Scalable NDN Forwarding: Concepts, Issues and Principles

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

Tian Song
School of Computer Science
Beijing Institute of Technology
Beijing, China 100081
Email: songtian@bit.edu.cn

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

***Abstract*—Named Data Networking (NDN) is a recently proposed general-purpose network architecture that leverages the strengths of Internet architecture while aiming to address its weaknesses. NDN names packets rather than end-hosts, and most of NDN's characteristics are a consequence of this fact. In this paper, we focus on the packet forwarding model of NDN. Each packet has a unique name which is used to make forwarding decisions in the network. NDN forwarding differs substantially from that in IP; namely, NDN forwards based on variable-length names and has a read-write data plane. Designing and evaluating a scalable NDN forwarding node architecture is a major effort within the overall NDN research agenda. In this paper, we present the concepts, issues and principles of scalable NDN forwarding plane design. The essential function of NDN forwarding plane is fast name lookup. By studying the performance of the NDN reference implementation, known as CCNx, and simplifying its forwarding structure, we identify three key issues in the design of a scalable NDN forwarding plane: 1) exact string matching with fast updates, 2) longest prefix matching for variable-length and unbounded names and 3) large-scale flow maintenance. We also present five forwarding plane design principles for achieving 1 Gbps throughput in software implementation and 10 Gbps with hardware acceleration.**

***Index Terms*—Named Data Networking, Forwarding Plane, Longest Prefix Match**

I. Introduction

Named Data Networking (NDN) [1] is a recently proposed network architecture that supports efficient content distribution intrinsically. NDN takes the content-centric approach [2], focusing on what the content is rather than where the content is. Each NDN packet has a unique name. Since there are no source and destination addresses in an NDN packet, it is forwarded based on a lookup of its name. There are two types of packets in NDN networks, *Interest* packets and *Data* packets. To fetch content, a host sends an Interest packet, which contains the name of the requested content. A Data packet, which contains both the content and its name, will be returned if the requested content is available. NDN nodes are capable of caching Data packets, and the caching component is called *Content Store (CS)*. Large files can be chunked and thus transferred and cached at granularities smaller than the sizes of files and data objects. When an Interest packet arrives at an NDN node, if the requested content is cached, the Data packet can be returned directly from the node. If the name is not in the cache, the node records the content name and the arrival interface in the *Pending Interest Table (PIT)*, and then forwards the Interest packet via an interface identified by a name lookup in the *Forwarding Information Base (FIB)*, which contains NDN route entries. Storing pending interests in this way demands that the forwarding plane for NDN support both fast table lookups and fast table insertions and updates. IP, by contrast, has a read-only data plane. (IP payloads are both read and written, of course, but they do not require lookups and are not a challenge in high-speed data planes as a result.) NDN routes are managed hierarchically, as are IP routes. However, rather than address prefixes, NDN routes are identified by name prefixes. Overall, we expect that NDN route tables will be much larger than IP route tables, because names are longer and more numerous. In fact, NDN packet names have hierarchical structures similar to HTTP URLs, and each name consists of multiple *name component*. For instance, an NDN packet name may look like ndn://wustl.edu/web/research, where wustl.edu, web, research are the name components.

An NDN software prototype, CCNx [3], has been developed by PARC. The key component of CCNx is the *ccnd* daemon, which implements the packet forwarding plane. The current CCNx program operates on several operating systems, including Windows, Mac OS, Linux and Android and runs as an overlay on IP networks. It should be noted that NDN does not require IP architecturally, although it does require it pragmatically since IP provides global connectivity. To emphasize the point, we observe that it is possible to build an IP network as an overlay on top of NDN.

A scalable forwarding plane is the key for deploying NDN in a large scale, demonstrating its real-world feasibility. In this paper, we first present the core concepts of the NDN forwarding plane, and then explain the challenges of satisfying its scalability requirements. By analyzing and simplifying CCNx data structures and overall organization, we can identify the key issues in scalable forwarding. To conclude, we discuss the principles of designing and implementing such a system, which is the focus of our ongoing work.

The NDN forwarding plane needs to support fast packet name lookup, intelligent forwarding strategies and effective

Dr. Tian Song is involved in this work when he is a visiting scholar at Washington University in St. Louis.

cache replacement policies [1]. In this paper, we focus on fast packet name lookup in particular since name-lookup rates directly impact the scalability of the forwarding plane. Our measurements show that the peak throughput of the CCNx implementation is much lower than the 1 Gbps link rate that we consider a minimum requirement. The challenges of designing a scalable forwarding plane derive primarily from variable-length names and the read/write nature of packet forwarding. As a result, efficient algorithms and data structures, as well as advanced hardware devices, are needed to accelerate NDN forwarding. Based on NDN forwarding characteristics, we present several design principles that should generally be followed.

It is important to understand the status and practical limits of the current NDN prototype design in order to guide future research in this field. We note that recent papers have considered closely related topics, such as general content-centric router design [4], cost estimates for building content centric networks [5] and theoretical analyses of the performance of content centric networking in general [6]. To date, however, there has been no discussion or investigation of how data structures and algorithms should be applied to the specific topic of name-based forwarding, as embodied in the NDN forwarding plane. In particular, there is no literature on how the current CCNx software is implemented and how it performs in detail, except for an initial latency measurement in [7]. We feel the essential problems in NDN forwarding need to be further clarified and that a formal description of these challenges will be beneficial for the broader research community.

In this paper, we make the following contributions.

1) We present an experimental evaluation of the current CCNx implementation. Our results show that CCNx needs substantial efficiency gains to achieve sustainable gigabit performance.
2) We describe the CCNx implementation details relevant to performance. This is the first such description of the major data structures and algorithms used in CCNx.
3) We provide a formal definition of the three key problems in scalable NDN forwarding plane design. We analyze and simplify the packet forwarding operations in CCNx and NDN, and identify three major problems: *exact string matching with fast updates*, *longest prefix matching for variable-length and unbounded names* and *large-scale flow maintenance*.
4) We articulate design principles for scalable NDN forwarding plane design. Each of these five principles represents a potential research direction.

The paper is organized as follows. We present the concepts of the NDN forwarding plane in section II. In section III, CCNx performance measurement and profiling results are discussed. Section IV presents the organization of NDN forwarding data structures and operations and key issues in NDN forwarding. Section V presents five design principles for achieving 1 Gbps in software and 10 Gbps in hardware. We conclude the paper in Section VI.

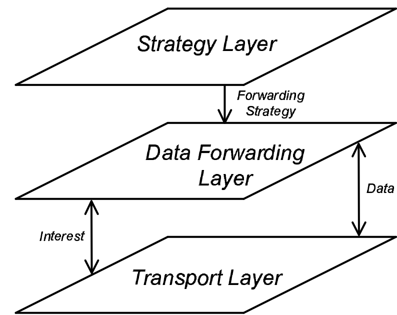

Fig. 1. Layers of the Forwarding Plane in CCNx

II. The NDN Forwarding Plane

In this section, we first describe in detail the three functions that are supported by the NDN forwarding plane, and then present the forwarding plane layers used in the current CCNx implementation. We point out that *fast name lookup* is the most critical problem for scalable NDN forwarding.

NDN forwarding plane [1] supports fast name lookup, intelligent forwarding strategies and effective caching policies. Logically, an NDN forwarding plane consists of a *Content Store*, which caches Data packets, a *Pending Interest Table*, which is used to store pending Interest requests, and a *Forwarding Information Base* that stores forwarding rules.

1) Fast name lookup. Since NDN packets do not have source and destination addresses like IP packets, an NDN packet is forwarded by performing a lookup of its content name. NDN name lookup could happen at PIT, CS and FIB, and it involves both longest prefix match and exact string match with fast updates.

2) Intelligent forwarding strategies. In NDN, it is possible for a packet to have multiple outgoing interfaces. The forwarding strategy selects the most efficient interface(s) by making forwarding decisions not only based on FIB lookup results, but also considering the network environment.

3) Effective caching policies. The Content Store is the component that makes NDN favor content distribution. Effective caching replacement policies can improve cache-hit rates and thus further improve content distribution performance.

The NDN forwarding plane decides what to do for each incoming packet. In practice, the NDN forwarding plane can be divided into multiple layers based on the functions that need to be supported. Figure 1 shows the layers CCNx used to implement its forwarding plane. The strategy layer selects forwarding strategies and impacts the forwarding decisions. Packet forwarding, pending Interest management and temporary content storing are performed in the data forwarding layer. The data forwarding layer is controlled by the strategy layer. The transport layer handles network communication and it can be viewed as an interface to format and transport data using the IP network.

A scalable NDN forwarding plane, in order to support a 10 Gbps link rate, must process tens of thousands of Interest and Data packets per second. In addition, to make caching effective, the number of cached Data packets also should be large,

which subsequently makes the Content Store name lookup more time consuming. The time spent on name lookup impacts the maximum data rate. Intelligent forwarding strategies and effective caching policies improve NDN content distribution performance but do not directly impact NDN scalability. As a result, *fast name lookup is the key problem in making NDN forwarding scalable*.

III. CCNx Performance Study

It is important to understand the status of the current NDN prototype and its practical limits. In this section, we present our preliminary performance evaluation of CCNx. Our measurement results show that the current CCNx implementation cannot meet the 1 Gbps link rate requirement. After analyzing CCNx peak throughput, we classify the problems that slow down the forwarding performance into *engineering* problems and *method* problems.

Our performance study was conducted in the Open Network Laboratory (ONL) [9]. ONL is a free and publicly-accessible network-system testbed and its easy configurability and monitoring infrastructure makes evaluating new network designs simple. The CCNx implementation evaluated in this work is ccnx-0.4.0, released on September 15, 2011. CCNx was compiled using `gcc-4.4.1` with optimization level `-O4`. The core component, *ccnd*, is configured with all default environment variable values. The Content Store size, i.e., the number of packets that can be cached on a router, impacts the overall system performance. This size is set at the default value of $50,000$, determined by the strong relationship between the CS size and the hard coded garbage collection time interval. CCNx supports both TCP and UDP, but we use only TCP in our experiments. We studied the impact of Data packet payload size on throughputs, the sizes of packets are 1, 256, 512 and 1024 bytes. To generate CCNx traffic, the built-in *ccncatchunks2* and our *ccndelphi* programs are used as the client program and server program, respectively. The *ccncatchunks2* program generates a sequence of Interest packets to fetch a large file. The generated Interest started with the name ccnx:/URI/0, where the last portion is the chunk index. It fetches the next chunk of the file by increasing the chunk index. The *ccndelphi* program generates Data packets with random payloads, and it is designed to send back Data packets as soon as possible.

CCNx peak throughput is reached when the CPU of the CCNx router is fully utilized. In general, clients send Interest packets to the server side via the CCNx router, and then the server-side machines send back corresponding Data packets via the CCNx router. To saturate the CCNx router, we use multiple clients to generate Interest requests, and multiple servers to generate Data packets. Figure 2 shows the topology for throughput measurement. Each desktop in the topology represents a dedicated single-core machine equipped with a 2.0 GHz AMD processor and 512 MB of memory. There are 16 servers on top, and 16 clients on bottom. The single core machine in the middle is a CCNx router. Network-Processor Routers (NPRs) [10], shown as circles in Figure 2, connect these machines and monitor throughput.

CCNx router throughput is set to include two values, *Outgoing Throughput*, denoted as *Out*, and *Incoming Throughput*, denoted as *In*. We differentiate outgoing and incoming throughput because they are not necessarily the same for a CCNx router, unlike in an IP network where outgoing and incoming throughputs are always very close, if not the same. We record both *Out* and *In* of the NPR port connecting to the CCNx router. These throughput values are sampled every 1 second. For each experimental configuration, we select top 20 throughput values to compute the average peak throughput and calculate a 90% confidence interval.

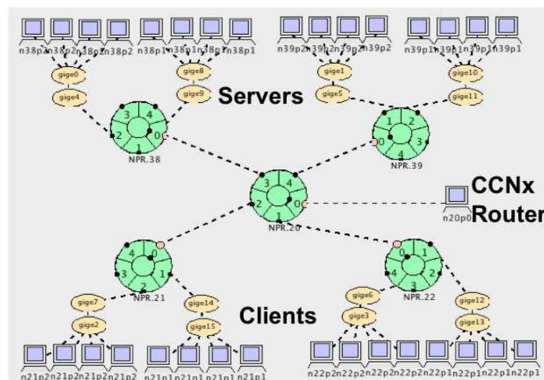

Fig. 2. Throughput Measurement Topology

Two experimental configurations were used for measuring CCNx router peak throughput. In the first case, every client requests a different file, thus the retrieved Data packets can not be shared among clients, i.e., having a Content Store does not bring any benefit. In the second case, all clients request the same file. As a result, every Data packet is shared by the clients, and the Content Store brings the highest potential benefit.

i) **No Shared Packet**. Figure 2 shows the topology for the no shared packet case. 16 client-server pairs are used for traffic generation. Every client runs the *ccncatchunks2* program to send Interest packets. Interests sent by client i have names like ccnx:/i/chunk, where i = 0...15 and the chunk starts with value 0 and increases by 1 for each generated Interest request. Interest packets sent from Client i will be routed to Server i, which runs the *ccndelphi* program to reply with Data packets. Both *ccncatchunks2* and *ccndelphi* run until they get killed. We vary the Data packet payload size, which is configurable in the *ccndelphi* program. Both the data rates (Mbps) and packet rates (packets per second) are measured, and the measured throughput results are shown in Figure 3.

According to Figure 3, the outgoing and incoming throughput are always the same, or very close in terms of data rate and packet rate. In addition, incoming packet rate is slightly higher than outgoing packet rate because when the CCNx router is saturated, some Interest packets cannot be forwarded in time

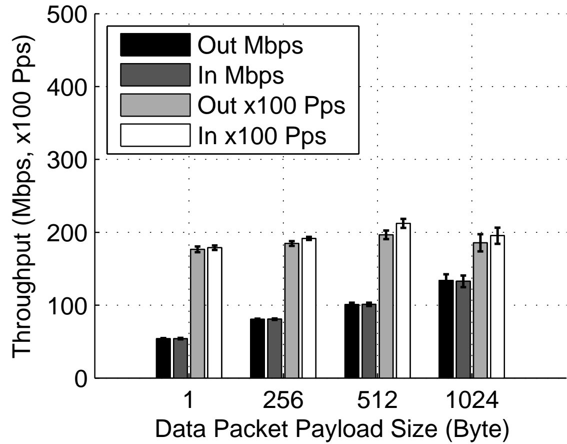

Fig. 3. No Shared Packet Case: CCNx Peak Throughput vs. Data Packet Payload Size (90% CI)

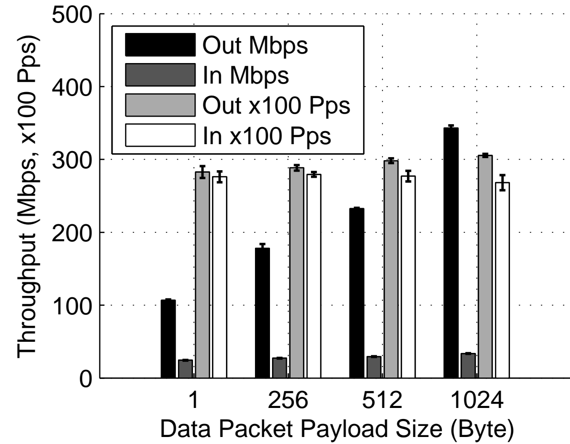

Fig. 4. All Packet Shared Case: CCNx Peak Throughput vs. Data Packet Payload Size (90% CI)

and then get dropped by the router. Data rates are the same because no Data packet is shared among clients, and thus each packet contributes to incoming and outgoing throughput only once. Figure 3 shows that as the payload size decreases, packet rate does not change much, which demonstrates that the majority of packet processing is done on the packet headers rather than on payloads. Since packet rates stay the same for different payload sizes, data rates increase when payload size increases. Note that even when the payload size is only 1 byte, there is still a high data rate because the throughput monitoring tool counts the entire packet, which includes both the packet header and payload.

ii) **All Packet Shared**. In the second setup, all of the clients send Interest packets with name ccnx:/0/chunk to the same server host. Since only a single copy of the duplicated Interest packet is forwarded by the CCNx router to the server, most of the Data packets are delivered from the Content Store to the clients. We started with 16 clients and 1 server, but the generated traffic was not enough to saturate the CCNx router. As a result, we increased the number of clients to 32 in this experiment.

Delivering a single Data packet in this case would result in 32 incoming Interest packets with 1 incoming Data packet, 1 outgoing Interest packet and 32 outgoing Data packets. Since Data packets with payloads are larger than Interest packets, the outgoing data rate should be much higher than incoming data rate, which is what Figure 4 shows. The packet rates are relatively the same across different payload sizes, although we notice that the outgoing packet rate is slightly higher than incoming packet rate. Recall that the outgoing traffic is mainly Data packets, while incoming traffic is mainly Interest packets, which implies that in CCNx Data packets are processed slightly faster than Interest packets.

Comparing Figure 3 and Figure 4, we notice that the outgoing data rate in the second case is much higher. This demonstrates that serving content directly from Content Store is more efficient and improves system throughput. It is important to note that in both cases the peak throughputs of the current CCNx implementation, 150 Mbps for the first case and 350 Mbps for the second case, are far from the 1 Gbps link rate requirement. In addition, the peak performance studied here can hardly be achieved in the real world. When the number of requests is increased, with longer packet names being used and more complicated forwarding strategy being deployed, the system throughput will degrade. As a result, identifying the bottleneck of the CCNx forwarding plane and improving its performance are critical.

We used *Gprof* [11] to profile the saturated $ccnd$ daemon. We profiled the daemon 3 times and list the top 10 most time-consuming functions in Table I. Unexpectedly, more than 60% of the time was taken by functions related to packet name decoding, and these functions are shown in italic in Table I. In particular, the *ccn_skeleton_decode* function, which is the lowest level packet decoding function, takes 35.46% of the entire program running time. Upon further investigation, it turns out that the original CCNx implementation chooses to store content in encoded format. Furthermore, the *Content Skip List (CSL)*, which is an index for the CS, does not store decoded Data packet names by itself. As a result, when the CSL is queried, an average of $logn$ content needs to be decoded on the fly to get their names, where n is the number of content entries stored in CS. In this experiment, n is always greater than $50,000$.

Based on the above analysis, two sets of problems slow down the CCNx software. The first can be categorized as *engineering* problems, such as the name decoding scheme. The other set are *method* problems. For instance, functions related to name lookup, such as the *hashtb_seek* function listed in Table I. Although engineering problems have a great impact, we recognize that academic research should focus more on the method problems, which are the main concern in this paper. Engineering problems are strongly related to design decisions, while solving method problems is fundamental to achieving fast name lookup.

TABLE I
TOP 10 TIME-CONSUMING FUNCTIONS

Function Name	Percentage(%)
ccn_skeleton_decode	35.46
ccn_compare_names	12.53
ccn_buf_match_dtag	8.95
ccn_buf_advance	6.64
ccn_buf_match_blob	5.07
ccn_buf_decoder_start_at_components	4.94
ccn_buf_match_some_blob	3.32
content_skiplist_findbefore	3.06
ccn_buf_check_close	2.34
hashtb_seek	1.36
Other functions	15.7
Sum	100

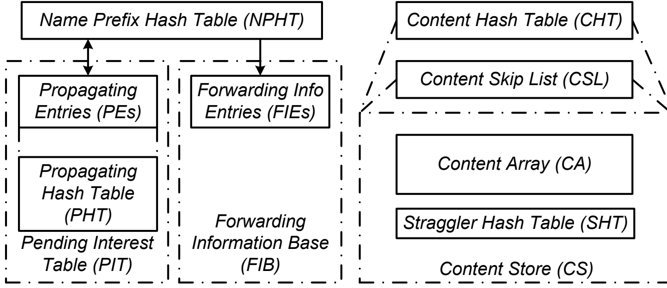

Fig. 5. CCNx Data Structures

IV. ISSUES IN SCALABLE NDN FORWARDING

In this section, we discuss the critical issues in scalable NDN forwarding plane design. We first present the data structures and operational flows employed by CCNx to implement NDN forwarding functions. To the best of our knowledge, this is the first time CCNx implementation details are discussed in the literature. We simplified CCNx data structures and operational flows so that a simpler design is presented. By analyzing the operational flows in the simplified design and comparing it with IP forwarding, we identify three most critical issues in scalable NDN forwarding, and we present their formal description.

A. Data Structures and Operational Flows in CCNx

1) Data Structures: Recall that each NDN forwarding node has three logical components, namely Content Store, Pending Interest Table and Forwarding Information Base. Figure 5 shows the data structures that implement these three components in CCNx.

The logical FIB and PIT share a hash table named *Name Prefix Hash Table (NPHT)*, which indexes the *Propagating Entries(PEs)* and *Forwarding Info Entries (FIEs)*. Each bucket in the NPHT has pointers pointing to PEs and FIEs, where PEs and FIEs are the structures storing detailed pending Interest information and forwarding information, respectively. The *Propagating Hash Table (PHT)* is keyed by the *nonce* field, which is unique for each Interest packet. PHT stores all the nonce field of the Interest packets presented in PIT (in the form of PEs). PHT prevents loops in the network, which will be explained later in the CCNx operational flows.

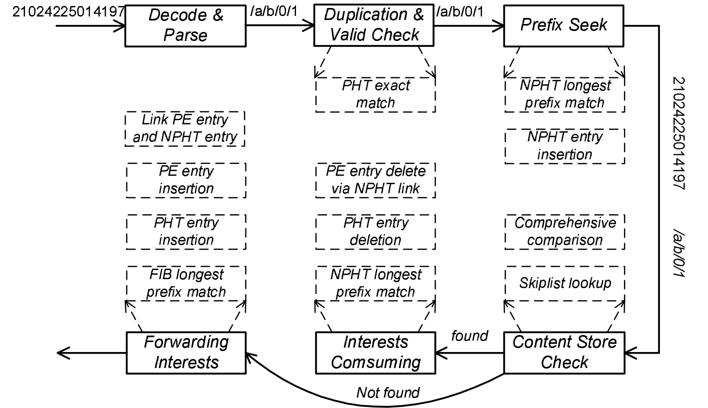

Fig. 6. Operational Flow of Processing Interest Packets

For Content Store, each Data packet is assigned a unique *accession number* when it enters the daemon, and the accession number increases by one for each incoming Data packet. The cached Data packets are stored in an array named *Content Array (CA)* indexed by the accession numbers. Since the number of the slots in CA is fixed, there is a range of accession numbers that CA supports. At the very beginning, it is from 0 to the size of the array. As time passes, old cached content gets kicked out of the CS, and the starting and end accession numbers of CA increase, which is similar to a sliding window. However, there might be some old but popular Data packet whose accession number is out of the range that CA supports. These packets are stored in the *Straggler Hash Table (SHT)* to save space; otherwise, the Content Array would need to support a larger range of accession numbers. Two data structures summarize the Content Store, namely *Content Hash Table (CHT)* and *Content Skip List (CSL)*. The CHT is a hash table keyed by the Data packet full name. CSL is a standard implementation of the skip list [8] data structure. CSL is chosen because it supports content-order lookup [2].

2) Operational Flows: Interest packets and Data packets are processed differently in NDN. CCNx implements two operational flows in the `ccn_process_incoming_interest` and `ccn_process_incoming_content` functions.

i) **Interest Packets.** Figure 6 shows the operational flow of processing Interest packets in the CCNx implementation. When an Interest packet arrives, it is decoded from its binary wire format and then parsed. The fields of the Interest packets, such as packet name and other optional fields, are stored in an internal structure. For instance, the name /a/b/0/1 shown in Figure 6 is the parsed Interest packet name. The optional fields are not presented in this figure as they do not affect the core packet forwarding processing. After that, an exact string matching is performed on the PHT to ensure this is not a packet forwarded by this node. The lookup key is the nonce field of the Interest packet. If there is a match, this packet has indeed been forwarded by this node before, and it is still waiting for the corresponding Data packet to return. As a result, this Interest packet will be discarded.

If this is a new Interest packet, CCNx checks NPHT to make

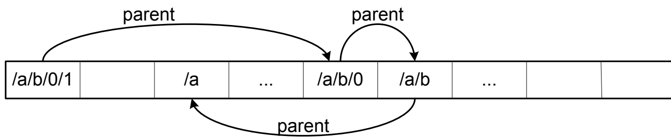

Fig. 7. NPHT After Processing /a/b/0/1

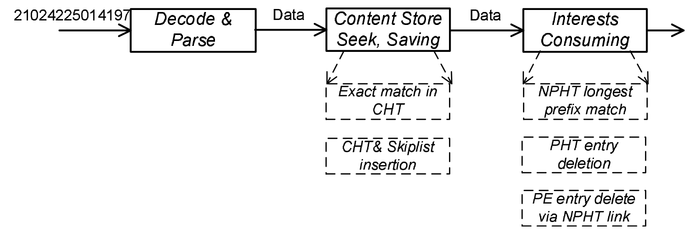

Fig. 8. Operational Flow for Handling Data Packets

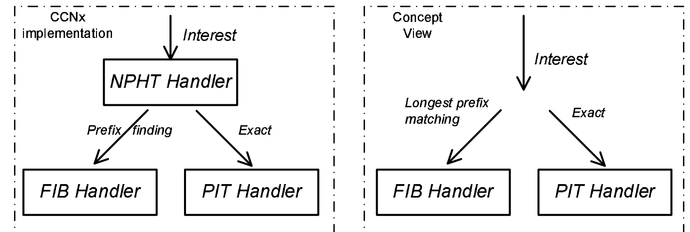

Fig. 9. CCNX Implementation vs. Concept View of NDN Forwarding Plane

sure all prefixes of this Interest packet exist in NPHT. We call this stage *Prefix Seek*. New entires are inserted if there are prefixes not stored in the NPHT yet. Figure 7 shows how the NPHT looks like after the name /a/b/0/1 is processed. Prefixes are linked by *parent* pointers for fast name lookup. After the Prefix Seek stage, the packet name is sent to the Content Store to check whether a cached content can satisfy this Interest or not. In Content Store, Content Skip List is queried to find potential matching content following standard skip-list lookup procedure. If a potential match is found, the content is pulled from the CS, and an comprehensive match is performed to check whether the content indeed satisfies the Interest. This checking is needed because there are other restrictions in the Interest packet that do not get represented by packet names, such as using the exclusion filters to exclude certain content [1].

If this Data packet is a satisfied match, the Interest packet will be consumed. Currently, CCNx checks all the prefixes of the content name to consume as many Interests as possible. In this case, if the name of the Data packet is /a/b/0/1, the prefixes /a/b/0, /a/b and /a will all be examined. CCNx checks all the prefixes because the content /a/b/0/1 may also satisfy /a/b/0 if Interest /a/b/0 makes clear that only the first three name components are required to be matched. As a result, a single Data packet may consume many Interest packets, but it also takes more processing time. We think performing all prefix checks in the NDN forwarding plane is an open question. If all prefix check is not a required feature, then Interest consuming can be reduced to an exact string matching problem, as we will show in the simplified operational flow design. After finding all the Interest packets that can be consumed, this Data packet is sent back to these Interests senders, and the consumed Interest names are deleted from the NPHT and PHT.

If no Data packet stored in the CS can satisfy this Interest, the FIB is consulted to find the proper next-hop information for this packet. CCNx achieves this by performing a longest prefix finding on the NPHT. It first checks whether there is a valid FIE for the longest prefix, i.e., the full packet name. Then the rest of the prefixes can be retrieved in length-decreasing order by following the parent pointers. The process stops when a valid PE is found or the root prefix is reached. When this packet is forwarded, a Propagating Entry containing the entire Interest message is constructed and then linked to the NPHT bucket that stores the full name of this packet. Then the nonce field of this Interest packet is inserted into the PHT.

ii) **Data Packets.** The operational flow of processing Data packets in the CCNx implementation is shown in Figure 8. When a Data packet arrives, just as with Interest packets, this Data packet is decoded and parsed first. After that, an exact string matching on CHT is performed to make sure that this packet has not been stored in the CS before, and that the lookup key is the full name of the Data packet. If this packet is not duplicated, its name is inserted to the CHT and CSL, and the content is stored in CS. Then this Data packet will try to consume as many Interests as possible, as we have just described.

B. Simplified Data Structures and Operational Flows

CCNx is a prototype NDN forwarding node, and some of its implementation decisions are not required by NDN. Figure 9 shows the CCNx implementation model and a conceptual view of the NDN forwarding plane. As Figure 9 shows, NPHT is used as an intermediate data structure to store prefixes for FIB and PIT in CCNx. As a result, the longest prefix match on FIB is converted to an exact string match in the NPHT. Sharing the NPHT between FIB and PIT is a software optimization since the memory utilization is reduced. It should be noted that supporting longest prefix matching in FIB and exact string matching in PIT would be able to realize core NDN forwarding functions.

As a result, the operational flows can be simplified as shown in Figure 10 for Interest packets and Figure 11 for

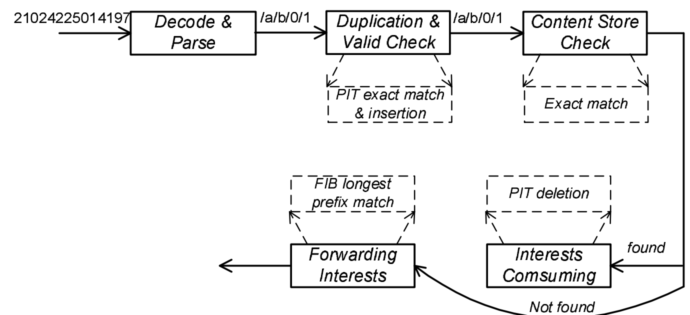

Fig. 10. Simplified Operational Flow for Handling Interest Packets

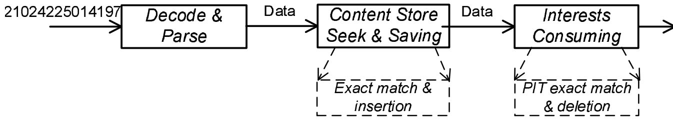

Fig. 11. Simplified Operational Flow for Handling Data Packets

TABLE II
PACKET FORWARDING IN IP AND NDN

	IP	**NDN**
Forwarding Key	IP address	Content name
Key length	32 bits	**Variable**
Forwarding rules	Longest prefix match	Longest prefix match
Per-packet READ	Yes	Yes
Per-packet WRITE	No	**Yes**

Data packets, respectively. For each incoming Interest, the forwarding plane first checks duplication and prevents loop using exact string matching in PIT, and then find possible contents from the Content Store. If no content is found, a longest prefix match is performed on FIB to locate forwarding information and then the Interest is forwarded. If content is found, the Interest is consumed by deleting the entry in PIT. Basically, the conceptual flow is the same as the one in CCNx shown in Figure 6, except that we remove the NPHT data structure. Each incoming Data packet is stored in CS by performing an exact match and, the corresponding Interest is consumed. It is easier to understand the operational flow when NPHT is removed.

The simplified operational flows cover the core NDN forwarding functions. Based on this simplified operational flows, efficient data structures and algorithms can be designed to optimize the forwarding performance. In particular, simplicity and less interactive flows are important for pipelined-based architectures in hardware design.

C. Key Issues to Be Solved

Comparing with IP forwarding, NDN forwarding is complicated and more challenging. Table II lists the differences between IP packet forwarding and NDN packet forwarding. Our ultimate goal is to achieve a 1 Gbps forwarding rate in software implementation and 10 Gbps with hardware acceleration. To achieve that, we identify three critical issues to be solved in NDN forwarding. We present these three issues in the order of importance.

1) Exact string matching with fast updates: In exact string matching with fast updates, there is a lookup key k and a set of strings stored in *Set*. The problem is to verify whether k is in *Set*, and then perform operations such as insert, delete or update values in *Set*. In the NDN forwarding plane, exact string matching with fast updates is performed in PIT lookups or Content Store lookups. A PIT lookup inserts a new Interest packet or updates an existing Interest packet, or deletes a consumed Interest packet. For a Data packet, a Content Store lookup results in inserting this Data packet into the CS or updating the expiration time of a stored Data packet. In the worst case, every packet requires an update. This problem can be treated as membership verification and anchored multi-string matching.

Although high-speed exact string matching has been heavily studied in the fields of network intrusion detection [12] [13] and content filtering [14] [15], existing methods can not be readily applied in NDN. To the best of our knowledge, none of the aforementioned studies can support updating on a per-packet basis.

2) Longest prefix matching for variable-length and unbounded names: In longest prefix matching (LPM), there is also a lookup key k and a set of strings *Set*. The problem is to find a string s from *Set* such that s is the longest prefix of k among all the strings in *Set*. Longest prefix match is performed in FIB lookups. Since generally the forwarding rules are not updated very fast, the implementation of LPM should mainly focus on fast lookup on variable-length and unbounded names.

LPM has been heavily studied for IP lookup and URL filtering [16] [17] [18]. Although NDN longest prefix match is the same as IP longest prefix match in principle, existing methods for IP lookup do not support NDN well. NDN packets have variable-length and unbounded names, unlike IP packets, which have fixed 32-bit addresses as the lookup keys. Most of the proposed IP LPM solutions are slower or require larger memory storage for longer lookup keys.

3) Large-scale flow maintenance: Flow maintenance is another way of implementing the Pending Interest Table. We identify large-scale flow maintenance as an independent problem since in hardware solutions, it is not efficient to allocate PIT entries dynamically or using pointers in the design. Alternatively, the hardware solutions can choose to maintain the pairs {name, incoming interfaceID, outgoing interfaceID} in the forwarding plane. We name this pair *ndn-status*. Flow maintenance is similar to IP network per-flow monitoring. However, NDN packets have variable-length names to process, not fixed length five-tuples. In addition, since the number of content names is unlimited, the flow table can be extremely large, which also makes its maintenance challenging.

V. DESIGN PRINCIPLES

For large-scale NDN deployments, our minimum acceptable targets are to achieve 1 Gbps forwarding performance with a software implementation and 10 Gbps with hardware. To solve the above issues efficiently, we offer design principles and identify potential opportunities for optimization.

A. Aim for Constant-Time Operations

The goal of fast name lookup is to achieve constant lookup time for variable-length names. Since names have unbounded length, this goal may be, in principle, unachievable. In practice, however, most names are likely to fit within one Interest packet.

In NDN forwarding, a longer name requires greater processing time. We measured the impact of the number of components in packet names on CCNx router performance and verified this behavior experimentally. The experimental setup was the same as in Section III, with the system topology

shown in Figure 2. We set the Data packet payload size to 1024 bytes. This experiment differed from the first case of Section III in that the requested names were longer. Client i sends out Interest packets with names ccnx:/i/i.., where i = 0...15 and the number of the name components is varied in each experiment configuration. In this way, we vary the number of components in packet names; the measured performance results are shown in Figure 12.

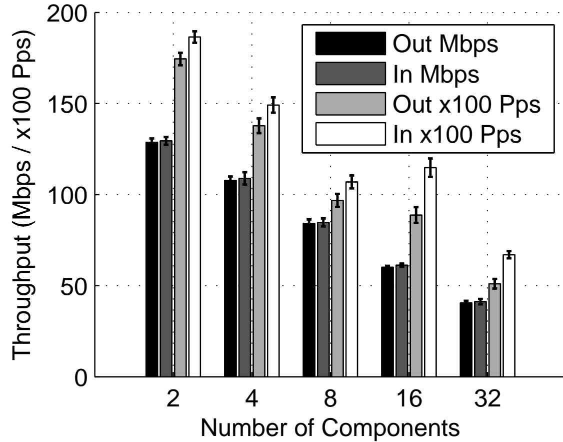

Fig. 12. CCNx Peak Throughput Performance vs. Number of Name Components (90% CI)

Figure 12 shows that CCNx router throughput, in terms of both data rate and packet rate, decreases as the number of name components increases, which is what we expect. In particular, when the number of name components reaches 32, the data rates are close to 40 Mbps, which is far below the 1Gbps packet processing goal.

To design (near) constant name lookup mechanisms, one possible direction is to map longer names to shorter names internally at each NDN node.

B. URL-format for Optimization

When designing algorithms, we can recognize that names in NDN have a format similar to HTTP URLs. Our second principle is to exploit the URL format for optimization. Two features of URL format can be exploited. The first feature is the component-based string matching. Exact and longest prefix matching should take components as the matching units, rather than individual characters. The second feature is to exploit URL characteristics. For instance, the distribution of the number of name components in URLs in real-world networks can be used to guide the design of heuristics that achieve constant-time performance when the expected distributions are observed in NDN traffic. It is mentioned in [5] that most URLs have fewer than 30 name components. With such guidance, we could design lookup structures capable of achieving constant time lookup latency for names with 30 or fewer components.

C. Simple Data Structures for Fast Updates

Propagated Interest packets need to be inserted into the PIT so that when the corresponding Data packet returns, the Data packet can be successfully delivered to the interface that requested it. After this, the record will be deleted from the PIT. In the worst case, updates need to be done on each packet's arrival. Updates in data structures have been studied before but never (so far as we know) studied at the frame arrival rates in the data plane. We feel that using simple data structures to implement components that require fast update is the only path to high performance, since there will be little time between arrivals to manipulate complicated, inter-connected data structures. These data structures include hash tables, d-left hash tables [19] and counting bloom filters [20]. Tree- or Trie- like data structures usually need readjusting in order to balance the data structure following an edit, and thus may not be as efficient as hash-based solutions.

D. Efficient Packet Encoding/Decoding

According to our CCNx profiling results, a significant portion of packet processing time is spent on packet decoding, which is unusual for high performance router design, where string lookup is the bottleneck. CCNx employs a complicated XML format to encode packets, which is good for network architecture and protocol design as it is very flexible, but it slows down the entire data plane. There are two ways to solve this problem. The first way, which we think is very challenging, is to develop a packet decoding algorithm that can decode XML encoded packets quickly and efficiently. A simpler way is to define a new packet format that favors fast packet encoding and decoding. For instance, a simple and efficient packet format could be similar to IP packets, where each field of the packet has fixed length. The fields can store offsets or other information of the packet, such as packet flags, etc..

E. Different Content Store Policies

The Content Store not only enables Data packet caching and redistribution, it also supports features beyond matching exact content. For example, an Interest packet can carry additional flags to tell a CCNx router to respond with the latest version of content without requesting a newer one from the producer. The latest version is determined by the default ASCII order in the name suffix. For example, suppose that one node requests ccnx:/a/b/c as an Interest, including the "latest" flag. Suppose further that the CS has content with the names of ccnx:/a/b/c/0 and ccnx:/a/b/c/1, which have prefix of /a/b/c (here 0 and 1 are version IDs, a common convention amongst CCNx applications). Then the CS will respond with ccnx:/a/b/c/1 as the latest version. These complicated features go far beyond exact matching and longest prefix matching. As a result, our fifth principle is to handle this feature differently according to the placement of routers and their corresponding performance requirement. For edge routers with expected performance in the range of 1Gbps, this feature should be supported. For core routers or higher performance nodes with 10 Gbps as a target rate, we do not suggest supporting this feature. Rather, we think the goal should be to maintain simplicity and limit CS matching to exact matching with hardware acceleration.

VI. Conclusion and Future Work

In this paper, we describe the NDN forwarding plane and analyze its implementation in CCNx. To the best of our knowledge, this is the first description and analysis of CCNx implementation details, including both data structures and organization.

We have described a direction that we believe will lead to data structures and a data plane organization that will achieve scalable performance. After simplifying the data plane organization with respect to CCNx, we identify three key issues in NDN forwarding and present five principles to guide the design of a scalable NDN forwarding plane. Our current and future work focus on detailed solutions to the three key issues, which are challenging for next-generation networking.

Acknowledgment

The authors wish to thank Pierluigi Rolando for writing the original version of the *ccndelphi* software when he was a visiting scholar at Washington University in St. Louis. The authors also thank John Dehart and Jyoti Parwatikar for their support on the Open Network Laboratory. This work has been supported by National Science Foundation Grant CNS-1040643.

References

[1] The NDN project team. Named Data Networking (NDN) Project. In PARC Technical Report NDN-0001, October 2010.
[2] Van Jacobson et al.; Networking Named Content, In Proc. of CoNEXT 2009, Rome, December, 2009.
[3] CCNx: http://www.ccnx.org/.
[4] Somaya Arianfar; Pekka Nikander; Jorg Ott; On Content-Centric Router Design and Implications. In Proc. of ACM ReArch 2010, November 30, 2010.
[5] Diego Perino; Matteo Varvello; A Reality Check for Content Centric Networking. In Proc. of ACM ICN 1011, August 19, 2011, pp: 44-49.
[6] Giovanna Carofiglio; Massimo Gallo; Luca Muscariello; Bandwidth and Storage Sharing Performance in Information Centric Networking. In Proc. of ACM ICN 2011, August 19, 2011, pp: 26-31.
[7] Jiachen Chen; Mayutan Arumaithurai; Lei Jiao; Xiaoming Fu; K.K.Ramakrishnan; COPSS: An Efficient Content Oriented Publish/Subscribe System. In Proc. of ANCS 2011, Brooklyn, New York.
[8] William Pugh; Skip Lists: A Probabilistic Alternative to Balanced Trees; Commun. ACM, Vol 33, Issue 6, June, 1990, pp: 668-676.
[9] John DeHart et al.; The Open Network Laboratory; In Proc. of the 37th SIGCSE Technical Symposium on Computer Science Education, 2006.
[10] Charlie Wiseman et al.; A Remotely Accessible Network Processor-Based Router for Network Experimentation; In Proc. of 4th ACM/IEEE ANCS, 2008.
[11] Graham, Susan L. and Kessler, Peter B. and McKusick, Marshall K.; Gprof: A Call Graph Execution Profiler; SIGPLAN, April, 2004.
[12] Nan Hua; Haoyu Song; Lakshman, T.V.; Variable-Stride Multi-Pattern Matching For Scalable Deep Packet Inspection; In Proc. of IEEE INFOCOM 2009, April 19-25 2009, pp: 415-423.
[13] Tian Song; Wei Zhang; Dongsheng Wang; Yibo Xue; A memory efficient multiple pattern matching architecture for network security. In Proc. of IEEE INFOCOM 2008, April 13-18 2008, pp: 166-170.
[14] Derek Pao; Xing Wang; Xiaoran Wang; Cong Cao; Yuesheng Zhu; String Searching Engine for Virus Scanning; In IEEE Transactions on Computers, Nov. 2011, Volume: 60 Issue:11, pp: 1596-1609.
[15] Sailesh Kumar; Patrick Crowley; Segmented Hash: An Efficient Hash Table Implementation for High-Performance Networking Subsystems; In Proceedings of the 2005 ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS). Princeton, N.J. October, 2005.
[16] Yi-Hsuan Feng; Nen-Fu Huang; Chia-Hsiang Chen; An Efficient Caching Mechanism for Network-Based URL Filtering by Multi-Level Counting Bloom Filters. In Proc. of IEEE International Conference on Communications (ICC), 5-9 June 2011.
[17] Zhou Zhou; Tian Song; Yunde Jia; A High-Performance URL Lookup Engine for URL Filtering Systems. In Proc. of IEEE International Conference on Communications (ICC), 23-28 May 2010.
[18] Haowei Yuan; Benjamin Wun; Patrick Crowley; Software-based implementations of updateable data structures for high-speed URL matching. In Proc. of the 6th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (ANCS '10), October 2010.
[19] Andrei Broder; Michael Mitzenmacher; Using multiple hash functions to improve IP lookups; In Proc. of IEEE INFOCOM 2001, April 22-26 2001, pp: 1454-1463.
[20] Andrei Broder; Michael Mitzenmacher; Network Applications of Bloom Filters: A Survey; Internet Mathematics 1 (4) 2005. pp: 485-509.