March 25, 2014
PARC

CCNx 1.0 Implications for Router Design

Abstract
This paper examines what is required to create a CCN router that is efficient, intelligent, and responsive to the growing concerns of scale, security, and latency observed on the Internet today. First, we discuss the requirements that CCN imposes on a router design in terms of specific resources and constraints. Next, we review the PARC-OS software architecture with its Linux virtual machines and distributed forwarding technology. The paper then describes the current research hardware platform, a pair of terabit-class routers and a virtualized software environment that offer significant local processing and storage resources for experimentation. The final section on Implementation Notes presents considerations for CCN on wire-speed hardware. We conclude with a very brief overview of next steps.

Keywords
Content Centric Networks – Named Data Networks

Point of Contact: Dick Sillman <dick.sillman@parc.com>

Contents

Introduction

The functional goal of the Internet Protocol as conceived and created in the 1960s and 1970s was to enable two machines, one comprising resources and the other desiring access to those resources, to have a conversation with each other. The operating principle was to assign addresses to end-points, thereby enabling these end-points to locate and connect with one another. Since those early days, there have been fundamental changes in the way the Internet is used — from the proliferation of social networking services to viewing and sharing digital content such as videos, photographs, documents, etc. Instead of providing basic computer-to-computer connectivity, the Internet has become largely a distribution network with massive amounts of video and web page content flowing from content providers to viewers. Internet users of today are demanding faster, more efficient, and more secure access to content without being concerned with where that content might be located.

To address the Internet's modern-day requirements with a better fitting model, PARC has created a new networking architecture called Content-Centric Networking (CCN) that is a strict superset of the original communication architecture. CCN operates by addressing and delivering content objects directly by name instead of merely addressing network end-points. In addition, the CCN security model explicitly secures individual content objects rather than securing the connection or "pipe." Named and secured content resides in distributed caches populated automatically on demand. When requested by name, CCN delivers named content to the user from the nearest cache, thereby traversing fewer network hops, eliminating redundant requests, and consuming less resources overall.

As part of the CCN program, PARC has developed a research prototype of the CCN protocol, APIs, and sample applications that have been made available as open-source software since late 2009. This software runs as an overlay on

IP and has been used by over 150 institutions across the world during the past four years. We've used the collective community experience from the wide deployment of our software to inform a new iteration of the CCN protocol and stack. Our next goal is to build a CCN router to serve as a "clean slate" research platform that partners can use for experimentation and development of their next-generation products.

This paper examines what is required to create a CCN router that is efficient, intelligent, and responsive to the growing concerns of scale, security, and latency observed on the Internet today. First, we discuss the requirements that CCN imposes on a router design in terms of specific resources and constraints. Next, we review the PARC-OS software architecture with its virtual machine and distributed forwarding technology. The paper then describes the current research hardware platform, a pair of terabit-class routers and a virtualized software environment that offer significant local processing and storage resources for experimentation. The final section on Implementation Notes presents considerations for CCN on wire-speed hardware. We conclude with a very brief overview of next steps.

1. CCN Requirements

This section reviews the platform requirements for running the CCNx core protocols. The section Forwarding Plane discusses how a forwarder handles Interest message, which leave state at each node and follow the FIB, and Content Object messages, which consume the reverse path state at each node. It also discusses issues around fast-path caching, called the Content Store, and message fragmentation due to path MTU. The section Control Plane reviews the type of control functionality necessary for a CCNx router. It focuses on PIT, FIB, and Content Store management. The section Management Plane describes CCN-specific management tasks. These include using CCN to manage CCN, key distribution, and cache control.

1.1 Forwarding Plane

This section describes the CCNx 1.0 protocols and outlines their implementation. The following section describes modifications to the implementation for CCN native packet fragmentation, such as running directly over Ethernet. The next section developes equations to estimate the sizes of various tables, such as the PIT and FIB and gives results for a few scenarios. We then describe two optimizations: Express Headers and Autonomous System Switching. These technologies greatly simplify system design and resource usage. Finally, we summarize the main results in a table.

The core protocol specifies that an Interest name must exactly match a Content Object Name. In addition, if the Interest has a KeyId restriction, that must be numerically equal to the Content Object's publisher KeyId; it does not imply a cryptographic operation on the fast path. If the interest has a Content Object Hash restriction, then the forwarder must verify that the SHA-256 hash of the content object (minus some per-hop) equals the value in the Interest. This is a strong restriction that involves a cryptographic hash. When responding from the Content Store, a system must verify an Object's signature if the Interest has a KeyId restriction to prevent cache poisoning attacks.

A forwarder has three notional tables: Forwarding Information Base (FIB), Pending Interest Table (PIT), and Content Store (CS). An implementation may use a different information organization so long as the external behavior is the same. The FIB is the routing table. It is a longest-matching-prefix name table populated by a routing protocol or with static routes. The PIT records Interest state such that a Content Object may follow the reverse Interest path. It also serves for Interest aggregation, so multiple similar interests do not get forwarded upstream. The Content Store is the in-network Content Object cache. The Content Store is optional.

Definition 1.1. (Satisfy Interest) *A Content Object satisfies an Interest if and only if (a) the Content Object name exactly matches the Interest name, and (b) the Content Object KeyId equals the Interest KeyId restriction, if given, and (c) the computed Content Object Hash equals the Interest Content Object hash restriction, if given.*

As a Content Object moves through the "fast path" of the network, it matches pending interests at each hop according to the rules in Definition 1.1. Only end systems verify cryptographic signatures. Each hop may need to compute the Content Object Hash, if the pending interest includes the Content Object Hash restriction.

The network honors only one security guarantee along the fast path. If a user requests content by name and Content Object Hash, then it will delivery exactly that. Of course, if a system is attached to a misbehaving network, it may deliver incorrect content, so the user must still always verify. If a user requests content by name and KeyId, there is no cryptographic assurance that the correct publisher responded – we rely on routing to deliver Interests to an authoritative publisher. There may, of course, be misconfigured systems or transitory attacks. A user's only recourse is to discard an invalid response (content for which the key did not verify the signature) and re-ask the same question. Because the attacks are transitory, a persistent requester will eventually get the correct content.

A CCN forwarder may aggregate two pending Interests if they have the same constraints, based on Definition 1.1. For example, if two Interests have the same name and KeyId restrictions, a system only needs to forward one of them. On the reverse path, a single Content Object may satisfy multiple non-similar Interests. For example, if Interest #1 asks for /foo/bar and Interest #2 asks for /foo/bar with KeyId 5, then a response Content Object with the name /foo/bar and KeyId 5 would satisfy both, even through they are not aggregated.

A CCN forwarder may aggregate two Interests with different lifetimes. If the remaining lifetime of the first Interest is longer than the second, it aggregates the second as normal, but only replies down that path if the response Content Object arrives during its lifetime. If the second Interest has a longer lifetime, the system may choose to (a) forward the second Interest, treating it as un-aggregated, or (b) aggregate it and forward it only if no response arrives within the first Interest's lifetime.

The Content Store has more restrictive matching rules. The reason for more restrictive rules is that the content store persists over time, so if incorrect content gets in to the cache, it may cause persistent failures unless strong rules govern responses.

Definition 1.2. (Content Store) *If an Interest has a KeyId restriction, then the Content Store must verify a cached object's signature before returning it to the requestor. This means that either the Interest must carry the desired public key or the Content Object itself must carry its own public key. If an interest has a Content Object Hash restriction, then the Content Store must verify a content object's hash prior to returning it to the requestor.*

A forwarder must implement a forwarding strategy. We require – at minimum – two strategies. The forwarding strategy is determined by the FIB entry for a prefix. The first strategy is Single Path and the second strategy is All Path. In general, an Interest should not be forwarded back out its ingress port. The exceptions are non-broadcast multiple access networks, or Mobile Adhoc networks with hidden terminals. The exceptional cases require specific routing protocols and network headers beyond the scope of the present document.

Definition 1.3. (Single Path Strategy) *A forwarder will send an Interest along a single forwarding path. It may alternate the single path between many alternatives, but a single Interest will only go one way.*

Definition 1.4. (All Path Strategy) *A forwarder will send an Interest out all eligible interfaces, like a Multicast service.*

The original CCN strategy is a foraging strategy. The foraging strategy is an optional strategy. It periodically probes the alternative paths for a FIB entry to discover the best path without requiring the routing protocol to carry quality metrics.

Definition 1.5. (Foraging Strategy) *For each FIB prefix, the forwarder tracks the average response time per eligible egress port. An Interest will be sent out the best port, and sets a response timer based on that interfaces average response time. If no answer comes within the response timer, the forwarder tries the next-best interface and sets a response timer based on its average response time. If no answer comes, the Interest is forwarded out all remaining eligible interfaces. For each Interest, the response timer for the best interface is decremented from the best value to force periodic probing of alternate paths.*

Forwarding loops should be prevented by a routing protocol, both for unicast (Single Path) and multicast (All Path) forwarding strategies. Due to configuration errors or transient loops, some Interests may cycle. CCNx 1.0 uses a Hop Limit field in the Interest header to prevent loops. It must be decremented each time a forwarder transmits an Interest out an external interface. Some implementations may use an optional Nonce field for faster detection of duplicates and loops. If a system supports a nonce, it should discard any Interest that arrives carrying the same nonce as has been seen recently. The length of "recently" is up to the implementer, but should be significantly longer than the longest network cycle time.

A Content Object cannot cycle on its own, because it consumes the PIT state used to forward it. A Content Object can only cycle if an Interest cycles, therefore preventing Interest cycles prevents Content Object cycles.

The CCNx 1.0 node behavior, at the CCNx message level, is as follows. A later section describes working with fragmented messages.

1. Receive Interest
   (a) If the interest may be satisfied from the Content Store, return that object to the previous hop, then discard the Interest.
   (b) Add or aggregate in PIT. If aggregated, the system must implement one of the above strategies for Interest lifetimes.
   (c) Lookup the Interest name in the FIB, and forward it out all interfaces, excluding the ingress interface and those interfaces previously used if the PIT entry was already pending.
   (d) If the new Interest extends the lifetime of the PIT entry, then the forwarder should re-express the interest after the current lifetime expires and adjust the Interest's lifetime to the difference.
   (e) If a PIT entry's lifetime expires without being satisfied, the PIT entry is silently discarded.
   (f) If the Interest has a Scope "0" it is only forwarded to "local" applications.
   (g) If the Interest has a Scope "1" it is forwarded to "local" applications, and if it arrived from a "local" interface, it is forwarded out "remote" interfaces.
   (h) If the Interest has a hop count per-hop header, then that header field is decremented on reception and it is only forwarded to "remote" interfaces if the difference is greater than zero.
2. Receive Content Object
   (a) A node looks in the PIT to find all PIT entries satisfied by the Interest (see Definition 1.1). If one or more of the PIT entries has a Content Object Hash restriction, the forwarder must verify the cryptographic hash matches the PIT entry, otherwise it

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|          T_INTFRAG            |             LENGTH            |
+---------------+---------------+---------------+---------------+
|                      Fragment Stream ID                       |
|                                                               |
+---------------+---------------+---------------+---------------+
|       Path Minimum MTU        | X X | FragCnt | X X | FragNum |
+---------------+---------------+---------------+---------------+
```

Figure 1. Interest Fragment Header

does not satisfy those entries. If the forwarder finds one or more satisfied PIT entries, it forwards the Content Object out the interfaces denoted as ingress interfaces of the PIT entries, then removes those satisfied PIT entries.

(b) A Content Object that does not match any PIT entries should be dropped.

(c) A forwarder should verify that a Content Object arrives from an expected previous hop. If the previous hop is not in the FIB forwarding path for the name, it is likely the Content Object is an off-path injection attack, and it should be dropped.

(d) If the node has a Content Store, it may save the Content Object in the Content Store. See the caching section about Content Store cache directives.

3. Expire Pending Interest

   (a) A PIT entry should carry an Expiry time. A Content Object should not be forwarded along a reverse path of an expired PIT entry. An implementation may chose to use timers or lazy expiration of PIT entries, such as by using an Expiry field. If the PIT hash table does not shrink, old PIT entries do not need to be freed, they will just be overwritten when accessed.

Forwarding Fragmented Messages

If CCNx messages are fragmented, then the above forwarding rules need to include a fragment matching mechanisms. See the document "CCNx End-To-End Fragmentation" for details on the fragmentation protocol.

Figure 1 shows the header added to an Interest message for end-to-end fragmentation. The field T_INTFRAG is the TLV type and the field LENGTH is the TLV length. The field Fragment Stream ID is 64-bit random identifier of the fragment stream assigned by the fragment originator. In some implementations, it could be a hash of the entire Interest message. The field Path Minimum MTU is the smallest MTU seen along the Interest's path. The field FragCnt is the number of fragments in the Interest message and FragNum is the 0-based index of the current message in the fragment stream.

```
                    1                   2                   3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+---------------+---------------+---------------+---------------+
|          T_OBJFRAG            |             LENGTH            |
+---------------+---------------+---------------+---------------+
|                      Fragment Stream ID                       |
|                                                               |
+---------------+---------------+---------------+---------------+
|      Object Maximum MTU       | X X | FragCnt | X X | FragNum |
+---------------+---------------+---------------+---------------+
|                      Interest Stream ID                       |
|                                                               |
+---------------+---------------+---------------+---------------+
```

Figure 2. Interest Fragment Header

Figure 2 shows the header added to a Content Object message for end-to-end fragmentation. The field T_OBJFRAG is the TLV type and the field LENGTH is the TLV length of the header. The Fragment Stream ID is an identifier of the content object fragment stream. It should be derived from the intrinsic properties of the Content Object, such as from its hash. It may be a random number. The Interest Stream ID must match the Interest's Fragment Stream ID to facilitate matching on the reverse path. The Object Maximum MTU is the maximum size of the fragment stream. It is needed to ensure that a response is only forwarded along reverse paths of sufficient size. The FragCnt and FragNum are as in an Interest.

Considering fragmentation, a system may aggregate two Interests if they have the same constraints *and the second Interest's MTU is greater than or equal to the first.* That is, if a second Interest arrives and has a smaller Path Minimum MTU, then it must not be aggregated because a response Content Object may be larger than its allowable MTU and thus not usable.

1. Originate Interest

   (a) Interests must be fragmented by the originating system to the minimum MTU (1280). The routable name components must be included in the first fragment. The Path Minimum MTU must be set to the maximum supported value.

2. Transmit Interest

   (a) When an Interest goes out a specific interface, the Path Minimum MTU field must be set to the minimum of the current value or the interface's MTU.

3. Receive Interest

   (a) A system must wait to receive fragment index 0, because that is the fragment that has the routable information in it. If other fragments arrive out of order, a system may buffer them[1]. Because all forwarding requires fragment 0, out of order arrival is limited to the link to the previous CCNx hop.

(b) Based on fragment index 0, the system looks up the Interest in the PIT.

i. If no similar interest exists, then it creates a new PIT entry, noting the Fragment Stream ID and the Path Minimum MTU.

ii. If a similar interest exists with the same Fragment Stream ID, the current fragment accounted in the BitMap, and if the first occurrence of the index, forwarded.

iii. If a similar interest exists with a different Fragment Stream ID

A. If the existing Interest's Minimum Path MTU is less than or equal to the current Interest's Minimum Path MTU, then the current Interest is aggregated with the existing Interest.

B. If the existing Interest's Minimum Path MTU is greater than the current Interest's Minimum Path MTU, then the current Interest cannot be aggregated and must be forwarded. The current Interest is treated as a separate PIT entry.

4. Originate Content Object

(a) A process that receives an Interest and satisfies it with a Content Object must ensure the Content Object MTU is no larger than the Interest's Path Minimum MTU. It must fragment or re-fragment the Content Object for that MTU, otherwise it must not respond. An example of such a process is the Content Store or an application.

5. Receive Content Object

(a) Match it to PIT entries based on the Interest Fragment Stream ID carried in the Content Object Fragment Header.

i. If no Content Object Fragment Stream ID is recorded in the PIT, then note the current fragment stream's ID in the PIT and forward it.

ii. If the PIT entry already has a Content Object Fragment Stream ID, then drop the current packet if the Fragment Stream ID does not match.

iii. The PIT must track which fragments of the Content Object have been forwarded along the reverse path, similarly to how they were tracked for Interests along the forward path.

(b) If there is a Path MTU recorded in the PIT entry equal to or greater than the Content Object's Maximum Path MTU, then the Content Object fragment stream may be used. Forward the Content Object along those reverse paths and remove them from the PIT entry

6. Transmit Content Object

(a) No special action is taken. The Content Object's Maximum Path MTU is fixed by the end system that created the fragment stream.

Table Design, memory size, and complexity

We will also assume that all reads are 64 byte cache lines. This is true on most Intel CPUs. The NP-4 reads the first 64 bytes, then the next 32 bytes, keeping those 96 bytes in cache, then alternating two 32-byte buffers. We will assume 64-bit pointers.

Terms:

1. Let R be the line rate, in bits per second.
2. Let K_b be the average Content Object bytes to Interest bytes.
3. Let K_h be the fraction of Interests that carry a Content Object Hash restriction.
4. Let N be the number of CPUs or NPUs.
5. Let B be the average bytes in a Name.
6. Let C be the cache line size (i.e. 64 bytes for Intel or 32 bytes for NP4).

With these terms, the maximum average Interest rate is the line rate divided by the total transmission units, $K_b + 1$. That is, if we have a byte ratio of 10:1 for Content Objects to Interests, then there are 11 transmission units in each direction, one of which is an Interest. This assumes there are Interests and Content objects flowing in both directions with statistically identical distributions.

$$R_{int} = \frac{R}{K_b + 1}$$

The maximum average Content Object bit rate is the remainder of the available bandwidth.

$$R_{obj} = R - R_{int} = \frac{K_b * R}{K_b + 1}$$

[1] If a system discards out-of-order packets, it should send a control message back towards the source asking for the Interest to be re-sent. It should retain the index 0 fragment.

As an example, let us assume we have a 15:1 ratio, such as 1500 byte Content Objects and 100 byte Interests, then $R_{int} = 0.0625R$ and $R_{obj} = 0.9375R$. For a 100 Gbps interface, that's 6.26 Gbps Interests and 93.75 Gbps Content Objects. If we have 64 Kbyte Content Objects and 100 byte Interests ($K_b = 655$), the rates are $R_{int} = 0.0015R$, and $R_{obj} = .9985R$.

Therefore, the maximum average SHA-256 computation rate is

$$R_{sha} = K_h * R_{obj}$$

If we assume that 90% of Interest carry a Content Object Hash restriction ($K_h = 0.9$), then for 15:1 ratio and 100 Gbps, the SHA-256 rate is 84.375 Gbps. For a 655:1 ratio, the SHA-256 rate is practically 100 Gpbs.

On the Intel Haswell CPU[2], Intel reports as few as 2.67 cycles/byte. For 12.5 Gbyte/sec, the necessary CPU rate is 33.375 GHz. At 4 GHz per thread, that requires 8.34 threads. So, in practice with 80% efficiency, it needs 10.5 threads, or perhaps 1.5 quad core CPUs. So, a practical system would likely require 3-4 CPUs, where each core would have a SHA-256 thread and a forwarding thread, giving 4 computations per quad core CPU.

PIT Memory Use

To estimate the size of a PIT table, let us assume a single PIT table entry is the tuple {*KeyPtr, IngressSetPtr, EgressSetPtr, ExpiryTime*}. The KeyPtr is memory on a free-list[3] that stores the PIT entry's hash key, which is the tuple {*Name, KeyId, ContentObjectHash*}. This estimate does not consider fragmentation.

Based on data from Cisco[4], a naive mapping of today's RESTful requests to Interests would make most Interests a few hundred bytes. We will assume that they are 750 bytes, on average, which is likely much larger than the true population average. The KeyId and ContentObjectHash are 32-byte SHA-256 values. Not every Key will have both a KeyId and ContentObjectHash, but we will assume both fields are present, plus a one byte flags field to indicate presence. An average free-list entry is 814 bytes (750 + 32 + 32). Therefore, each Key is 1KB, on average, using the proposed free-list.

The IngressSetPtr and EgressSetPtr track the interface Ids of where the Interest came in (IngressSet) and where it has been forwarded (EgressSet). We will assume ports are a 1-byte slot, 2-byte interface value and are allocated in 64-byte free-lists, so there are 16 entries per entry.

The ExpiryTime is a 64-bit system time indicating when the PIT entry expires.

The number of bytes used by a PIT entry is then give by *PitEntryBytes*, which counts the total bytes used by the hash table entry and the free-list entries. The factor b is the inefficiency multiplier for the hash table, say 1.4 for 70% efficient.

$$\begin{aligned} PitEntryBytes \; &= \; \beta\,(4*size\,(ptr) + \; size\,(time)) + \; size\,(key) \\ &+ \left\lceil \frac{N_{ingress}}{16} \right\rceil * size\,(PortSet) \\ &+ \left\lceil \frac{N_{egress}}{16} \right\rceil * size\,(PortSet) \end{aligned}$$

Assuming that, on average, $N_{ingress} \leq 16$ and $N_{egress} \leq 16$, then the average $PitEntryBytes = 1.4*(32+8)+1024+64+64 = 1208$ bytes, where the first 40 bytes (56 bytes with inefficiency) are the PIT hash table entry and the remaining bytes are free-list allocated using 64-byte and 1024-byte entries.

Let R_{int} be the Interest rate (e.g. 10 GByte/sec), B_{int} be the average Interest size (e.g. 1500 bytes), RTT be the average Content Object round trip time (e.g. 200 msec), T_{max} (e.g. 2 sec) be the maximum Interest lifetime, and a (e.g. 80%) be the successful response fraction, then the number of outstanding interests, N_{int}, is

$$N_{int} = \; \alpha * \frac{R_{int}}{B_{int}} * RTT + (1-\alpha) * \frac{R_{int}}{B_{int}} * T_{max}$$

For the example values given, N_{int} = 1.06 million + 5.33 million, so approximately 6.5 million interests. This value is dominated by the system-specific maximum lifetime, in this example 2 seconds.

Therefore, the average PIT table size under load is $N_{int} * PitEntryBytes$, which is 7.852 Gbytes, including the 1.4x inefficiency multiplier. Therefore, it would be reasonable to design a system with up to 16 GB of memory for the PIT.

FIB Memory Use

Based on a survey from Cisco[5], there are approximately 10^{12} URIs indexed on Google and approximately 10^8 DNS names. Therefore, a FIB table could be around the same order of magnitude in core routers. We do not expect the number of routes to scale as the count of URIs – the CCNx hierarchical name aggregates collections. We believe the number of core routes would be somewhere between the DNS scale and the URI name scale, such as 10^9. End systems and edge routers do not need an entire routing table; they only need to know how to reach core routers.

In a recent paper, CuckooSwitch[6] showed an Intel DPDK router with 10^9 IP FIB entries forwarding 47 Gbps on one

[2]`http://www.intel.com/content/www/us/en/communications/haswell-cryptographic-performance-paper.html`

[3]We will assume 16 bytes for doubly-linked list, so free-list sizes of 48 bytes, 496 bytes, 1008 bytes, etc., for different pool sizes.

[4]Cisco's presentation to ICNRG at IETF 89, `http://www.ietf.org/proceedings/89/slides/slides-89-icnrg-10.pdf`

[5]`http://trac.tools.ietf.org/group/irtf/trac/attachment/wiki/icnrg/IRTF-CCNAndIPRouting-2.pdf`

[6]D. Zhou, Bin Fan, H. Lim, M. Kaminsky, and D.G. Andersen. 2013. Scalable, high performance ethernet forwarding with CuckooSwitch. CoNEXT '13.

CPU. Therefore, it should be feasible to make a CCN FIB scaled to at least the DNS domain name scale.

In terms of memory, if we assume a similar 2,4-cuckoo hash table as in CuckooSwitch, then each table entry is 2 buckets of 4 hashes, where each bucket fits in a 64 byte cache line. Therefore, we achieve 4 keys in 128 bytes (there's two hashes per key, one in each bucket) plus free-list name entries for names and EgressSetPtrs. Therefore, a hash based routing table requires *FibEntryBytes*.

$$\begin{aligned} FibEntryBytes &= \frac{\beta * 128}{4} + size(name) \\ &\quad + \left\lceil \frac{N_{egress}}{16} \right\rceil * size(PortSet) \end{aligned}$$

As above, b is the hash table inefficiency, which we will continue to assume is 1.4. Let us assume that the average forwarding FIB entry will fit in 240 bytes, on average, so a single 256 byte free-list allocation is needed. We will also assume that $N_{egress} \leq 16$ and the *PortSet* size is 64 bytes. Therefore, $FibEntryBytes = 44.8 + 256 + 64 < 365$ bytes.

Therefore, for DNS-scale FIB tables of 10^8 entries at 365 bytes per entry, the total FIB size would be 37 GB. A reasonable system design would thus allow for 64 GB in a core router, perhaps with expansion capability to 328 GB for 10^9 entries. It is likely not practical to have so much memory on a single NPU or CPU, due to limits on memory bandwidth, so the FIB table may need to be sharded between multiple cores. Such large tables would also impose considerable strain on a routing protocol that must transport the names and perform path calculations. One could realize a small savings if the Internet routing protocol limited routable prefixes to a certain length, such as up to 3 name components of no more than 128 bytes total.

Protocol Timers

Interest messages carry a Lifetime parameter, measured in milli-seconds (msec). This is the maximum time that a router should consider the Interest valid, though it may discard an unanswered Interest sooner. A forwarder does not need to actively discard Interests from the PIT, it may use a lazy strategy.

If a second Interest arrives at a router with a longer Lifetime than the remaining time in a current similar entry, the forwarder has two choices. It could aggregate the Interest and at the expiry of the first Interest forward the second with the residual lifetime in it. Or, it could forward the second Interest immediately, trading the traffic overhead for reduced state and timers.

Content Objects in the Content Store may carry expiry times (an absolute value) or freshness values (a relative time decremented at each hop). A Content Store does not need to actively purge Content Objects when the timer runs out, it may use a lazy strategy.

An implementation of the foraging strategy requires many fine-grained timers. It may be appropriate for some smaller networks or low-speed networks, but is not appropriate for high-speed routers.

If a router uses Nonces for loop prevention and duplicate detection, then some implementations may set timers to expire nonces. We do not advocate this practice. Rather, we construct a hash table for nonce lookup combined with an LRU nonce list so the oldest nonces can be overwritten by new nonces.

Label Forwarding (Express Headers)

Express Headers are a way to pre-compute several expensive keys used to lookup Interests and Content Objects. They reduce the load on each hop of CCNx messages.

Express Headers is a method to use fixed size, flat byte strings to forward CCNx packets with Hierarchically Structured Variable Length Identifiers (HSVLI), thus simplifying the work done at a packet forwarder. A first byte string, called the Similarity Hash (SH), represents the query in an Interest. The Similarity Hash remains invariant as a packet moves through the network. A second byte string, called the Forwarding Hash (FH), represents the longest matching prefix in the routing tables that matches the Interest name. In one variation, the Forwarding Hash may change hop-by-hop if the underlying routing tables change, such that it always represents the best match at the previous hop. In another variation, the Forwarding Hash is calculated by the end node for first hop, and remains constant for the life of the Interest. A Content Object, sent in response to an SH/FH Interest, carries the SH/FH header along the return path so the Content Object may be forwarded along the proper path.

In order to forward CCN packets on the fast path we need to do some form of name matching in the PIT and FIB. As discussed in the previous section one way to do this is to go over the complete name and find the longest prefix match (for FIB) and an exact match (for PIT/CS). This is normally done via hashes, but that means we would need to calculate multiple hashes over potentially long sections of packets. To alleviate this we use express headers, where a set of labels are used in place of name matching.

Matching based on express headers does not require special hardware. It is possible for a network processor to take care of matching the required fields. At this point in time the minimum hardware requirements is ability to match 16 byte fields on a data structure. This is true both for content objects as well as interests. The values don't need to be calculated; they are taken straight off the first part of the packet (per-hop headers).

Using Express Headers may result in significant CPU and memory savings. This is because the forwarding information is summarizes in 32 bytes, not a long variable length name. This could result in about a 10x to 20x memory savings. It also eliminates the need for high-speed lookup of long CCN names.

Autonomous System Switching

As shown in Fig. 3, one may organize the components of a CCN system throughout an Autonomous System. This

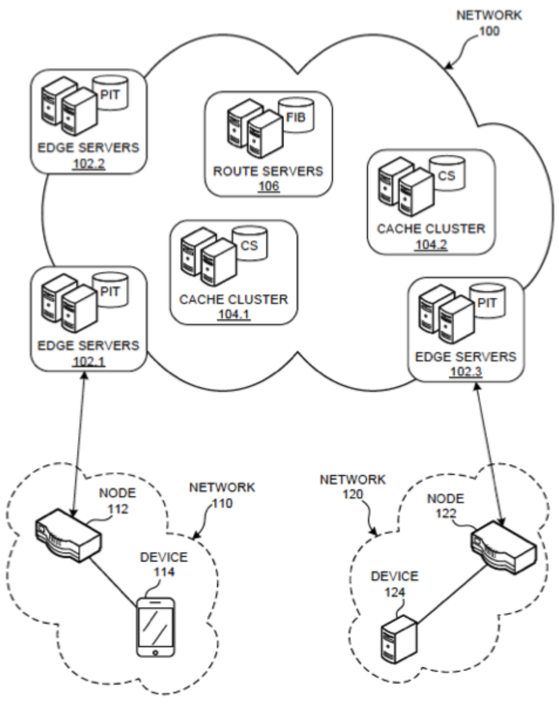


Figure 3. Autonomous System Switching

distributes the work horizontally over many systems while retaining locality of content.

This construction provides a content-centric network (CCN) autonomous system (AS), which solves the problem of distributing various CCN processing tasks and resources across a plurality of computing devices. Specifically, the CCN AS can include a set of edge servers, cache servers, and routing servers that can use label switching (e.g., Multiprotocol Label Switching, or MPLS) to control the flow of interests and content objects within the AS.

The edge servers (routers) interface with external systems to process interests for content objects, and also maintain a pending interest table (PIT) to keep track of pending interests. The cache servers include a repository for caching content objects that are likely to be requested in the near future. A cache server can satisfy an interest by returning a corresponding content object directly to an AS node that has attached a label to the interest. The route servers, on the other hand, are optimized to perform fast lookup operations on a forwarding information base (FIB) to determine an outgoing "face" of the CCN AS to use to forward an interest to a remote system. The CCN "face" implements a virtual interface for the CCN AS, such that multiple device interfaces can handle data traffic for the CCN face. When an egress edge server of the CCN AS receives a content object that satisfies an interest, the egress edge server can forward the content object toward a corresponding ingress edge server, while bypassing the route servers.

In contrast, a typical CCN device can include a content store (CS), a FIB, as well as a PIT. Hence, an autonomous system that is realized using typical CCN devices would need to duplicate the full set of CCN tasks across all network nodes, which would make it difficult to scale the AS to process a larger throughput of interests, or to satisfy a larger range of structured names.

For example, in a typical CCN device, the FIB needs to be updated periodically, such as when the CCN device receives advertisements for new content. The CCN device uses the FIB to determine an interface that can be used to forward an interest to another network node. Hence, an AS that is implemented using typical CCN nodes would require all AS nodes to include a FIB to determine how to forward an interest toward an egress node of the AS. However, in embodiments of the present invention, not every CCN AS node needs to maintain a FIB. Some CCN AS nodes can be configured to determine how to forward an interest to a network outside the CCN AS (using a FIB), while other CCN AS nodes can be configured to process an interest internally (e.g., a cache server). Hence, CCN AS nodes that are not configured to route an interest to a remote system do not need to maintain or access a FIB.

Further, in a typical CCN device, the PIT needs to be updated at a high rate, such as when a new interest is received or when an old interest is satisfied. The PIT stores entries for interests that are to be matched against incoming content objects, and each entry is only removed when a match is made, or when an interest times out. In some cases, the timeout period can be relatively large, in the order of a few seconds. This can cause the CCN node to turn over several gigabytes of data in the PIT every few seconds, such as to add entries for new interests and to remove entries interests that have been satisfied or have expired. Hence, the PIT typically needs to be updated at twice the data throughput of the CCN device and can grow substantially large, which makes it impractical for an AS to maintain a PIT at every network node. In embodiments of the present invention, the CCN AS uses edge nodes to maintain a PIT for the AS, thereby relieving other types of CCN AS nodes of having to maintain a PIT.

Forwarding Summary

Table 1 summarizes the forwarding technologies for CCNx 1.0 and some PARC developed techniques to improve system performance.

Table 1. Summary of Forwarding Plane Technologies

Technology	Results
CCNx 1.0	**Internet Router 100 Gbps Line Card** • PIT size: approximately 8 – 16 GB • DNS-scale FIB size: approximately 32 – 328 GB • SHA-256 hashes: approximately 3-4 CPUs **Enterprise router 100 Gbps Line Card** • PIT size: 8 – 16 GB • Enterprise-scale FIB size: under 1 GB • SHA-256 hashes: 3-4 Haswel CPUs
Express Headers	**Advantages** • Fixed 32 bytes of header for forwarding • Greatly simplifies long name routing • Reduces PIT and FIB sizes **Disadvantages** • Global agreement on name hash functions or route label lookup database (either on end node DNS or in mid-hop FIB)
Autonomous System Switching	**Advantages** • Functions (PIT,FIB,CS) scale horizontally and vertically • Only edge nodes have PITs • Only cache clusters have Content Store • Only router servers have FIBs • Simple inter-system routing protocols with small state • Direct reverse path cut-through of Content Object from Interest egress node to Interest ingress node **Disadvantages** • Requires underlying switching technology (e.g. IP/MPLS)

1.2 Control Plane

There are many controllable components in a CCN router. Some components are similar to IP routers and are managed and controlled in a similar fashion. Other components are custom to CCN.

The Control Plane primarily consists of invoking operations on a CCN Router that cause it to report or modify its current or future behavior. Operations originate from a potentially diverse set of sources. See the section on Management for more information about the management sources of control information.

The CCN-router must present an interface to all of the controllable aspects of the device.

FIB Control Plane

The FIB Control Plane observes and governs the routing information stored in the CCN FIB in a router.

The FIB Control Plane must support or provide the ability for a properly authorized directive to cause a CCN Router to:

1. Report the number of entries in the FIB.
2. Report the details of one or more FIB entries.
3. Create or update a FIB entry.
4. Remove a FIB entry.

Furthermore, the FIB must support the management features that are likely to include historical and statistical data on the FIB entries themselves.

PIT Control Plane

The PIT Control Plane observes and governs the dynamic, return-path routing information stored in the CCN FIB in a router.

The PIT data is very dynamic as in normal operation every CCN Interest packet stores a PIT entry, every CCN Content Object reply examines and deletes a PIT entry, and each PIT entries has a time-to-live, which causes them to expire.

The PIT control plane must support the creation of a PIT entry, a PIT lookup by name and delete by name.

The CCN-router must at least support the management ability to query for a PIT entry. Other features may be the ability to create, update and delete a PIT entry.

The PIT contains reverse-path routing information for Interest packets

Content Store Control Plane

The Content Store Control Plane observes and governs the dynamic state and content of multi-tiered content storage in a CCN router.

Each tier may provide features relevant to that tier. For example, tier 0 may have multiple strategies for adding new content while tiers 2 and 3 may have sophisticated, cooperative mechanisms for managing long-term cached data. Each of these tiers will have Control Plane operators to govern the content store.

A CCN-router Content Store may be many terabytes or petabytes in size and will have specific control plane requirements that address the specific storage technologies that comprise the Content Store.

The CCN-router must be able to populate its Content Store via the network, using interests and caching the results as well as external storage devices.

The content store must be able to delete content by selected fields in the content objects. For example, the name, publisher, time-to-live, or by specific hash

1.3 Management Plane

The management plane in a typical IP router coordinates functions across the three planes - management, control, and data and it may also perform management functions for the entire network. The management plane also is used to manage a device through its connection to the network. The management plane typically exposes a command-line interface as well as supports a variety of protocols such as Simple Network Management Protocol (SNMP), Telnet, HTTP, Secure HTTP (HTTPS), and SSH. These management protocols are used to monitor the device (or network), gather statistics such as number of packets forwarded or dropped packets count as well as appropriately configuring various parameters on the router.

A CCN router must support all the functions listed above. In addition, it must be able to support functions specific to CCN including managing trust policies, cache pre-population, key management, coordinating functions across the CCN control plane, etc.

In a CCN router, the management plane itself can be implemented over CCN. One can use the CCN protocol itself to gather statistics from the router, set appropriate configuration parameters, etc. A CCN router must expose appropriate interfaces to enable these tasks. The management plane must take into account the federation of multiple routers and also provide the capability to manage the network as a whole.

Handling CCN specific functionalities:

We do not mandate CCN routers to verify the signature in every CCN Content Object that it forwards; yet we do not preclude routers from verifying content signatures. In case a CCN router wants to support signature verification, the management plane must support the following features:

1. Ensure appropriate public key or other appropriate certificates are loaded
2. Certificates are still valid and have not been revoked or expired
3. Ensure that appropriate signature verification programs are appropriately loaded
4. Ensure that appropriate trust mechanisms are available and correctly configured on the router

Even if a CCN router will not be checking content signatures, some of the above functionality needs to be supported by the management plane for supporting secure routing protocols, traffic engineering, etc.

The management plane in a CCN router may also be require to optionally support Content Store management tasks such as predicting content objects that would be requested and prefetching such content.

Other functionality that the management plane must support includes ensuring availability of hash calculation programs, ensuring that all software on the router is regularly updated and/or upgraded, etc.

Early CCN routers may have a dual stack – one for IP and one for CCN. In such cases, each stack must have its own management plane.

2. Research Router Platform

2.1 PARC-OS Architecture

PARC-OS is a layered, distributed architecture as shown in Figure 4 consisting of:

1. Hardware layer
2. Virtualization layer
3. Guest Kernel Layer
4. Application layer

PARC-OS architecture's HW layer is the research platform, which provides:

1. A control Ethernet TIPC network for the cards in a chassis to communicate with each other.
2. Fabric switch and fabric interface associated with each card for the data forwarding.
3. Network Processor Unit (NPU) that does the data forwarding in the fast path.
4. Storage device for storing software images, logs and configuration. It can be used for caching the content objects for CCN protocol.

PARC-OS architecture's SW layer provides:

1. PARC's Linux distribution.
2. Control Plane based on Open source Quagga.

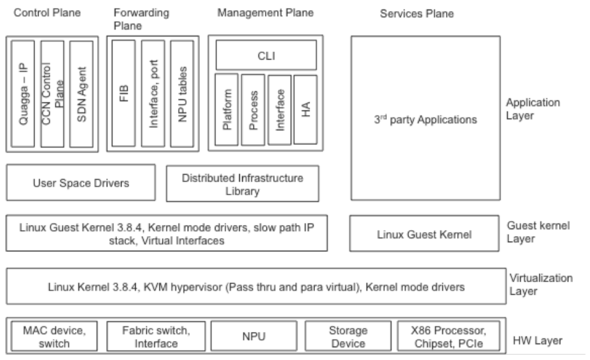


Figure 4. PARC-OS Architecture

3. Forwarding plane software that distributes the FIB to ELC and populates the FIB entries in the kernel and the NPU.

4. Distributed infrastructure provides high availability, faster application development and the management plane software.

The Linux distribution is based on Open Source Linux 3.8.4 with yocto embedded Linux build system. The open source linux with its rich set of tools and applications makes the PARC-OS system to run any Linux applications with no modifications.

The virtualization layer runs Linux kernel 3.8.4 with KVM 1.4.2 hypervisor. The virtualization layer provides the following benefits:

1. In Service Software Upgrade

2. Resource isolation to implement features like multi-tenant, separation of services

The guest kernel layer also runs Linux kernel 3.8.4. Performance critical devices such as NPU, Fabric devices, MAC devices are passed thru directly to the Guest Linux kernel. The slow devices such as IPMI, I2C, CPLD are para-virtualized.

The Application software in the routing VM is divided into three planes. The control and management plane runs on the switch cards with redundancy (SXC) and the forwarding plane runs on the Line Cards (ELC).

1. Control Plane

2. Forwarding Plane

3. Management Plane

The control plane consists of a set of Linux processes that are responsible for computing the control and policy information. The control plane provides the computed information to the forwarding plane.

The forwarding plane consists of a set of Linux processes that programs the NPU's tables with the information provided by the control and management plane. The forwarding plane also provides abstraction of the physical and virtual ports, and monitors the ports for faults and provides the information to the control and management plane.

The management plane is responsible for configuration and management of the router VM. The CLI provides configuring ports as interfaces, managing the platform functionality such as upgrading software images, behavior of the process monitoring and restart and redundancy management.

The distributed infrastructure provides High Availability such as process, VM restarts, managing watchdog functions and faster application development.

2.2 IP Control Plane

The IP Control plane is based on Open source Quagga – Industry standard IPv4/v6 routing software. Quagga supports the routing protocols OSPF, BGP, RIP, ISIS and is distributed with the GPL license. The Quagga suite also includes a CLI component to configure the routing protocols.

2.3 IP distributed forwarding Plane

The Quagga control plane using the routing protocols computes the Routing Information Base (RIB) and Forwarding Information Base (FIB) based on the topology, configuration, and information from neighboring routing devices.

Quagga installs the FIB tables to the local kernel for slow path forwarding. In PARC-OS distributed chassis based architecture, the FIB needs to be distributed to all the Line cards for fast path forwarding. The Forwarding Plane Manager (FPM) does the FIB distribution. FPM listens for the kernel route table updates and distributes the information to the fibagent process running on the Line Cards. The fibagent process on the line cards manages the FIB table for both the local kernel and the NPU.

2.4 SDN Agent Control Plane

In SDN the computing of the routes itself is done outside of the router. The SDN agent receives Openflow messages and is responsible for distributing FIB to all the Line Cards. We are looking into implementing the SDN Agent.

3. Research Platform

In order to provide a platform for CCN research and development, PARC entered into a partnership that yielded a pair of terabit speed, high availability, carrier class routers code named **Penn** and **Teller**. These chassis-based systems offer up to 11.2 terabits of switching capacity that interconnects a variety of intelligent network line cards, and these cards are interchangeable between Penn and Teller. Fully redundant hardware components assure no single point of failure, and a completely virtualized software environment provides the high availability required of web-scale service providers.

3.1 System Architecture

The most salient features of Penn and Teller's system architecture are:

Table 2. PARC-OS Features

Functionality	Description
Languages/Tools support	Scripting languages – shell, python, perl, ruby, Programming languages C, Java. Easy to bring in additional packages that are compatible with Linux 3.8.4
Layer 3 features	Industry standard ipv4/ipv6 Quagga routing suite including OSPF, BGP, ISIS, RIP. Distributed forwarding plane to distribute the FIB to the Line cards and kernel.
Management	IPMI, CLI
High Availability	Process monitoring and restarts, watchdog monitoring for VM level and card level restarts, SXC fail-over for control plane availability. Hardware designed for high availability.
Virtualization	Control plane, management plane and forwarding plane runs in VM and allows launching additional VMs for services and 3rd party software without impacting the routing functionality. Makes In Service Software Upgrade possible. Critical HW devices are directly assigned to the VM for better performance.

1. Non-blocking, memory-less, load balanced, fully redundant switch fabric.
2. Completely isolated Data, Control, and Management planes.
3. Totally independent chassis management path to the Shelf Manager.
4. Uses no custom ASICs.

As shown below, each line card connects to the switch controller cards via dedicated serdes lanes on the chassis backplane

3.2 Penn

Penn is a 14-slot chassis that can house up to 12 network line cards within 16 Rack Units (RU) of vertical space. Each card slot can access up to 1.1 terabits-per-second (Tbps) of half-duplex switching capacity. The system accepts up to four switch fabric controller cards that provide a non-blocking switch fabric between the line cards. The chassis backplane provides 48 serial "serdes" lanes from each network card to the switch controller cards, offering plenty of headroom for future bandwidth expansion.

3.3 Teller

Teller delivers exceptional performance and high availability in a compact footprint that accommodates up to four network cards. The 6 RU form-factor is achieved by integrating the

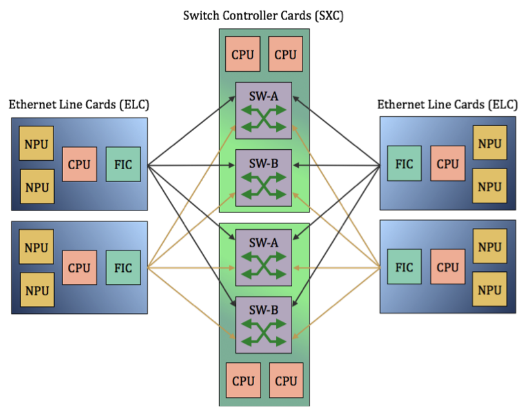

Switch Controller Cards (SXC)
Ethernet Line Cards (ELC)
Ethernet Line Cards (ELC)
CPU
NPU
FIC
SW-A
SW-B

Figure 5. Research Platform Architecture

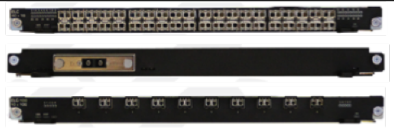
system controller, switch fabric, and shelf manager into one half-width card, providing dual redundancy within a single vertical space.

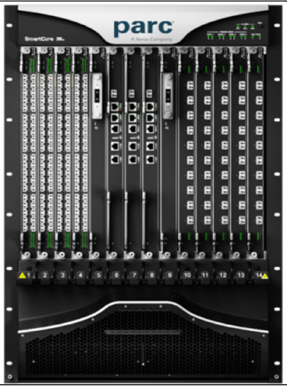
3.4 Ethernet Line Cards (ELC)

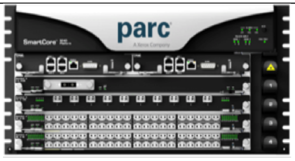
Penn and Teller systems comprise four different line cards distinguished by speed and number of Ethernet ports. These intelligent cards include a 1.73 GHz Intel Xeon dual-core processor with 4 Megabytes (MB) of L3 cache and two DIMM sockets providing between 16 and 64 Gigabytes (GB) of main memory. This provides lots of available compute power to handle modern SDN applications.

Each ELC includes two Network Processing Units (NPUs) with dedicated low latency memory for forwarding tables and statistics, along with a 2x1 MB 80-bit Ternary Content Addressable Memory (TCAM) for line rate packet processing. A Switch Fabric Interface chip drives 48 serial "serdes" lanes to the switch fabric. The control plane is implemented with 1 Gbps Ethernet links between each ELC and the system controllers.

The four types of Ethernet line cards are summarized below:

Ethernet Line Cards	
ELC 40x1	Number of Ports: 40 Data Rate: 1 Gbps
ELC 100G	Number of Ports: 1 Data Rate: 100 Gbps
ELC 10x10	Number of Ports: 10 Data Rate: 10 Gbps
ELC Combo Card	5 Ports @ 10 Gbps 20 Ports @ 1 Gbps

Penn System Specs		
Physical Dimensions	Height: 16 RU Card Slots: 14 Units per 19" Rack: 3	
Card Capacity	Controllers: Up to 4 SXCs Line Cards: Up to 12 ELCs	
Switching Capacity	Backplane: 11.2 Terabits Half Duplex Per Slot: 1.1 Terabits Half Duplex	
High Availability Features	Dual Redundant Power Entry Modules Dual Redundant Fan Trays Dual Redundant Shelf Manager Cards Dual Redundant -48 VDC Power Inputs	

Teller System Specs		
Physical Dimensions	Height: 6 RU Card Slots: 4 Units per 19" Rack: 8	
Card Capacity	Controllers: Up to 2 CFMCs Line Cards: Up to 4 ELCs	
Switching Capacity	Backplane: 4.6 Terabits Half Duplex Per Slot: 1.1 Terabits Half Duplex	
High Availability Features	Triple Redundant Power Entry Modules Redundant Fans Dual Redundant Shelf Manager Cards Dual Redundant -48 VDC Power Inputs	

3.5 Switch Fabric Controller (SXC)

For the Penn system, the Switch Fabric Controller, or SXC, implements the switching fabric between line cards and handles control plane processing. Each SXC card includes two 1.73 GHz Intel Xeon quad-core processors, each with 8 MB of L3 cache and three DIMM sockets that provide up to 96 GB of main memory. Two switch fabric chips reside on the SXC to switch data flows between line cards, and a switch fabric interface chip enables the SXC to process control plane traffic. The SXC also provides a 1 Gbps Ethernet switch that interconnects all line cards for Out-Of-Band (OOB) management functions.

Figure 6. SXC Controller Card

As depicted above, the SXC front panel offers two Ethernet connectors for management and control plane communication. There is also a serial console port for management and diagnostic access.

3.6 Control Fabric Management Card (CFMC)

For the Teller system, the Control Fabric Management Card integrates the system controller, switch fabric, and shelf manager onto a single half-height card. CFMC includes one switch fabric chip and one 1.73 GHz Intel Xeon quad-core processor with two DIMM sockets that support up to 32 GB of memory. As seen below, the front panel provides two gigabit Ethernet interfaces for management and control plane access, along with a serial console port for diagnostics.

Figure 7. Control Fabric Management Card

3.7 Content Object Storage

On Penn and Teller, the Content Object repository is a centralized entity residing on switch controller cards. The diagram below depicts the PIT tables, Content Storage Lookup Tables (on ELC and SXC) and FIB tables to route interest and content packets.

There are many issues that need careful consideration when designing the storage hierarchy for Content Objects. This will be a major focus of ongoing research in CCN, and some of the initial observations are as follows:

1. CCN content objects require cryptographic processing, so one needs to consider the CPU processing bandwidth available when deciding where to locate the storage elements.

2. Useful life of a given content object: The "shelf life" of an object on core routers is orders of magnitude smaller than that of edge routers. As a result, edge routers will be provisioned with more externally managed storage compared to core routers.

Table 3. Content Object Storage Tiers

	Penn	Teller
Tier-0	96 GB of DDR-3 memory on each SXC.	32 GB of DDR-3 memory on each CFMC.
Tier-1	On-board SATA SSD on SXC. 80 GB to 1 TB currently available.	On-board SATA SSD on CFMC. Currently 32 GB, up to 128 GB available.
Tier-2	iSCSI based externally managed storage arrays attached to 1/10/100Gig ELCs.	iSCSI based externally managed storage arrays attached to 1/10/100Gig ELCs.

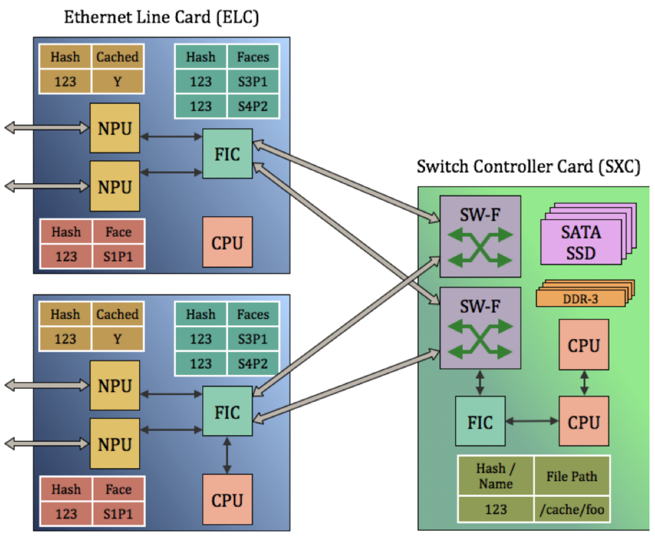


Figure 8. Object Storage Architecture

A tiered approach is necessary to implement various strategies implemented by the control plane for storing content objects. The following table outlines the hierarchy available on the Penn and Teller research platforms:

4. CCN Implementation Notes

4.1 Specialized fast-path hardware

CCN forwarding has a number of requirements that make the fast past challenging at high data rates. To achieve these forwarding requirements we will require cooperation with certain hardware elements.

For example, using a specialized hash processor could enable wire-speed hashing of CCN messages, including various specialized embedded hashes. The hash processor could, for example, execute a SHA-256 hash on an entire Content Object while concurrently executing a SipHash over successive name components for use in a PIT table lookup.

4.2 Name-based Matching

CCN names are composed of an arbitrary number of segments of arbitrary size, with a maximum length of 64K. We need to match the names in an interest packet to a series of data structures. The PIT must match exactly and the FIB must do a longest-prefix match.

The common implementation of this process is to match names using hashes and hierarchical data structures. This is done by using a fast, secure stream hash over the name element by element and storing the hashes at each point. The hashes are then used to match the data structures. For example, the final hash (over the whole name) is used to search in the PIT. The first or second hash are used to start the search in the FIB.

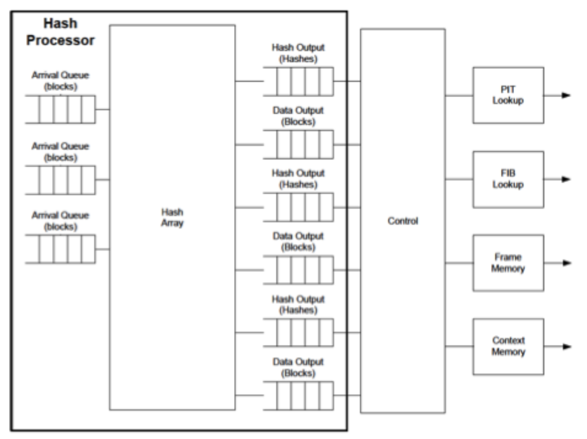


Figure 9. Specialized hash processor

We don't expect the fast path to handle pure name matching in the first hardware iteration. This will be handled in the slow path (on a general CPU).

In the case of future hardware, we would benefit in having a network processor (or pre-processor) that can calculate a hash list for incoming names in packets. This hash does not need to be a cryptographic hash but it must be secure to prevent collision attacks that might affect the router performance.

It is expected that routers that need to perform this calculation will insert and/or fill in the express header so that routers further up the path don't need to perform hash computations.

4.3 Content Object Hash

CCN has a requirement that interests that contain a content object hash must be answered only with a content object whose hash matches the requested hash. This means that we need to verify the hashes match before we send the content object back as a reply to the interest.

The content object hash is a SHA-256 hash of the content-object message. This does not include the per-hop headers or the static packet header. This hash matching must be done at line-rate for the system to process interests with content object hashes in the fast path.

Our current hardware does not support this feature. It's not possible with the current hardware rev to calculate SHA-256s on the content objects at a rate of 100Gbps. Future hardware revisions will have this feature.

There are various ways to achieve this computation. Some ways to achieve this include: custom silicon, general processors, fast network processors with crypto functions and crypto cores on FPGAs.

4.4 Fragment forwarding

CCN requires routers to support end-to-end fragmentation. Specifically, this means that packets don't have to be fragmented in flight. When forwarding fragmented packets, the head will include a fragmentation Id and a fragment number.

If the first fragment doesn't arrive first, the router may drop the packet. If a packet comes in and needs to be fragmented, the router can drop the packet.

The first packet must have enough information to be forwarded. This means it is using label based forwarding (express headers), the it contains the name segments needed for forwarding, or it's a fragmented response (content object) that contains an interest Id.

The fragment Id of interests (interest Id) must be recorded in an index structure so that subsequent fragments can be forwarded on the same path. This structure must point to the PIT entry. The PIT entry must contain the outgoing face for this fragmented interest so all fragments flow out in the same direction.

In the case of fragmented content objects, these fall in 2 similar categories. If the content object is using express headers, the express header will be used as the identifying field to look up the respective PIT entry. If the content object is not using express headers then the fragmentation header is used to look up the correct PIT entry. This is done via the same index structure used by the incoming interest fragments. Content object fragments will indicate what interest Id they are a response to.

All fragment Ids are 16 bytes. The router must support these 16 bytes for matching and searching. CCN defines that a packet can have at most 64 fragments so a single 64 bit array can be used to keep track of what fragments have been received and forwarded. In most cases the maximum number of fragments will be less than 64 given that the maximum packet size is 64K, the minimum MTU is 1280 and that fragmentation must be packed (only last fragment smaller than detected MTU).

4.5 Fragment reassembly

CCN has a requirement that interests that contain a content object hash must be answered only with a content object whose hash matches the requested hash. This has implications on fragmentation.

If a PIT entry specifies that the interest requested a specific content object hash we can't forward fragments until we have checked that the resulting content object matches. This implies that we need to hold all fragments and reassemble the object before replying. We do not need to re-fragment the object, we can send out the same fragments we received.

We expect fragments will come in close to each other. The previous hop must also have checked the content object hash before forwarding. This means that we don't have to keep fragments around for too long.

If network elements are collaborating and can trust each other the first element to receive the fragments can check the content object hash so subsequent router do not need to perform the check and can do cut-through forwarding of the fragments.

4.6 Caching

The CCN requirements on caching are strict. In order to prevent attack amplification caches (content stores) must only reply to interests under a specific set of conditions.

For interests with a KeyId restriction the cache must only reply if it has verified that the key is the actual key that signed the content object. This means that the key must be available for the verification and that we have the right hardware/software to perform the computation.

The recommended path is for content objects to be verified when they are added to the content store. This can happen in a lazy manner. By verifying at insert time there is no verification required at reply time. This is important for fast responses from cache. Since caching is optional certain caches may choose to not reply to KeyId restricted interests.

For interests with a contentObjectHash restriction the forwarding path needs to check the match. From an implementation perspective it's possible for the cache to compute the contentObjectHash on insert and keep the value around. That way there is no computation needed on reply.

Cache Layers CCN allows caching to happen in the network via content stores. The cache system is a complex one that can be tiered. In many routers we can consider caches divided into 3 tiers.

Each of these caches work by matching interests to content objects using the cache match rules. Like any caching architecture each layer offers a tradeoff between storage and cost (latency).

The policy for storing content objects at any layer in the cache can get quite complex. There is a number of possibilities that need to be explored further. At this time the envisioned use is policy driven (cache content from producer XXX) with a dynamic/adaptive component.

Fast path cache (L0) The fast path cache is a cache that can run at close to line rate. This is normally done via memory. This memory can be part of line cards.

Physical attached storage (L1) Physical attached storage is the next level of caching. This has longer latency but can store drastically more content.

Network attached storage (L2) Network attached storage content can come in different forms. Sometimes this can be treated as raw storage (iSCSI) and other times it might use a more complex protocol to reach a cache node. Performance varies depending on the attached component at the other end. It is expected that this will be custom configured in most situations.

SAN/NAS

5. Next Steps

Over the past 12 months, PARC has made tremendous progress towards realizing the research platforms described earlier. Our forward-looking research plan is being executed in three phases:

Project Clean Slate
CCN 1.0
CCN Router

Phase 1: Define and build a "clean slate" research platform We are currently in Phase 1 and already have a demonstrable router running completely open-source software. Next steps include enhancements for manageability and AAA features. These router platforms will be made available for universities and partners to conduct research.

Phase 2: Get CCN running on the research platform This phase will get a CCN 1.0 implementation up and running on our clean slate platform. A hybrid approach will allow IP and CCN traffic to be routed simultaneously.

Phase 3: Build an optimized CCN router In this phase, we take everything learned from Phase 2 and apply it in the design of an optimized CCN router that runs at line rate. This router will enable development of numerous software features and use cases that will become available as the CCN community expands.