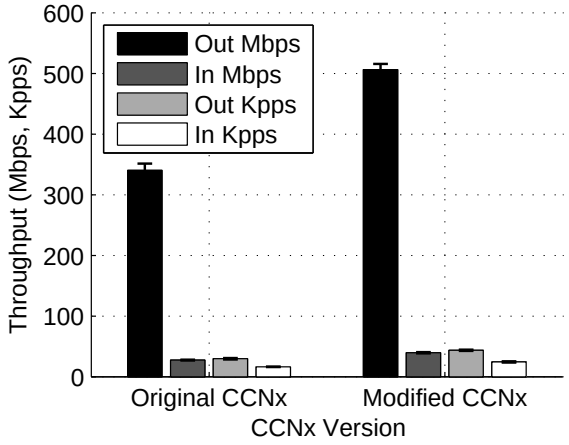

Fig. 8. Original and Modified CCNx Peak Throughput

applied name encoding schemes to reduce memory requirement in NDN. [15] modeled and simulated the performance of content distribution using CCNx. None of these works have done large-scale CCNx content distribution measurements. The second category concerns the performance of alternative content distribution methods. IMP [16] studied improving P2P content distribution performance by using the locality information available to ISPs. CoBlitz[17] broke large files into chunks and performed content distribution via HTTP.

VI. Conclusion and Future Work

We have experimentally evaluated the NDN prototype and compared its content distribution performance to that of an HTTP-based alternative. Our performance results show that the CCNx implementation of NDN cannot sustain gigabit line rates. However, the results also demonstrate the advantages of applying the NDN architecture in networks with lossy access links. Without lossy access links, lighttpd/Squid achieved 10x greater throghput compared to CCNx; with lossy links, CCNx outperformed lighttpd/Squid by 4x. We made a modest change to the CCNx software and found that the peak throughput was improved by 40%.

This is a first step in evaluating content distribution performance, and NDN in particular. In future work, we plan to develop a fully functional traffic generator to explore a richer distribution of content names and request patterns. Additionally, work is under way to design a scalable NDN forwarding plane using a mixture of software and hardware mechanisms to achieve substantial performance improvements.

References

[1] Stefan Saroiu et al. An analysis of internet content delivery systems. In *OSDI '02*.
[2] Ashok Anand et al. Packet caches on routers: the implications of universal redundant traffic elimination. In *SIGCOMM '08*.
[3] Ashok Anand et al. Redundancy in network traffic: findings and implications. In *SIGMETRICS '09*.
[4] Ion Stoica et al. Internet indirection infrastructure. In *SIGCOMM '02*.
[5] Teemu Koponen et al. A data-oriented (and beyond) network architecture. In *SIGCOMM '07*.
[6] Ashok Anand, Vyas Sekar, and Aditya Akella. Smartre: an architecture for coordinated network-wide redundancy elimination. In *SIGCOMM '09*.
[7] Petri Jokela et al. Lipsin: line speed publish/subscribe inter-networking. In *SIGCOMM '09*.
[8] Van Jacobson et al. Networking named content. In *CoNEXT '09*.
[9] Lixia Zhang et al. Named data networking (ndn) project. Technical Report NDN-0001, NDN, 2010.
[10] Lucian Popa, Ali Ghodsi, and Ion Stoica. Http as the narrow waist of the future internet. In *Hotnets '10*.
[11] Charlie Wiseman et al. A remotely accessible network processor-based router for network experimentation. In *ANCS '08*.
[12] Haowei Yuan and Patrick Crowley. Performance measurement of name-centric content distribution methods. In *ANCS '11*.
[13] Haowei Yuan, Tian Song, and Patrick Crowley. Scalable ndn forwarding: Concepts, issues and principles. In *ICCCN '12*.
[14] Yi Wang et al. Scalable name lookup in ndn using effective name component encoding. In *ICDCS '12*.
[15] Muscariello Luca et al. Bandwidth and storage sharing performance in information centric networking. In *ICN '11*.
[16] Shakir James and Patrick Crowley. Imp: Isp-managed p2p. In *P2P'10*.
[17] KyoungSoo Park and Vivek S. Pai. Scale and performance in the coblitz large-file distribution service. In *NSDI '06*.