STRATEGY & INNOVATION

CONTENT DELIVERY WITH CONTENT-CENTRIC NETWORKING

Prepared by:

Greg White
Distinguished Technologist, Core Innovation
g.white@cablelabs.com

Greg Rutz
Lead Architect, Advanced Technology Group
g.rutz@cablelabs.com

February 2016

Table of Contents

List of Figures

EXECUTIVE SUMMARY

Information Centric Networking is an emerging networking approach that aims to address many of the shortcomings inherent in the Internet Protocol; in current networks and in the networks envisioned in the future. One specific approach to Information Centric Networking, also known as Content-Centric Networking (CCN) and/or Named Data Networking (NDN), appears to be gaining mindshare in the research community and in industry, and promises to significantly improve network scalability, performance, and reduce cost over a network built on the Internet Protocol. CCN/NDN provides native and elegant support for client mobility, multipath connectivity, multicast delivery and in-network caching; many of which are critical for current and future networks, and all of which require inefficient and/or complex managed overlays when implemented in IP. Further, CCN/NDN provides a much richer addressing framework than that existing in IP, which could eliminate significant sources of routing complexity.

As HTTP/IP is baked into the networking field (equipment, software stacks, applications, services, engineering knowledge, business models, even national policies), it may seem daunting to consider the use of a non-IP protocol. However, while IP has been a phenomenally successful networking protocol for the last 40 years, as technology and time progress it is reasonable to believe that we won't be utilizing it forever. In this paper, we examine whether some of the networking issues that exist in current networks already point to CCN/NDN as being a better alternative, and if so, how an HTTP/IP network operator might begin to reap the benefits while minimizing the cost and complexity of the transition.

One aspect of CCN/NDN, the native support for in-network caching, offers a much simpler and potentially more optimal alternative to the Content Distribution Network (CDN) paradigm that has been built to support HTTP/IP content distribution. An investigation into the mechanics of an HTTP-based CDN (as it might be implemented by an Access Network Service Provider) reveals a number of areas in which this approach is strained. The possibility exists for a phased transition of an HTTP-based CDN into a network making use of the CCN protocol, and highly optimized for content distribution. In addition, the native multicast distribution capability of CCN/NDN can be leveraged to optimize the transport of linear video streaming services, without introducing all of the complex control/management plane aspects inherent in IP Multicast.

Our analysis indicates that it is possible to realize significant benefits through an incremental introduction of CCN/NDN into an existing CDN implementation. CCN->HTTP and HTTP->CCN proxies can create small "islands" of CCN/NDN functionality that can begin to eliminate some of the performance and management issues encountered in modern CDNs. By striping content across multiple caching routers at the Content Object-level, the "hot cache" problem is eliminated. Additionally, link utilization is reduced because content retrieved from higher-tier cache levels will not "hairpin" through the caches on the way to the client as it does today. It is important to note that these benefits can be achieved without making modifications to existing IP routers.

CCN/NDN has other features that may be equally compelling for cable network operators (notably native mobility and multipath networking). These are not covered in this report. We anticipate a follow-on report that will investigate those aspects.

To be clear, CCN/NDN is not a mature protocol ready for immediate deployment. A number of R&D issues are still being worked in academic and industry research groups (including CableLabs), and implementations (both endpoint stacks and network gear) are still largely in the reference implementation or proof-of-concept stage. However, development work is active and confidence is high that the protocol will be made ready for prime time in the next 3-5 years.

1 INTRODUCTION

Content-Centric Networking (CCN) is an innovative new networking protocol designed around the fact that today's Internet is largely used as a medium over which *content* flows from producers to consumers. This is in contrast to the host-centric Internet Protocol (IP), which is designed around the paradigm of point-to-point communication between two network endpoints. CCN is a single incarnation of a larger field of research known as Information-Centric Networking (ICN).

In the majority of current network applications, consumers demand specific data objects, but don't particularly care from which server or network endpoint the data is served. In fact, many network technologies we have come to rely upon (Content Delivery Networks, Load Balancers, GeoDNS, etc.) were designed specifically to bridge the gap between the host-centric nature of IP and the information-centric demands of modern Internet applications.

One of the key features of CCN (and several other ICN approaches) is the idea that each network node (i.e., router or endpoint) incorporates a Content Store to potentially cache data that has traversed its interfaces. Similar to an HTTP Transparent Cache, this has the benefit of moving content storage closer to the network edge and the devices that are requesting it. However, unlike an HTTP Transparent Cache, this functionality is built into the network itself, and is enabled due to the fact that CCN provides object security (objects are signed and potentially encrypted by the origin publisher, and stored immutably in caches) rather than connection security (the transport connection from the client to a trust-verified server is encrypted) as is provided in HTTP (via HTTPS). As a result, secure objects are cacheable in CCN, where objects carried over a secure connection are not.

A network in which storage is a first-class entity is expected to operate much more efficiently and with higher performance than today's HTTP/IP network. Moreover, the role of the CDN is largely subsumed by the network itself, instead of relying on the typical "overlay" of CDN components onto the basic IP network.

This paper describes a typical IP network with CDN overlay and discusses how a CCN-based solution might improve performance as well as eliminate some of the problems encountered with modern CDN deployments. The focus is on the technical aspects rather than on the impacts this would have on the business of providing CDN services. Additionally, it is recognized that a move to a fundamentally different networking protocol cannot be made overnight. Thus, proposals are presented for incrementally introducing CCN into existing IP CDN networks, while attempting to recognize some of the benefits of CCN at each transitional step. Our suggestions are intended to limit the backwards-compatibility issues of network nodes, servers, and client devices that may not have yet been upgraded to support CCN.

As of the writing of this paper, the architectures proposed here have not been tested experimentally. The architectures are provided to guide the development of a more fine-grained system design, and we identify some areas where further research may be needed in order to validate the approach or to select from multiple options that are presented here. Moreover, this work is based heavily on one CDN technology and network topology, and on the current version of the CCN protocol (1.0 at the time of this writing). It is understood that other CDN technologies and topologies exist, and that the CCN protocol will likely evolve over time. Application of this work to other CDNs, and use of subsequent versions of the CCN protocol is left for future work.

2 Regional IP Network & CDN Overlay

To facilitate the description of the transition of a typical CDN to an Information-Centric architecture, this section describes a model IP network topology with CDN overlay. The model is reflective of the composition and scale of network design that would be typical in a mid-to-large size MSO with multiple subscriber regions and only a few content origination locations.

2.1 Regional IP Network

To begin, we introduce a model of a typical cable operator regional network, which provides Internet access to residents and businesses located within that region. Figure 1 shows a graphical representation of the model network. The diagram depicts several Core Regional Access Networks (CRAN), each of which provides Internet connectivity to a specific geographic region of the United States. For simplicity, only the “XYZ CRAN” shows all components of the cable network down to the individual subscriber, but it is assumed that these elements are also present in the other CRANs.

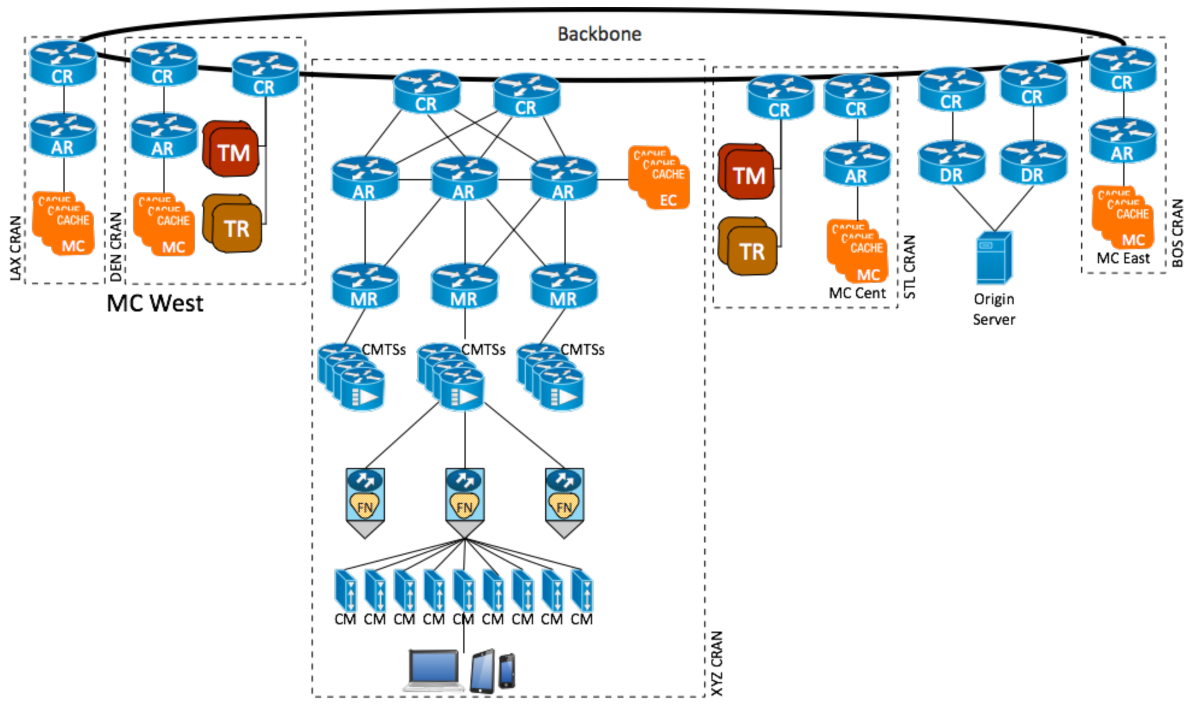


Figure 1 - Model Network Architecture

The portion of the network between the cable modems (CM) and the cable modem termination system (CMTS) is referred to as the Hybrid Fiber-Coaxial (HFC) network. The fiber portion of the HFC network connects each CMTS to multiple fiber nodes (FN), each of which serves a neighborhood (30-300 CMs). The fiber node is located at the neighborhood, and it converts between the optical signaling used on the fiber link, and the electrical signaling used on the coax feeders and drops that reach the individual households and CMs. The network capacity between the CMTS, through a single fiber node, to the 30-300 CMs is shared among that set of users. Ethernet frames are transported over the HFC network using the Data-Over-Cable Systems Interface Specification (DOCSIS®).

From this point, our model network consists of a hierarchy of increasingly powerful routers that aggregate traffic from the neighborhood fiber nodes to the network’s connection with the provider's backbone. At the first level, aggregating traffic from multiple CMTS routers are the municipal routers (MR). One level up from the MR is the aggregation router (AR), of which there are several to accommodate the traffic from a handful of MRs. Finally, the ARs are connected to two core routers (CR), which link this regional network to the long-haul links between regions. The AR nodes are linked with each other to provide redundancy and paths for intra-regional traffic.

Many ISPs operate networks in multiple regions of a country, sometimes separated by hundreds or thousands of miles. In the model network, we depict four regional networks (in addition to our generic "XYZ" network), all connected to the operator's Internet backbone; LAX, DEN, STL, and BOS.

2.2 CDN Overlay

A Content Delivery Network (CDN) is a collection of servers distributed throughout a network with the intent of reducing core network load and improving application performance by bringing popular content closer to the clients that are requesting it. In the model network, we use an open source CDN implementation developed by Comcast Cable called Traffic Control [1]. The Traffic Control CDN consists of three main applications; Traffic Router, Traffic Monitor, and Apache Traffic Server. There are, in fact, several other applications that are part of the Traffic Control suite, but only these three will be relevant in the description of our example CDN.

2.2.1 Cache Tiers and Groups

Caching servers in the Traffic Control system are divided into an implementation-specific hierarchy of cache "tiers". Requests for content that are not satisfied by a particular cache (cache-miss) are forwarded on to the next tier up the hierarchy (see Figure 2).

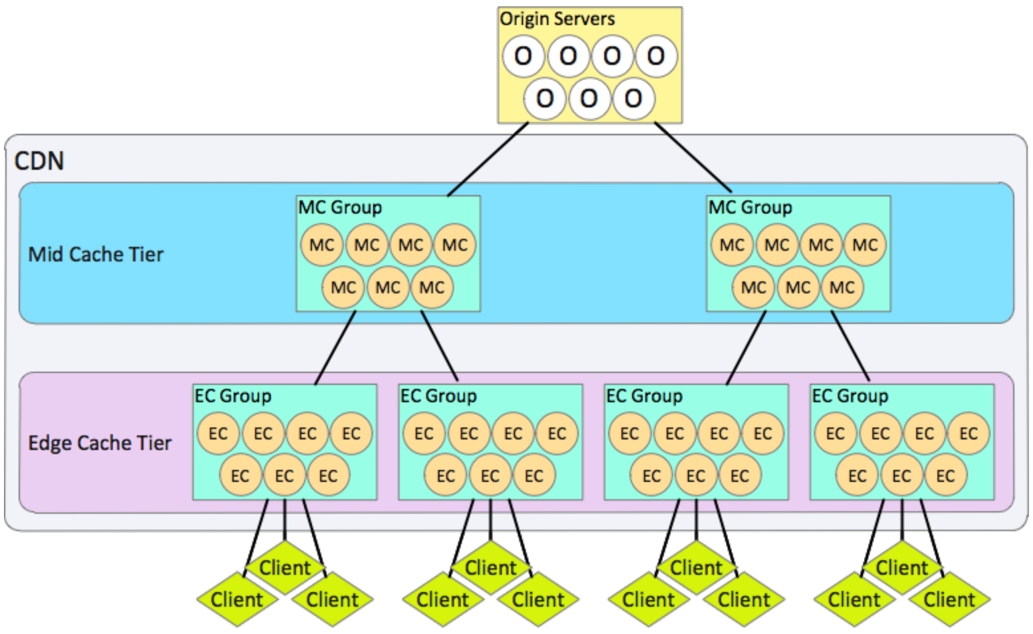


Figure 2 - CDN Tier Structure

Within each tier, collections of individual caches form a Cache Group, each of which is assigned to a geographic or topological region within the CDN. The model CDN has two tiers of caching, as depicted in Figure 1. The Edge Caches (EC) are closest to the client devices in the network and provide the fastest response times, and, consequently, the best user experience. Each CRAN has its own Edge Cache Group to serve clients located within that region, and the clients only directly interact with these ECs. While Figure 1 shows the entire EC Group being connected to a single AR in the CRAN, it is more likely that the ECs will be distributed across multiple ARs in the CRAN.

If a particular content item is not cached at the EC level, the requests are forwarded to the next tier caches, the Mid Caches (MC). Mid Caches are fewer in number than Edge Caches and are strategically located in a few CRANs to satisfy Edge Cache misses from a super-region (a set of CRANs served by a Mid Cache Group). Our model network contains three super-regions, and thus three Mid Cache Groups: West, Central, and East. Note that while an Edge Cache Group will have all of its member ECs located within the same CRAN, Mid Cache Groups may very well have MCs distributed across one or more CRANs (e.g., MC West has its MCs split across the LAX and DEN CRANs in the model network). Finally, if the requested content is not found in the MC Cache Group, the requests are sent to one or more Origin servers, which are always expected to have the content available. Every cache server in the CDN runs a copy of Apache Traffic Server (ATS) to handle HTTP content requests and proxying to the next-tier cache.

2.2.2 Traffic Router

The intelligence center of the CDN is the Traffic Router (TR) application. It serves as the central arbiter for all content routing decisions. The TR contains a list of all caches in the CDN and their positions in the caching hierarchy, called the Coverage Zone Map. The TR uses the information in the Coverage Zone Map, along with a geolocation database that maps client IP addresses to locations (CRANs), to identify the EC Group for a particular client.

In large scale CDNs, the resource requests for a population of clients are distributed across multiple caches in the Cache Group. This improves scalability by allowing the load to be spread among multiple caches. It also improves availability by allowing individual caches to be added to or removed from the Group without significant disruption in CDN performance. In order to spread request load across multiple caches in an effective way, it is important that all users' requests for a specific resource be directed to the same cache. This maximizes the probability of a cache hit and eliminates duplication of resources across multiple caches in the Group. This cache assignment is done using a hashing algorithm, whereby some piece of information attached to the request message and unique to the content being requested (asset ID, segment URL, etc. in the HTTP request header) is hashed, and the hash value is used to determine which cache in the Cache Group will handle that content request. This is typically done using a “consistent hash” algorithm [10], a specialized hashing technique that ensures that even when the number of caches in the Cache Group changes, only very limited remapping of content to caches needs to take place. In the Traffic Control CDN, the TR is responsible for performing the consistent hash operation to direct a client request to the appropriate first-tier cache, and each EC is responsible for the consistent hash operation when it selects an MC. In the case that the consistent hash operation is done by multiple devices (e.g., multiple TR instances that serve the CDN, or the multiple ECs in the EC Group), it is important that all of them utilize the identical mapping.

2.2.3 Content Routing

The Traffic Router is responsible for directing clients to the most optimal Edge Cache given the client’s location and the health of similarly located caches. There are two basic mechanisms used by the TR to accomplish this task: DNS Routing [4] and HTTP Routing [5]. For both approaches, it is necessary to make the TR instances be the authoritative DNS for all domains used to retrieve content (with the requests being load balanced across the pool of TRs).

For DNS Routing, the client issues a DNS query for the hostname associated with the content it wishes to retrieve. This DNS query is received by a local DNS server and forwarded to a TR instance. The TR then simply looks at the hostname in the DNS lookup and responds with the IP address of an EC selected based on that name and the IP address of the local DNS server that referred the request. The client makes its content request directly to the server address that was produced as a result of the DNS lookup. TRs can be configured with a list of hostname prefixes that should be handled using

DNS Routing. In DNS Routing, the TR would identify the zone of the client via the IP address of the local DNS server, and then use a consistent hash of the requested hostname to select a specific EC within the EC Group covering that zone. As a result, all clients in that zone would be directed to the same EC whenever they attempt to access content at the same hostname. For CDNs that handle diffuse requests across a large number of hostnames, this may be a sufficient method to distribute those requests across the EC pool.

The more sophisticated routing mechanism is HTTP Routing. In this scenario, the TR responds to the DNS query with its own IP address instead of a cache IP address. The client then issues the HTTP content requests directly to the TR, at which point the TR responds with an HTTP 302 redirect to the appropriate cache. This technique allows the TR access to more information regarding the request, such as the client's IP address and the HTTP request headers, including the full URL, which it can then use to make more intelligent cache selection decisions.

In a typical content request operation (using HTTP Routing) the CDN interaction works something like this. A client's HTTP request for a particular content file arrives at the TR, due to the TR resolving the URL hostname to its own IP address. The TR identifies the appropriate EC zone based on information provided in the request (IP address, HTTP headers, etc.). The TR then selects a specific EC in the EC Group covering that zone by executing the consistent hashing algorithm on some portion of the URL or other information passed in the request. The TR then responds with an HTTP 302 redirect message pointing to the selected EC. In response to the 302 message, the client issues a new request to that EC for the content. The result of HTTP Routing is that all clients within a zone are directed to the same EC whenever they attempt to access the same URL.

2.2.4 "Hot" Caches

While consistent hashing is highly beneficial for the reasons stated above, there is a side-effect which needs to be considered. When a particular cache ends up serving highly popular content, it can experience request frequency levels that exceed its ability to respond. In order to prevent this situation, another Traffic Control component called "Traffic Monitor" continuously monitors the load on each cache in a Cache Group and alerts the Traffic Routers when the load on an individual cache exceeds a configured threshold.

The simple solution to this situation is for the TRs to treat the "hot" cache as unavailable for new content requests until its load returns to a more acceptable level. By doing this, the consistent hashing algorithm redistributes across the remaining caches in the Cache Group all new requests that would have been assigned to the hot cache. Content requests that were already underway continue to utilize the hot cache, but as these transfers finish, the load naturally reduces until the cache can be marked as available again.

An effect of this solution is that the load (both in terms of requests per second, and in terms of assigned content objects) on all of the other caches in the Cache Group increases, which both reduces caching efficiency and increases the likelihood that another cache quickly becomes hot.

2.2.5 Traffic Monitor

The Traffic Monitor application is responsible for querying the health state of each cache and making that information available to the Traffic Router. Every ATS cache exposes a private URL and REST API that is used by the TM to access traffic statistics and other server status for that cache. Multiple TM servers run in parallel, and each builds their own independent view of the current CDN health. The TM consults a number of its peers and utilizes their responses, along with its independent view of cache health, to calculate a health value that is consistent with information gathered across the CDN. The TM periodically provides this information to the Traffic Routers.

A configuration called the Health Protocol instructs the TM of the circumstances under which a cache is to be marked as “unavailable”. A cache may become unavailable due to high network traffic loads, hardware or software anomalies, or other business-specific conditions that necessitate temporarily removing the cache from a group. As stated earlier, the consistent hashing algorithm prevents excessive re-mapping of content when Cache Group membership changes occur.

2.2.6 Request and Content Flow

To summarize the concepts introduced in the previous sections, let’s walk through the complete set of steps involved in performing a content request and returning the desired content back from the CDN. Assume that our client device has acquired the HTTP URL of a particular media file it wishes to download.

To initiate the request, the client device resolves the domain name component of the media URL. Our Traffic Router instance has been assigned as the authoritative DNS for the domain(s) associated with all content managed by the CDN. When multiple TR instances are present in the CDN, DNS requests are distributed across the instances in round-robin fashion. The TR resolves the domain name with its own IP address which causes the client to send its HTTP request directly to the TR.

Upon receiving the HTTP request, the TR selects the appropriate EC Group based on its Coverage Zone Map and the source IP address of the request. It then performs the consistent hash algorithm to select one of the available ECs within the EC Group and return an HTTP 302 Redirect response to the client, indicating the selected EC in the “Location” response header. In response, the client re-issues its HTTP request to the selected EC.

When the EC has the requested content available in its content store, it is returned directly to the client. Otherwise, the EC sends a request to the origin server, by way of a selected MC proxy. As stated earlier, each EC is assigned to an MC Group and the EC executes another consistent hashing process against some part of the request to select the individual MC within the MC Group. Upon receipt of the request, the MC examines its content store for the requested data and returns it, if found. If the content is not located with the MC, the cache forwards the request to the origin server, and the content is returned to the client, hairpinning its way back through the MC and the EC.

3 Content-Centric Networking

Information-Centric Networking (ICN) is a new approach to networking in which the addressable element of the protocol is the “information” itself, rather than the host endpoints, as is the case with IP today. Two main research groups are forging a path towards defining a standardized ICN protocol. The Content-Centric Networking (CCN) project [2] was launched at the Xerox Palo Alto Research Center (PARC) in 2009 and has an active body of participants from the research community and industry. The Named Data Networking (NDN) project [3], led by UCLA, is funded by a grant from the National Science Foundation (NSF) and has numerous research universities actively developing the protocol and contributing new innovation ideas. Both NDN and CCN protocols have a common root, and thus share many common design aspects. However, the protocols are, at this time, incompatible. For the purposes of this paper, we will focus on the CCN architecture and its associated semantics; however, much of what follows is true for NDN as well.

CCN/NDN protocols center around two primary packet types that facilitate a request/response communication paradigm. The “Interest” packet is an information request message, and it contains a unique name that identifies the resource being requested. The “Content Object” packet is the response message, and it contains not only the resource data, but also additional information to allow the requestor (or the network itself) to verify the provenance and integrity of that data. CCN actually defines two types of Content Objects. The first type is a simple data object as described earlier, and it carries the resource data as its payload. The second type is the “Manifest” Content Object. A Manifest is a composition of data names that describe the complete contents of a resource. Manifests can contain names that refer to simple Content Objects or even other Manifests. The Manifest is useful for transporting large data resources over typical network links that have constraints on the maximum packet size that can be transmitted. When a content producer publishes a resource under a particular data name, the underlying CCN network stack may actually respond to Interests for that name with a Manifest it generated, which splits the content into manageable chunks. The network stack in the consumer, upon receiving the Manifest, then generates Interest messages for all of the named data chunks listed in the Manifest to retrieve the entire resource.

In addition to the new fundamental packet types, CCN introduces changes to the operation of network routers (nodes). Most notably, CCN introduces on-path caching and stateful forwarding, two attributes that radically change the dynamics of the network. Network nodes, called Forwarders in the CCN architecture, receive and distribute packets via connection points called “Faces”. The term Face has been introduced to separate itself from the typical view of a physical or logical network interface. This is due to the fact that CCN Forwarders also exist on network endpoints and communicate with applications via Faces.

A key feature of CCN is the ability for every network node to incorporate a data cache, or Content Store. Content Objects passing through a Forwarder that are marked as “cacheable” by the publisher can be stored and served in response to future Interests for that same named data. The decision whether or not to cache a Content Object could be based on a policy set by the network operator, or it could simply cache all cacheable objects. When a Forwarder receives an Interest packet, it first checks its Content Store to see if it can respond directly. This functionality provides two main benefits for the consumer and the network operator. First, popular content can be cached in the network closer to the edge, with the result being that the consumer can expect to see lower average latency on requests and an overall improvement in experience. For the network operators, data requests satisfied at the network edge mean less traffic in the core network, which can lead to a reduction in overall infrastructure and interconnection costs. As a result, one should consider if caching in a CCN network could turn out to be a more elegant solution to content distribution than the typical CDN architecture overlaid on a traditional IP network.

When a forwarder receives an Interest packet that can't be satisfied by the Content Store, the forwarder needs to send that Interest on toward the appropriate producer. In order to do this, the CCN Forwarder maintains a Forwarding Information Base (FIB). The FIB is analogous to the routing table found in an IP router today. It contains a mapping of data name prefixes to egress Faces and is used to route Interest packets through the Forwarder. An example routing policy for a FIB would be a “longest prefix match”, in which the data name in the incoming interest is routed based on the longest matching prefix entry in the FIB.

When a forwarder forwards an Interest using its FIB, it also records the data name of the Interest and the Face on which the Interest arrived in its Pending Interest Table (PIT). The PIT is a form of ephemeral state maintained by CCN Forwarders, and is the embodiment of the stateful forwarding approach. The PIT state can be thought of as the “breadcrumb trail” left by an Interest packet as it traverses the network. As a Content Object moves through the network, Forwarders use the PIT table to determine the egress Face(s) on which the Content Object should be transmitted. To clean up the state left by the Interest packets, the entries in the PIT are deleted as the Content Object is delivered to the destination Face(s). CCN provides implicit multicast functionality when multiple Interests for the same data arrive at a Forwarder, since only a single PIT entry is created (referencing all relevant Faces), and thus a copy of the Content Object is delivered to all Faces identified by the entry in the PIT.

As mentioned previously, an important aspect of CCN is the concept of object security (as opposed to the connection security approach used in IP networks). Since the purpose of the CCN protocol is to satisfy requests for specific Content Objects, it is important that a client be able to validate that it received what it requested. Furthermore, since the requested Content Object could be delivered from any of multiple Content Stores (including those in routers that are not affiliated with the content producer), validating the server from which the client received the Content Object would be pointless. CCN therefore has a built-in (and always on) mechanism for the client to validate the provenance of a received Content Object. There are two forms that this mechanism can take. The first form is that the producer signs the Content Object with a digital signature, and includes the name of the producer certificate. If the client does not already have a copy of the producer certificate, it can retrieve and then validate the certificate using a traditional PKI approach. The second form is an optimization that reduces the production and validation complexity when a resource is made up of multiple Content Objects. In this form, each of the Content Objects is hashed (SHA-256), and the hash value is included in the manifest along with the Content Object name. The Manifest is then the only signed Content Object.

A further optimization to the second form of provenance validation has been proposed, and is currently being reviewed by the CCN community, under the moniker “nameless objects”. In this proposal, a Content Object can be identified by its SHA-256 hash value alone. The Interest that a consumer sends in order to retrieve a nameless object would include a non-unique name that is primarily used for Interest forwarding via the FIB, and CS/PIT matching would be done solely by the hash value.

4 CCN IMPACTS ON EXISTING NETWORKS

While CCN may have benefits over an HTTP-based CDN [6] [7], there are environments where an HTTP-based CDN is already deployed, and here the analysis is a bit more complex. In order for CCN to replace HTTP/IP, it needs to be introduced in a phased approach, minimizing disruption and cost, and accruing incremental benefits along the way. This introduces constraints and complexities at various phases of the transition that aren't present in the greenfield case. Whether these constraints and complexities completely erode the benefits is an open question.

Two key technologies are required to enable this transition: CCN-HTTP translation and CCN/IP tunneling. While these two technologies make the transition possible, they are the source of the additional complexity in the brownfield scenario.

CCN-HTTP translators come in two forms (which we refer to as HTTP->CCN and CCN->HTTP), and they allow an isolated CCN island to be deployed in an otherwise HTTP network. This allows a very flexible roll-out that can target equipment that is more easily converted to CCN support, or that provides early benefits. For example, an HTTP->CCN translator and a CCN->HTTP translator could bookend a bank of CCN cache nodes. To the rest of the network, this collection of devices would look and function as a large HTTP cache, and so could be introduced in place of such a cache.

The other transition technology, CCN/IP tunneling, is the expected method by which CCN will begin to be deployed in the majority of networks, not just in CDNs. Thus CCN initially becomes an overlay network over IP. Once multiple, isolated CCN islands are deployed, they can begin to be interconnected via CCN/IP tunneling, reducing the amount of protocol translation that needs to take place.

A combination of these two approaches is likely to provide the most attractive route to converting an HTTP CDN to a CCN delivery platform.

The key network elements in the CDN to upgrade are the caches and the routers. In our view, upgrading caches is likely to be more straightforward than upgrading routers, and has some near-term payoffs.

The conversion begins with isolated islands of CCN (via the use of translators), and evolves those into interconnected islands of CCN (via IP tunnels), then finally introduces native routing of CCN.

5 Benefits of CCN Relative to HTTP CDN

The integration of CCN into an existing content distribution network has the opportunity to provide a multitude of benefits when compared to the traditional HTTP approach. In the following sections, we highlight a few of these benefits and the ways in which they can improve both the performance and reliability of content delivery, and in doing so, we discuss the ways in which the benefits can be introduced into an existing HTTP CDN.

5.1 Simpler, Higher Performance Cache Implementation

Compared to an HTTP cache such as Apache Traffic Server, a CCN cache is an extremely simple device. While the HTTP cache needs to support managing TCP session state for all active connections (both incoming requests to the server portion and outgoing requests from the client portion due to cache misses), the CCN cache is totally connectionless and stateless. Also, the HTTP cache needs to support the HTTP server and HTTP client protocol and maintain state between them, whereas the CCN cache only parses a much simpler Interest message and then either returns a corresponding Content Object packet (unchanged) from its Content Store, or forwards the Interest message (again, unchanged). In many cases, the Interest name is expected to be in the form of a hash value, which could be used to optimize Content Store lookup. Further, the storage medium can be highly optimized based on the expected Content Object size. Since the vast majority of Content Objects will be the same size, there would be little wasted cache space.

5.2 Striping of Content and Cache Balance

As described above, large HTTP-based CDNs may split user requests across a set of caches in order to scale the caching platform performance. When doing so, user requests are distributed using a consistent hash on some information in the request that is common to all similar requests from other users. In some cases, the hashing is done on the URL of the content object being requested.

In an HTTP-based CDN that utilizes HTTP Routing, the TR needs to handle a potentially large number of client requests, which may be a limitation on the scalability of the TR. Handling a single client request likely involves multiple steps, as described in the Request and Content Flow section. In the case of retrieval of DASH video content, the initial resource request is for an MPD file. These MPD files commonly utilize relative pathnames as URLs for all of the subtending resources (so that the asset is portable across multiple server platforms without needing to re-create the MPD). The result is that the client will do one HTTP GET in order to retrieve the MPD, be redirected to an EC by the TR, and then utilize the same EC for all further resource requests for that asset. While this reduces the load on the TR, it also creates an asset-level granularity to the spread of content across the ECs in a Cache Group – a pretty coarse distribution given that an entire video asset can be multiple gigabytes, broken into thousands of resources.

When an EC has a cache miss, however, it can hash on the URL of the resource request in order to select an appropriate MC, and thus the MC content can be spread at the resource level, which could be 1000 times as fine grained as the asset level. However, even at the MC, large popular files (e.g., non-video) can result in one MC or another receiving a disproportionately greater load than the others.

As a result, the HTTP-based CDN has issues with caches becoming hot, and it requires a mechanism to prevent catastrophic breakdown, such as the one described previously.

In a CCN-based content distribution network, caches can be placed in-path between the clients and the origin servers. In order to scale cache performance, it might be necessary to deploy multiple CCN forwarders with content stores, similar to the Cache Group in the HTTP CDN. This could be done in multiple ways; e.g., by the use of a switch fabric to fan-out the Interest messages across the pool and then aggregate the forwarded (cache-miss) Interests on the other end as shown in Figure 3. Another approach would utilize the existing topology of an HTTP-based CDN, where caches are connected to a router (such as an Aggregation Router) in the regional network. Interests would be similarly fanned out to the set of caches, and cache-miss Interests would be returned to the Aggregation Router for forwarding toward the origin, as shown in Figure 4.

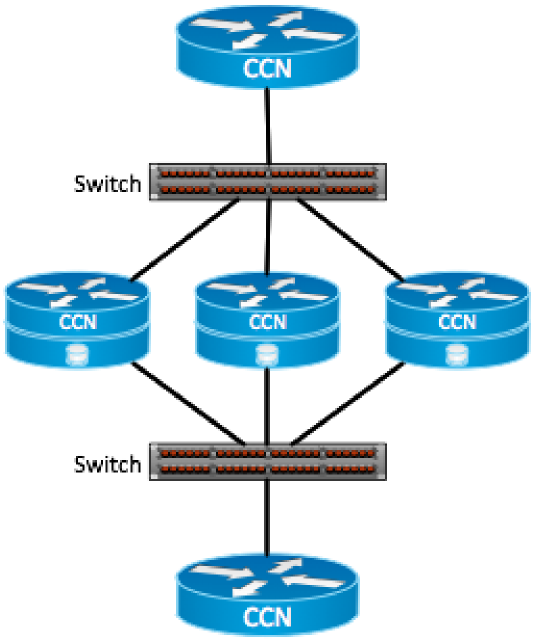


Figure 3 - In-Path Caching

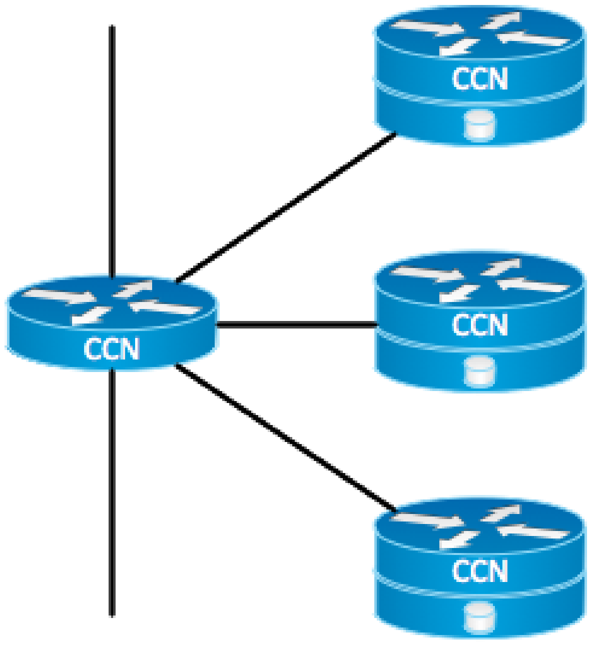


Figure 4 - Out-of-Path Caching

In both cases, the task of performing the consistent hashing operation would fall on the CCN forwarder immediately prior to the CCN Cache Group. This forwarder could be hashing based on the Interest name, and thus could stripe the Interests (and resulting content) at an extremely fine-grained level (at the content object level) across the Cache Group, essentially eliminating the hot cache problem.

Because this is a local operation, the hashing entity is not concerned with identifying the location of the client, and since the hot cache problem is eliminated, the traffic monitor infrastructure (and its resulting communication with the TRs) is no longer needed. The hashing entity will, however, still need to accommodate local caches going offline due to maintenance or failure, and new caches coming online. This could be a local action that takes place within the forwarder itself.

One ramification of performing this hashing operation at the Interest/Content Object level is that the hash operation needs to scale up at least two and possibly three orders of magnitude from how it is done in an HTTP CDN. Whether this introduces a scalability concern or not is an open question. However, we note that consistent hashing algorithms (such as the Highest Random Weight [8] algorithm) can be designed with low complexity, particularly in the case where the number of bins is small (as it is expected to be in the CDN case). Further, it may be reasonable, in certain instances where complexity is a strong concern, to use a simpler hash function that relaxes the consistency requirement (such as a modified jump consistent hash [9] or even a simple checksum). This may be considered a good tradeoff, particularly because the elimination of the hot cache problem means that caches will be added to or removed from the available cache pool much less often (i.e., only in the case of failure or scheduled service), and cache removals are likely to result in replacement caches being brought online in short order, perhaps obviating the need to redistribute requests.

In order to enable this striping benefit for a particular Cache Group, the caches need to natively support CCN, and both the south-facing and the north-facing faces need to support CCN. Minimally, this could be accomplished via an HTTP->CCN proxy south of the cache pool, and a CCN->HTTP proxy north of the cache pool.

5.2.1 Islands of CCN Caches

An EC Group in the example CDN could be targeted for conversion to CCN as follows. First, the subset of the ECs that are attached to a single AR are marked as “unavailable” in the TR, and then taken offline and upgraded to support CCN forwarding and Content Object caching. Second, a CCN->HTTP proxy is deployed north of this subset of ECs topologically, and the upgraded ECs are connected to that proxy (tunneling CCN over IP if needed). Third, an HTTP->CCN proxy is deployed to the south of the upgraded ECs topologically and the proxy is connected to the upgraded ECs (also using tunneling if needed). Finally, the DNS records for the updated ECs are changed to map to the HTTP->CCN proxy’s IP address, and those ECs are marked as “available” in the TR. Topologically, the EC Group might appear as shown in Figure 5, and an example implementation of this is shown in Figure 6, where an HTTP-CCN proxy appliance is inserted between the AR and the subset of attached caches.

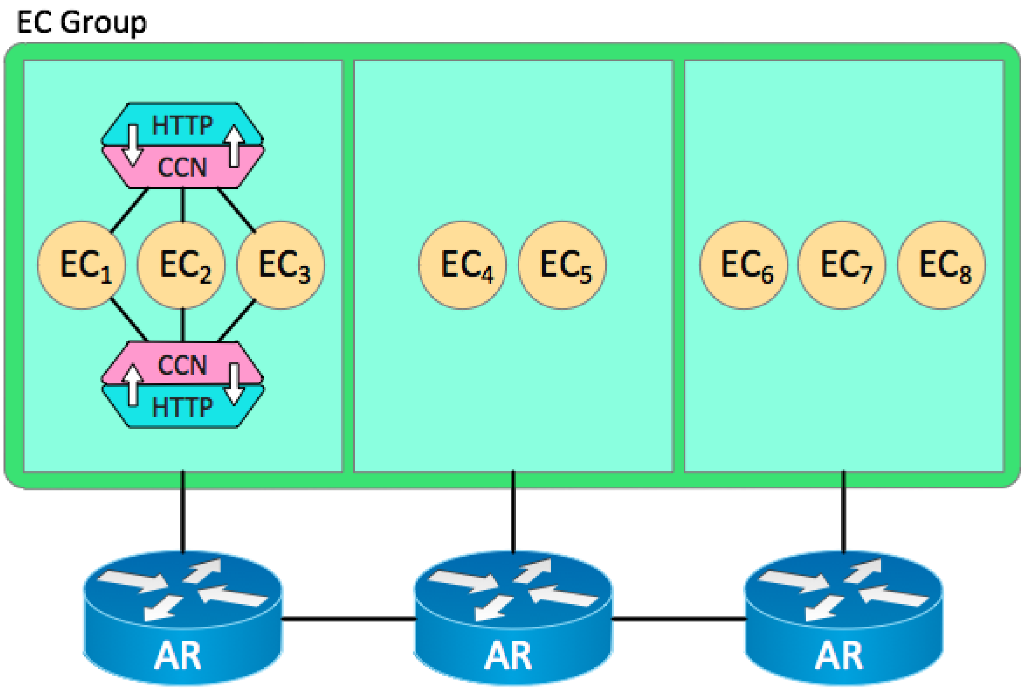
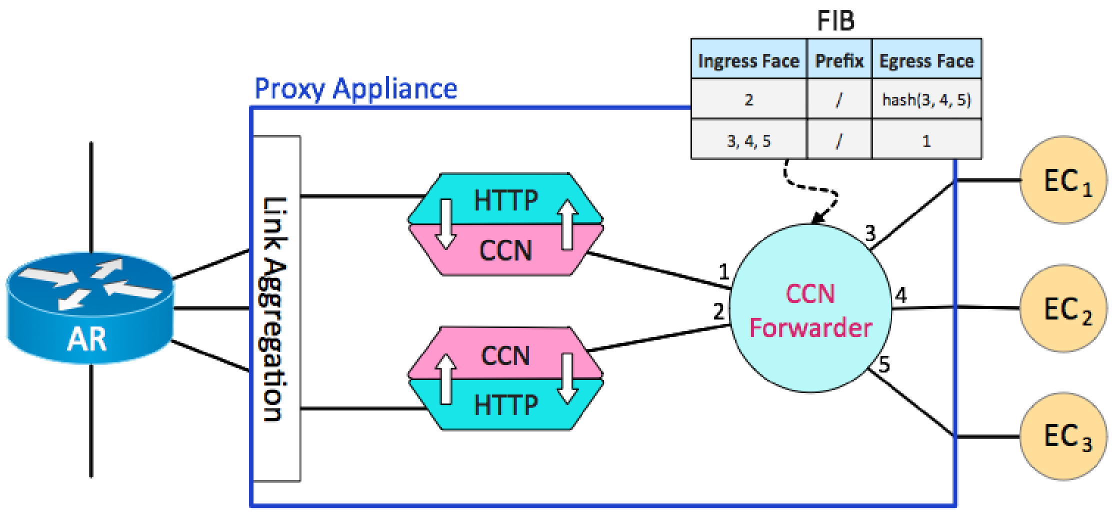


Figure 5 - CCN Island



Ingress Face	Prefix	Egress Face
2	/	hash(3, 4, 5)
3, 4, 5	/	1

Figure 6 - CCN Island Detail with Proxy Appliance

This introduction of CCN to a small portion of the network causes minimal disruption, and touches only a small number of components. In fact, the TR and the clients would be unaware that the transition occurred. The benefits, however, already begin to accrue. The probability of EC1, EC2 or EC3 becoming “hot” is significantly reduced, since the aggregate of the content requests that are assigned to those three ECs by the TR are now being striped at the Content Object level across the three caches, so the load to each one is now equal to one third of their aggregate traffic load.

Additionally, in the implementation shown in Figure 6, the proxy appliance brings with it some of the CCN Routing benefits (no hairpinning) that are described in Section 5.3, bringing further reductions in the load on the ECs. The proxy appliance also sets the stage for further evolution of the CDN, as described in more detail below.

Concerns with this arrangement might be the scalability of the proxy functions to handle the aggregate traffic load, or that it introduces a single point of failure in the system. This is an open area for research, but if it became an issue, one solution may be to deploy multiple devices as depicted in Figure 7.

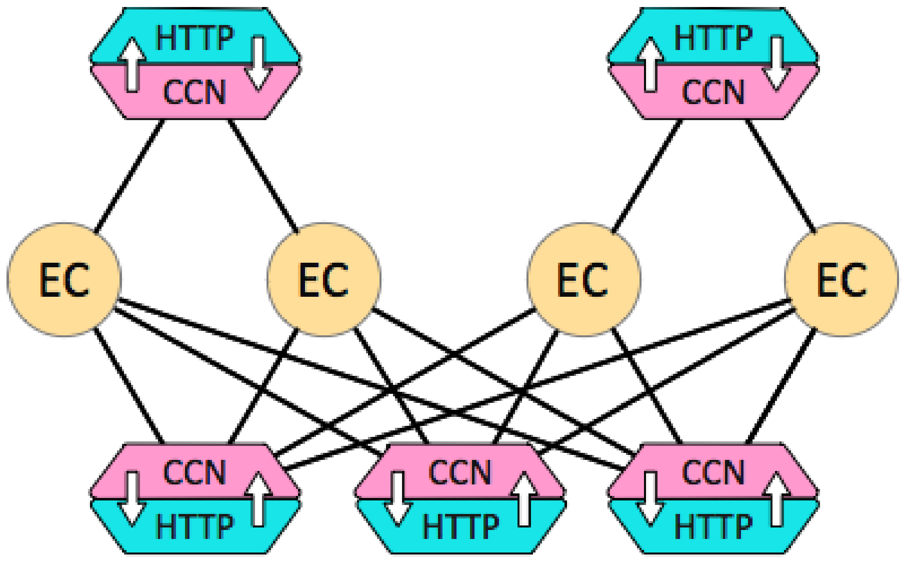


Figure 7 - Edge Cache Group with Multiple HTTP->CCN Proxies

5.2.2 Propagating CCN Beyond the Islands

Once all of the ECs in an EC Group have been updated as described above, it is possible to consider migrating the HTTP->CCN proxy function closer to the client devices and striping all content requests across the entire EC Group. For example, the HTTP->CCN proxy could be co-located with each MR and handle all of the content requests for the subtending clients (possibly by using DNS routing as described in the Content Routing section, or by using IP anycast addressing). Additionally, some residential gateway devices could be updated to support HTTP->CCN proxy functionality (e.g., as a transparent proxy). An upgraded EC group could be served by a mix of different HTTP->CCN proxies, some in customer gateways and some in the network. Each of these translators needs to be connected to the entire pool of upgraded ECs, likely via the use of IP tunneling. In the case of in-network proxies, static tunnels seem appropriate. In the case of placing proxies in gateway devices, it would be worth considering dynamic tunnel creation.

A similar approach as has been described for introducing CCN support in an EC Group can also be applied at an MC Group. Once an entire MC Group and all the EC Groups associated with it have been upgraded to support CCN, the CCN->HTTP proxy north of the ECs and the HTTP->CCN proxy south of the MCs can be shut off and replaced with simple CCN tunneling over IP.

In the example architecture utilizing a translation appliance as shown in Figure 6, once the HTTP->CCN and CCN->HTTP proxy functions have been disabled in that appliance, what is left is simply a set of tunnel termination points and a CCN router.

5.3 CCN Routing - No Hairpinning

Upgrading IP routers to natively support CCN forwarding allows further benefits to be accrued. In particular, it can reduce the complexity of IP tunneling, and minimize the amount of protocol translation in the network. Routers to which CCN caches are attached are particularly attractive candidates to be upgraded to support native CCN forwarding. While CCN supports in-path caching along the producer-to-consumer route, it may be some time before high-performance routers support large caches. As a result it is important to consider utilizing off-path caches in CCN. This also aligns with our example CDN topology, where ECs and MCs are attached to Aggregation Routers in the regional networks.

In this configuration, it is necessary to have content requests enter the router, be routed out to a selected cache, and then, in the case of a cache-miss, re-enter the router to be forwarded to a higher-layer cache or to the origin.

In one approach, the Aggregation Router has a FIB that takes into account ingress interface. Name prefixes, that are being handled by the caching infrastructure, would have FIB entries that point to the faces to which the caches are attached for the case when the Interest arrived from a consumer-facing face, and they would point toward the higher-layer cache (or origin) for the case when the Interest arrived from a cache-facing face. Thus an Interest from a consumer would be routed to the caches, and upon a cache-miss, the cache would simply forward the Interest back to the AR, where it would then be routed toward the higher-layer cache. As a result of this process, only a single PIT entry is created in the AR, regardless of whether there was a cache hit or a miss. In the case of a cache miss, the PIT entry would list two faces, one identifying the consumer's face, the other the cache's. Once the content object returns to the AR from a higher-layer cache or origin, it would be duplicated and sent to both the cache and the consumer.

In this approach, we propose that the cache has a specialized forwarder that functions as follows. The cache has a single face that represents its network connection to the AR. When an Interest arrives from that face, the cache examines its Content Store (as would any CCN forwarder), and responds with the cached object, if possible. Upon a cache miss, the forwarder simply returns the

Interest to the AR on the same face it arrived, and does not create a PIT entry. When the cache receives a content object, it simply inserts it into its Content Store.

By this process, the traffic on the link from the cache to the AR is significantly reduced in the case of a cache miss, consisting solely of Interest messages. The content returning from a higher-layer cache does not hairpin through the cache as it would in the HTTP case. Neglecting the size of Interest messages (since they are a small fraction of the Content Object size in the case of most content), this would be a bandwidth reduction equal to the cache hit ratio (i.e., if the cache hit ratio is 40%, the link bandwidth for CCN would be reduced by 40% compared to HTTP). The AR forwarding path would be reduced by a similar amount.

5.4 MULTICAST

Support for multicast distribution of Content Objects is built into the CCN protocol in the case that Interest messages from multiple clients are tightly correlated in time. In the case that Interests are spread out by the RTT (or less), multicast does not occur (but ideally cache hits would still provide a benefit in this case).

Since many access network links are costly and are shared media links with the ability to take advantage of multicast distribution (e.g., DOCSIS and PON), and since some applications (notably linear video) may benefit from it, it is worthwhile to consider how to encourage multicast distribution across those links.

One approach would be to enforce some strict time alignment between the hosts that are requesting the same linear video asset. This approach seems complex. The timing accuracy requirement could be reduced by delaying Content Object responses (thereby artificially increasing the RTT), but that only distributes the complexity.

Another approach is to deploy specialized CCN caching behavior in CPE gateway devices. In this approach, the gateway would identify the linear video stream being watched by a subtending client device, and would pre-emptively cache "future" Content Objects that are seen on the network for that same video stream. Depending on the amount of cache storage in the gateway, this could enable multicast distribution across a fairly wide range of time misalignment.

6 CCN Transition Components

6.1 HTTP->CCN proxy

The HTTP->CCN proxy function is the combination of an HTTP server and a CCN consumer. This is a straightforward function that receives an HTTP GET from a client, and generates the CCN Interest messages in order to retrieve the requested object from a CCN network.

In some cases, the CCN forwarder within this device needs to support being configured with multiple egress faces that are connected via CCN/IP tunnels to other CCN forwarders in the network (e.g., caches). In these cases, the forwarder implements consistent hashing on Interest name in order to distribute the content requests across the available caches.

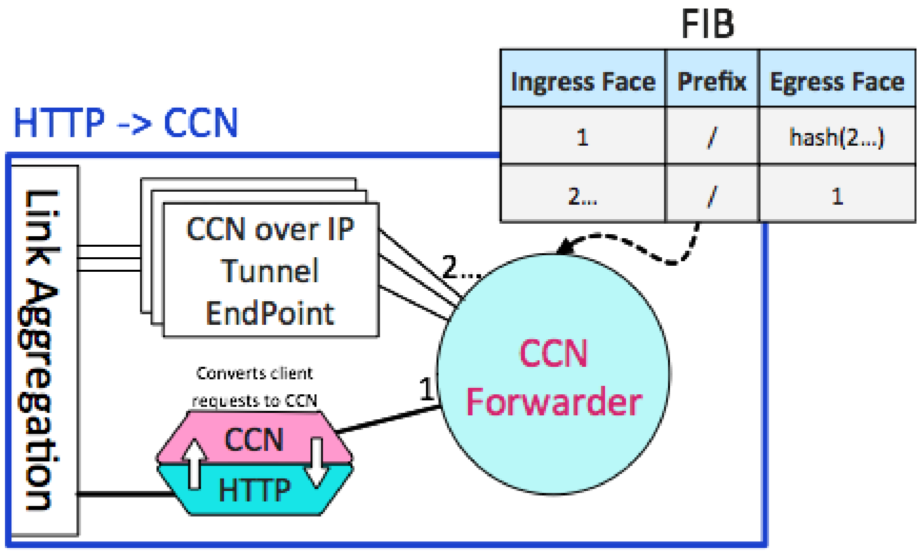


Figure 8 - HTTP->CCN Proxy

For performance and scalability reasons, it is expected that the HTTP server will cut-through the arriving Content Objects, rather than waiting to receive the entire resource.

This function could live in several different places in the network where it can have a stable IP address of which client devices can be made aware. One logical location for this function is in a home gateway device.

6.2 CCN->HTTP Proxy

The CCN->HTTP proxy function is the combination of a CCN publisher and an HTTP client. This function is a bit more complex than the HTTP->CCN proxy. Upon receiving a CCN Interest message from a consumer, this function needs to identify the corresponding HTTP resource, fetch the set of bytes that comprise the requested content object, and then return those bytes in an appropriately formatted content object. In the case, that the Interest was the initial Interest that is requesting a CCN Manifest, the proxy function will need to return a CCN Manifest for the resource.

We propose that an algorithmic mapping between HTTP resource name (i.e., URN) and CCN Name be established such that all translation proxy functions can generate one from the other.

There are a couple of options for handling the rest of the CCN->HTTP proxy functions. For the initial Interest case, where a Manifest is expected to be returned to the consumer, the CCN protocol is expected to standardize on a Manifest format that provides a list of the content object names and/or the content object hashes that make up the resource. The appropriate Manifest could be generated by the CCN->HTTP proxy each time an initial Interest is received, or it could be generated once and made available to all CCN->HTTP proxies via the origin server. In the case that the Manifest is generated by the CCN->HTTP proxy, the proxy would need to retrieve the resource size, calculate the number of content objects based on a pre-configured Content Object size, algorithmically generate the needed Content Object names, sign the Manifest, and then return it to the consumer. In this approach, it seems unreasonable to expect the CCN->HTTP proxy to generate content object hashes for each of the Content Objects, since doing so would require retrieving the entire resource. The implication of this is that hash-based Content Object validation would not be possible, and so the system would need to rely on signature-based validation of each Content Object. Thus, when the individual Interests for the Content Objects making up the resource are requested, the CCN->HTTP proxy would algorithmically determine an appropriate HTTP Range Request in order to retrieve the requested bytes, and then sign the Content Object as it is generated, allowing the consumer to then validate the signature. Alternatively, if the CDN is deployed in a closed network and Content Object signing creates a performance bottleneck, perhaps it would be attractive to disable Content Object validation.

Another approach would be to have the Manifest file generated once for each HTTP resource, and then made available (e.g., on the origin server). In this case, the Manifest generation function could calculate Content Object hashes for each of the Content Objects and include those in the Manifest, eliminating the need to have the CCN->HTTP proxy sign any Content Objects, and enabling the consumer to use the simpler hash-based validation. The Manifest file generated in this method could include additional information that is of use to the proxy, such as an explicit HTTP Range Request URL for each Content Object, eliminating the need for algorithmic generation of this URL by the CCN->HTTP proxy. In some cases with this approach it would be expected that the CCN->HTTP proxy would retain a cached copy of the information in this Manifest file, to avoid repeatedly fetching it.

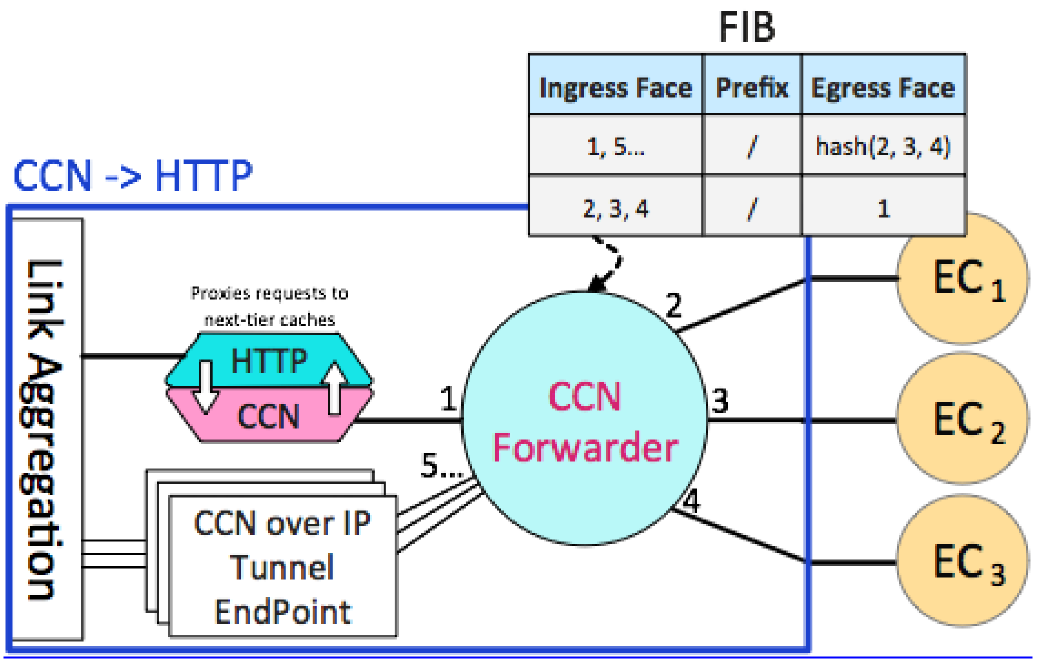

FIB
Ingress Face | Prefix | Egress Face
1, 5... | / | hash(2, 3, 4)
2, 3, 4 | / | 1
CCN -> HTTP
Link Aggregation
Proxies requests to next-tier caches
HTTP
CCN
CCN over IP Tunnel EndPoint
CCN Forwarder
1
2
3
4
5...
EC 1
EC 2
EC 3

Figure 9 - CCN->HTTP Proxy

7 UNSOLVED PROBLEMS

The previous sections proposed some ways in which CCN can be incrementally integrated into an existing CDN. However, the architectural description is by no means complete. There are several categories of problems that have yet to be addressed. Solutions to these problems will be required in any fully functional implementation.

Optimized CCN Router Implementation - Core and Edge IP router implementations have seen decades of development and optimization, whereas CCN router development is in its infancy, with only proof-of-concept implementations available at present. The topics of FIB/PIT sizing and memory bandwidth requirements, and content store implementation have been studied, but real development will require a stronger market signal to emerge.

Optimized CCN Cache Implementation - As discussed earlier, caching in CCN content stores is dramatically simpler than the equivalent in the HTTP world, and opportunities exist for very high performance caches. However, current implementations run in user-space and are explicitly referred to as non-optimized reference implementations.

Congestion Avoidance (CA) - In CCN networks, congestion avoidance is performed by the consumer rather than by the producer (server) as it is in TCP, and it consists of pacing the transmission of Interest messages in order to ensure that the delivery of a requested resource can make maximal fair use of the network. Current CCN CA algorithms are largely based on the TCP Congestion Avoidance algorithm, which presumes that the bandwidth-delay product of the network varies relatively slowly since, in large part, all data packets traverse the same path from server to client. In contrast, the Content Objects in a CCN resource request may be delivered from various producer or Content Store locations, and may take multiple paths to reach the consumer, so the concept of a bandwidth-delay product for the resource request as a whole can't be defined, making TCP-like CA a poor fit for CCN networks. This is an active area of research in the CCN community.

Cache control & semantics - Content distributors must have explicit controls for how and when content is cached in their networks. While the CCN protocol specifies an EXPIRY_TIME field in Content Objects, it lacks any further mechanisms for prioritization and on-demand purging of cached data. A network-wide control protocol needs to exist which allows for deleting objects (or otherwise making them unavailable) from network caches and updating cache policies. In a CCN world, where content is now an integral part of the network, it is likely that this functionality would be integrated into network management and monitoring systems.

Content Object size & fragmentation - The Content Object is an atomic unit of data that can be transferred across the CCN network, in many ways the corollary to a packet in an IP network. At the time of the writing of this paper, a maximum Content Object size (MTU) for CCN had not been specified, and there is some expectation that it may be larger (perhaps significantly larger) than the MTU in IP networks, thereby reducing the Interest message burden. In some cases, it may be necessary to fragment a large Content Object (either due to link technology limitations, or in order to reduce head-of-line blocking latency). While nothing has been specified yet with regard to this fragmentation mechanism, proposals have been made that it be done at the link level, so that it is transparent to the CCN forwarding plane.

Network Control -- Many of the routing policies introduced in this document are heavily dependent on the idea that all routers have a consistent understanding of the hashing policies to be used at each tier of the CDN. A change in the number of caches at any time or for any reason would require a simultaneous update of routing policies across several nodes in the network. A centralized network control mechanism would need to be in-place to perform these updates. This is likely a good application for SDN concepts.

CCN->HTTP Conversion -- In this paper we have described the need to perform conversion of CCN Interests to HTTP Requests, particularly in the early stages of the transition. We have proposed two methods to achieve this, but neither has been validated. We have some concerns that computational complexity of this function will make the earliest stages of the transition less attractive, and thus jeopardize the transition as a whole.

HTTP->CCN Conversion -- The conversion of HTTP requests into CCN Interests is simpler computationally than the inverse conversion. That said, we have not validated the approach experimentally, nor are we aware of implementations that could be utilized for experimentation.

Business Rules and Monetization – CDN may need to have features above and beyond the technical aspects of delivering content. Support for similar business-specific functionality and mechanisms for monetizing the distribution of partner content is needed in a CCN implementation as well.

8 CONCLUSION

CCN architectures are ideally suited to replace and improve upon the existing CDN overlay applications we have come to rely upon. Content, identified by a unique name, is returned to consumers by the closest node on the network that can provide it. In-network caching naturally facilitates the storage of popular content closer, topologically, to the clients that are requesting it. In addition, the request/response model used most widely for content delivery today (HTTP/S) is a fundamental part of the CCN paradigm in the form of Interest and Content Object packets. Finally, security implemented at the Content Object-level is a better fit for content distribution, and avoids many of the problems inherent in the end-to-end connection encryption approach (TLS).

It is possible to realize significant benefits through an incremental introduction of CCN into an existing CDN implementation. CCN->HTTP and HTTP->CCN proxies can create small “islands” of CCN functionality that can begin to eliminate some of the performance and management issues encountered in modern CDNs. By striping content across multiple caching routers at the Content Object-level, the “hot cache” problem is eliminated. Additionally, link utilization is reduced because content retrieved from higher-tier cache levels will not “hairpin” through the caches on the way to the client as it does today. It is important to note that these benefits can be achieved without making modifications to existing IP routers.

While the architecture proposed in this paper initially relies heavily on the CCN and HTTP proxy appliances, we are careful to point out that no proof-of-concept implementations have been developed at this time. It is unknown whether the processing requirements of these appliances will significantly impact the cost of the transition, and thus offset the benefits provided by CCN. More feasibility studies are required to determine the cost and scalability of these proxy devices in a high-volume CDN implementation.

APPENDIX A ALL CACHES SUPPORT CCN

By following the incremental integration approach defined in this paper, the CDN will eventually arrive at a deployment state in which all caches and origin servers have been upgraded to support CCN. At this point, there will be a mesh of CCN-over-IP tunnels between caches in adjacent tiers of the CDN. Consistent hashing policies among the network nodes will ensure even distribution of content across all caches at each tier. Additionally, only HTTP->CCN proxies will still be present in the system, as the need for the more complex CCN->HTTP proxies has been eliminated. Further, the proxy appliances that were deployed at each caching layer have now essentially become CCN routers. At this point, it may be feasible to subsume the actual caching functionality into those routers rather than keep the caches as distinct entities.

The HTTP->CCN proxies, however, will likely be required for much longer due to the need to support legacy client devices that will not be able to generate CCN network requests. The performance requirements of the HTTP->CCN proxies can be reduced by moving them closer to the network edge. As depicted in Figure 10, each “Service Group” (set of home gateway devices served by a single Municipal Router (MR)), can be assigned a proxy (connected to the MR) which will serve the requests of all legacy clients. The legacy clients can be directed to the closest proxy by the TR, or by using IP anycast. Using IP anycast, all proxies can be assigned the same network address and clients will be directed to their nearest proxy, topologically. In that case, the TR is no longer required.

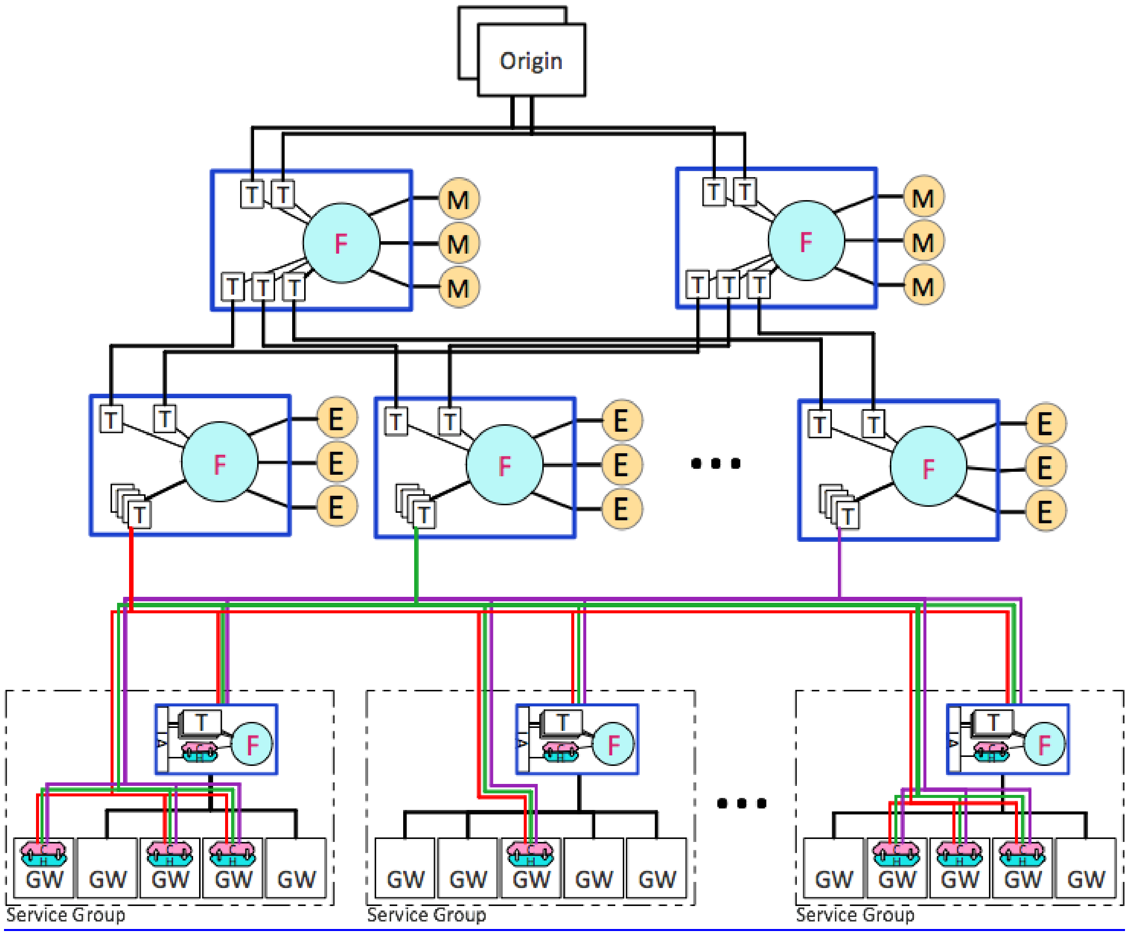


Figure 10 - Example CRAN with all Caches Upgraded to CCN

As a further step, the home gateways of individual subscribers can be updated with their own proxy implementations to serve devices within the home, reducing the dependency on the top-level proxy for that Service Group. The upgraded gateways would function as transparent proxies, intercepting traffic destined for the CDN and performing HTTP->CCN conversion implicitly. We can now start to introduce clients and applications that support native CCN-based content requests (over IP tunnels).

Throughout the description of this transition process, we have mostly avoided the discussion of updating routers to support CCN natively. Instead, we have chosen to represent CCN links as IP tunnels, and have shown the transformation of many of the proxy appliances into what amounts to simply CCN routers. In any plausible transition to CCN, IP routers will most certainly be upgraded to support both CCN and IP networks. Therefore, the primary area of concern will be in the performance implications of processing new network packet types. It is possible that existing routers may be able to support CCN through firmware updates, but it is unknown how well these platforms will scale without CCN-specific hardware support. The ideal solution may involve replacing the existing routers with new hardware that has been optimized for both CCN and IP network traffic.

APPENDIX B REFERENCES

[1] http://traffic-control-cdn.net/

[2] http://www.parc.com/services/focus-area/content-centric-networking

[3] http://named-data.net/

[4] http://traffic-control-cdn.net/docs/latest/overview/traffic_router.html#arrow-dns-content-routing

[5] http://traffic-control-cdn.net/docs/latest/overview/traffic_router.html#arrow-http-content-routing

[6] M. Mangili, F. Martignon, A. Capone, “A Comparative Study of Content-Centric and Content-Distribution Networks: Performance and Bounds”, IEEE Global Communications Conference (GLOBECOM), pp. 1403-1409, 2013

[7] D. Ma, Z. Chen, “Comparative Study of CCN and CDN”, IEEE INFOCOM Student Activities (Posters), pp. 169-170, 2014

[8] Thaler, David; Chinya Ravishankar. "A Name-Based Mapping Scheme for Rendezvous". University of Michigan Technical Report CSE-TR-316-96, http://www.eecs.umich.edu/techreports/cse/96/CSE-TR-316-96.pdf.

[9] J. Lamping, E. Veach, “A Fast, Minimal Memory, Consistent Hash Algorithm”, Google, http://arxiv.org/pdf/1406.2294.pdf

[10] Karger, D.; Lehman, E.; Leighton, T.; Panigrahy, R.; Levine, M.; Lewin, D. (1997). *Consistent Hashing and Random Trees: Distributed Caching Protocols for Relieving Hot Spots on the World Wide Web*. Proceedings of the Twenty-ninth Annual ACM Symposium on Theory of Computing. ACM Press New York, NY, USA. pp. 654–663. doi:10.1145/258533.258660

DISCLAIMER

ACKNOWLEDGMENTS

The authors would like to thank Jan Van Doorn (Comcast Cable) for his contributions to this paper.