Networking Named Content

October 13, 2009

Networking Named Content

Van Jacobson Diana K. Smetters James D. Thornton Michael F. Plass
Nicholas H. Briggs Rebecca L. Braynard
Palo Alto Research Center
Palo Alto, CA, USA
{van,smetters,jthornton,plass,briggs,rbraynar}@parc.com

ABSTRACT

Network use has evolved to be dominated by content distribution and retrieval, while networking technology still speaks only of connections between hosts. Accessing content and services requires mapping from the *what* that users care about to the network's *where*. We present *Content-Centric Networking* (CCN) which treats content as a primitive – decoupling location from identity, security and access, and retrieving content by name. Using new approaches to routing named content, derived heavily from IP, we can simultaneously achieve scalability, security and performance. We implemented our architecture's basic features and demonstrate resilience and performance with secure file downloads and VoIP calls.

1. INTRODUCTION

The engineering principles and architecture of today's Internet were created in the 1960s and '70s. The problem networking aimed to solve was *resource sharing* — remotely using scarce and expensive devices like card readers or high-speed tape drives or even supercomputers. The communication model that resulted is a conversation between exactly two machines, one wishing to use the resource and one providing access to it. Thus IP packets contain two identifiers (addresses), one for the source and one for the destination host, and almost all the traffic on the Internet consists of (TCP) conversations between pairs of hosts.

In the 50 years since the creation of packet networking, computers and their attachments have become cheap, ubiquitous commodities. The connectivity offered by the Internet and low storage costs enable access to a staggering amount of new content – 500 exabytes created in 2008 alone [13]. People value the Internet for *what* content it contains, but communication is still in terms of *where*.

We see a number of issues that affect users arising from this incompatibility between models.

- **Availability**: Fast, reliable content access requires awkward, pre-planned, application-specific mechanisms like CDNs and P2P networks, and/or imposes excessive bandwidth costs.

- **Security**: Trust in content is easily misplaced, relying on untrustworthy location and connection information.

To appear in CoNEXT 2009, December 2009, Rome, Italy

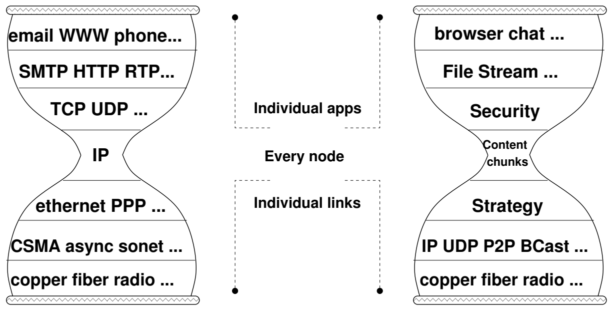


Figure 1: CCN moves the universal component of the network stack from IP to chunks of named content.

- **Location-dependence**: Mapping content to host locations complicates configuration as well as implementation of network services.

The direct, unified way to solve these problems is to replace *where* with *what*. Host-to-host conversations are a networking *abstraction* chosen to fit the problems of the '60s. We argue that *named data* is a better abstraction for today's communication problems than *named hosts*. We introduce *Content-Centric Networking* (CCN), a communications architecture built on named data. CCN has no notion of host at its lowest level – a packet "address" names content, not location. However, we preserve the design decisions that make TCP/IP simple, robust and scalable.

Figure 1 compares the IP and CCN protocol stacks. Most layers of the stack reflect bilateral agreements; *e.g.,* a layer 2 framing protocol is an agreement between the two ends of a physical link and a layer 4 transport protocol is an agreement between some producer and consumer. The only layer that requires universal agreement is layer 3, the network layer. Much of IP's success is due to the simplicity of its network layer (the IP packet - the thin 'waist' of the stack) and the weak demands it makes on layer 2, namely: stateless, unreliable, unordered, best-effort delivery. CCN's network layer (Section 3) is similar to IP's and makes fewer demands on layer 2, giving it many of the same attractive properties. Additionally, CCN can be layered over anything, including IP itself.

CCN departs from IP in a number of critical ways. Two of these, *strategy* and *security*, are shown as new layers in its protocol stack. CCN can take maximum advantage of multiple simultaneous connectivities (*e.g.,* ethernet and 3G and bluetooth and 802.11) due to its simpler relationship with layer 2. The *strategy* layer (Section 3.3) makes the fine-grained, dynamic optimization choices needed to best exploit multiple connectivities under changing conditions. CCN secures content itself (Section 5), rather than the connections over which it travels, thereby avoiding many of the host-based vulnerabilities that plague IP networking.

We describe the architecture and operation of CCN in Sections 2 through 5. In Section 6 we evaluate performance using our prototype implementation. Finally, in Sections 7 and 8, we discuss related work and conclude.

2. CCN NODE MODEL

CCN communication is driven by the consumers of data. There are two CCN packet types, `Interest` and `Data` (Figure 2). A consumer asks for content by broadcasting its interest over all available connectivity. Any node hearing the interest and having data that satisfies it can respond with a Data packet. Data is transmitted only in response to an Interest and consumes that Interest.[1] Since both Interest and Data identify the content being exchanged by name, multiple nodes interested in the same content can share transmissions over a broadcast medium using standard multicast suppression techniques [3].

Data 'satisfies' an Interest if the ContentName in the Interest packet is a prefix of the ContentName in the Data packet. CCN names are opaque, binary objects composed of an (explicitly specified) number of components (see Figure 4). Names are typically hierarchical so this prefix match is equivalent to saying that the Data packet is in the name subtree specified by the Interest packet (see Section 3.2). IP uses this convention to resolve the $\langle net, subnet, host\rangle$ hierarchical structure of IP addresses and experience has shown it allows for efficient, distributed hierarchical aggregation of routing and forwarding state while allowing for fast lookups.[2] One implication of this matching is that interests may be received for content that does not yet exist – allowing a publisher to generate that content on the fly in response to a particular query. Such *active names* allow CCN to transparently support a mix of statically cached and dynamically-generated content, as is common in today's Web. Name prefixes may also be context-dependent such as */ThisRoom/projector* to exchange information with the display projector in the current room or */Local/Friends* to exchange information with any friends in the local (broadcast) environment.[3]

The basic operation of a CCN node is very similar to an IP node: A packet arrives on a *face*, a longest-match look-up is done on its name, and then an action is performed based on the result of that lookup.[4] Figure 3 is a schematic of the core CCN packet forwarding engine. It has three main data structures: the FIB (Forwarding Information Base), Content Store (buffer memory) and PIT (Pending Interest Table).

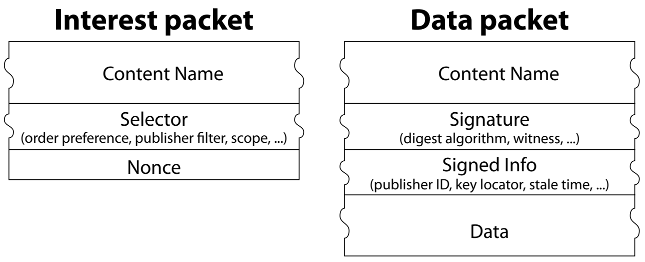

Interest packet
Content Name
Selector
(order preference, publisher filter, scope, ...)
Nonce
Data packet
Content Name
Signature
(digest algorithm, witness, ...)
Signed Info
(publisher ID, key locator, stale time, ...)
Data

Figure 2: CCN packet types

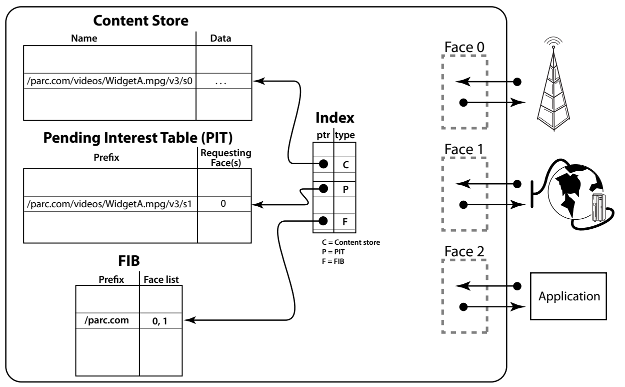


Figure 3: CCN forwarding engine model

The FIB is used to forward Interest packets toward potential source(s) of matching Data. It is almost identical to an IP FIB except it allows for a list of outgoing faces rather than a single one. This reflects the fact that CCN is not restricted to forwarding on a spanning tree. It allows multiple sources for data and can query them all in parallel.

The Content Store is the same as the buffer memory of an IP router but has a different replacement policy. Since each IP packet belongs to a single point-to-point conversation, it has no further value after being forwarded downstream. Thus IP 'forgets' about a packet and recycles its buffer immediately on forwarding completion (MRU replacement). CCN packets are idempotent, self-identifying and self-authenticating so each packet is potentially useful to many consumers (*e.g.,* many hosts reading the same newspaper or watching the same YouTube video). To maximize the probability of sharing, which minimizes upstream bandwidth demand and downstream latency, CCN remembers arriving Data packets as long as possible (LRU or LFU replacement).

The PIT keeps track of Interests forwarded upstream toward content source(s) so that returned Data can be sent downstream to its requester(s). In CCN, only Interest packets are routed and, as they propagate upstream toward potential Data sources, they leave a trail of 'bread crumbs' for a matching Data packet to follow back to the original requester(s). Each PIT entry is a bread crumb. PIT entries are erased as soon as they have been used to forward a matching Data packet (the Data 'consumes' the Interest). PIT entries for Interests that never find a matching Data are eventually timed out (a 'soft state' model — the consumer is responsible for re-expressing the interest if it still wants the Data).

When an Interest packet arrives on some face, a longest-match

[1]Interest and Data packets are thus one-for-one and maintain a strict flow balance. A similar flow balance between data and ack packets is what gives TCP its scalability and adaptability [20] but, unlike TCP, CCN's model works for many-to-many multipoint delivery (see Section 3.1).

[2]While CCN names are variable length and usually longer than IP addresses, they can be looked up as efficiently. The structure of an IP address is not explicit but instead implicitly specified by the contents of a node's forwarding table. Thus it is very difficult to apply modern $O(1)$ hashing techniques to IP lookups. Instead, $log(n)$ radix tree search (software) or parallel but expensive TCAMs (high end hardware) are typically used. Since the CCN name structure is explicit, ContentNames can easily be hashed for efficient lookup.

[3]This last example would use the explicit identity information created by CCN signing to allow friends to rendezvous via a fixed name rather than via complex enumeration or probing strategies. *i.e.,* the name says what they want to communicate and the signature says who they are in the context of the name, *e.g.,* 'a friend in the local environment'.

[4]We use the term *face* rather than *interface* because packets are not only forwarded over hardware network interfaces but also exchanged directly with application processes within a machine, as described in Section 6.

lookup is done on its ContentName. The index structure used for lookup is ordered so that a ContentStore match will be preferred over a PIT match which will be preferred over a FIB match.

Thus if there is already a Data packet in the ContentStore that matches the Interest, it will be sent out the face the Interest arrived on and the Interest will be discarded (since it was satisfied).

Otherwise, if there is an exact-match PIT entry the Interest's arrival face will be added to the PIT entry's RequestingFaces list and the Interest will be discarded. (An Interest in this data has already been sent upstream so all that needs to be done is to make sure that when the Data packet it solicits arrives, a copy of that packet will be sent out the face that the new Interest arrived on.)

Otherwise, if there is a matching FIB entry then the Interest needs to be sent upstream towards the data. The arrival face is removed from the face list of the FIB entry then, if the resulting list is not empty, the Interest is sent out all the faces that remain and a new PIT entry is created from the Interest and its arrival face.

If there is no match for the Interest it is discarded (this node does not have any matching data and does not know how to find any).

Data packet processing is relatively simple since Data is not routed but simply follows the chain of PIT entries back to the original requester(s). A longest-match lookup of a Data packet's ContentName is done upon arrival. A ContentStore match means the Data is a duplicate so it is discarded. A FIB match means there are no matching PIT entries so the Data is unsolicited and it is discarded.[5] A PIT match (there may be more than one) means the Data was solicited by Interest(s) sent by this node. The Data is (optionally) validated (see Section 5.1) then added to the ContentStore (*i.e.,* a C-type index entry is created to point to the Data packet). Then a list is created that is the union of the RequestingFaces list of each PIT match minus the arrival face of the Data packet. The Data packet is then sent out each face on this list.

Unlike IP's FIFO buffer model, the CCN Content Store model allows the node memory already required for stat muxing to simultaneously be used for transparent caching throughout the network. All nodes can provide caching, subject only to their independent resource availabilities and policies.

The multipoint nature of data retrieval by Interest provides flexibility to maintain communication in highly dynamic environments. Any node with access to multiple networks can serve as a content router between them. Using its cache, a mobile node may serve as the network medium between disconnected areas, or provide delayed connectivity over intermittent links. Thus CCN transport provides Disruption Tolerant Networking [11]. The Interest/Data exchange also functions whenever there is local connectivity. For example, two colleagues with laptops and ad-hoc wireless could continue to share corporate documents normally in an isolated location with no connectivity to the Internet or their organization.

3. TRANSPORT

CCN transport is designed to operate on top of unreliable packet delivery services, including the highly dynamic connectivity of mobile and ubiquitous computing. Thus Interests, Data, or both might be lost or damaged in transit, or requested data might be temporarily unavailable. To provide reliable, resilient delivery, CCN Interests that are not satisfied in some reasonable period of time must be retransmitted. Unlike TCP, CCN senders are stateless and the final consumer (the application that originated the initial Interest) is responsible for re-expressing unsatisfied Interests if it still wants the data. A receiver's strategy layer (see Figure 1) is responsible for retransmission on a particular face (since it knows the timeout for the upstream node(s) on the face) as well as selecting which and how many of the available communication interfaces to use for sending interests, how many unsatisfied interests should be allowed, the relative priority of different interests, etc.

Underlying packet networks might duplicate packets and CCN multipoint distribution may also cause duplication. All duplicate Data packets are discarded by the basic node mechanisms described in the preceding section. Though data cannot loop in CCN, Interests can loop and make it appear as if there is Interest on a face where no interest actually exists. To detect and prevent this, Interest packets contain a random *nonce* value so that duplicates received over different paths may be discarded (see Figure 2).

CCN Interests perform the same flow control and sequencing function as TCP ack packets. Flow control is described in the next section and sequencing in the following one. Since a node is guaranteed to see any Data resulting from its Interests, response time and rate can be directly measured and used to adaptively determine the best way to satisfy Interests in some prefix. This is described in the third section.

3.1 Reliability and Flow Control

One Interest retrieves at most one Data packet. This basic rule ensures that *flow balance* is maintained in the network and allows efficient communication between varied machines over networks of widely different speeds. Just as in TCP, however, it is possible to overlap data and requests. Multiple Interests may be issued at once, before Data arrives to consume the first. The Interests serve the role of window advertisements in TCP. A recipient can dynamically vary the window size by varying the Interests that it issues. We show the effect of such pipelining later in Section 6.2. Since CCN packets are independently named, the pipeline does not stall on a loss – the equivalent of TCP SACK is intrinsic.

In a large network, the end-to-end nature of TCP conversations means there are many points between sender and receiver where congestion can occur from conversation aggregation even though each conversation is operating in flow balance. The effect of this congestion is delay and packet loss. The TCP solution is for endpoints to dynamically adjust their window sizes to keep the aggregate traffic volume below the level where congestion occurs [20]. The need for this congestion control is a result of TCP's flow balance being end-to-end. In CCN, by contrast, all communication is local so there are no points between sender and receiver that are not involved in their balance. Since CCN flow balance is maintained at each hop, there is no need for additional techniques to control congestion in the middle of a path. This is not the same as hop-by-hop flow control, where backpressure between adjacent nodes is used to adjust resource sharing among continuous flows. CCN does not have FIFO queues between links but rather an LRU memory (the cache) which decouples the hop-by-hop feedback control loops and damps oscillations. (We will cover this topic in detail in a future paper.)

3.2 Sequencing

In a TCP conversation between hosts, data is identified by simple sequence numbers. CCN needs something more sophisticated because consumers are requesting individual pieces from large collections of data and many recipients may share the same Data packets. Locating and sharing data is facilitated by using hierarchical, aggregatable names that are at least partly meaningful to humans and

[5]'Unsolicited' Data can arise from malicious behavior, data arriving from multiple sources, or multiple paths from a single source. In the latter cases the first copy of the Data that arrives consumes the Interest so duplicate(s) will not find a PIT entry. In all cases the Data should be discarded since that preserves flow balance and helps guarantee stable operation under arbitrary load.

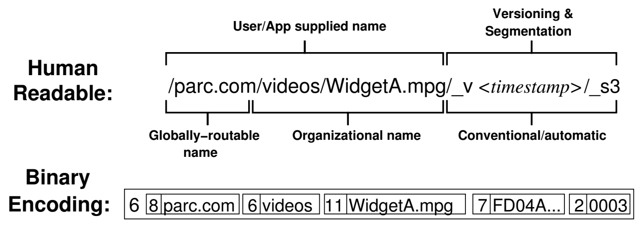

User/App supplied name
Versioning & Segmentation
Human Readable:
/parc.com/videos/WidgetA.mpg/_v <timestamp>/_s3
Globally-routable name
Organizational name
Conventional/automatic
Binary Encoding:
6 8 parc.com 6 videos 11 WidgetA.mpg 7 FD04A... 2 0003

Figure 4: Example Data name

reflect some organizational structure of their origin, rather than just the sequence in an ephemeral conversation. Despite this extra richness in CCN names, their transport function in Interests is exactly the same as that of sequence numbers in TCP ACKs: specifying the next Data the recipient requires.

Before explaining how the next Data is identified, we first describe the names in more detail. As mentioned, names are hierarchically structured so that an individual name is composed of a number of *components*. Each component is composed of a number of arbitrary octets – variable-length binary values that have no meaning to CCN transport. Names must be meaningful to some higher layer(s) in the stack to be useful, but the transport imposes no restrictions except the component structure. Binary encodings of integers or other complex values may be used directly without conversion to text for transmission. Name components may even be encrypted for privacy. For notational convenience, we present names like URIs with `/` characters separating components, as in Figure 4, but these delimiters are not part of the names and are not included in the packet encodings. This example illustrates the application-level conventions currently used to capture temporal evolution of the content (a version marker, `_v` encoded as `FD`, followed by an integer version number) and its segmentation (a segment marker, `_s` encoded as `00`, followed by an integer value which might be a block or byte number or the frame number of the first video frame in the packet). The final component of every Data packet name implicitly includes a SHA256 digest of the packet.[6]

An Interest can specify precisely what content is required but in most cases the full name of the next Data is not known so the consumer specifies it *relative* to something whose name is known. This is possible because the CCN name tree can be totally ordered (siblings are arranged in lexicographic order) thus relations like *next* and *previous* can be unambiguously interpreted by the CCN transport without any knowledge of name semantics.

For example, Figure 5 shows a portion of the name tree associated with Figure 4. An application that wants to display the most recent version of the video would express interest in '`/parc.com/videos/WidgetA.mpg` *RightmostChild*' which results in the highlighted traversal[7] and yields the first segment of the second version of the video. Once this was retrieved, the next segment could be obtained by sending an Interest containing its name with a *LeftmostRightSibling* annotation or by simply computing the `_s1` portion of the name since the segmentation rules are known (and determined) by the application.

As this example illustrates, the naming conventions for pieces of data within a collection can be designed to take advantage of the relative retrieval features of Interest packets and applications can discover available data through tree traversal. Although such naming conventions are not part of basic CCN transport, they are an important element of application design. We anticipate that a wide variety of reusable conventions will be standardized and implemented in shared libraries to provide applications with high-level abstractions such as files and media streams over CCN.

[6]The digest component is not transmitted since it is derivable. It exists so that an Interest or a link can unambiguously and exactly name any piece of content.

[7]The default traversal rule is *LeftmostChild*.

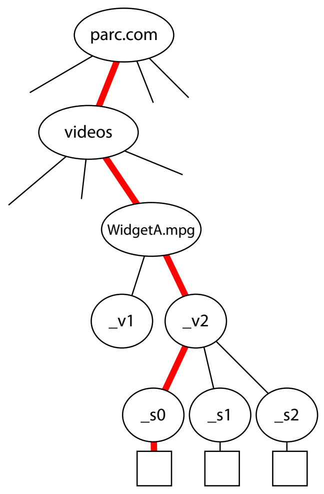


Figure 5: Name tree traversal

Interests, then, provide a form of restricted query mechanism over accessible content collections in a CCN, designed for efficient expression of what the receiver requires next. We do not have space to describe the details of the query options under development. It will be possible to restrict results by publisher, not just by collection, and to exclude content already obtained when simple ordering is insufficient. We are also developing higher level name discovery mechanisms that are more efficient for exploring large name subtrees when the content itself is not required.

3.3 Rich Connectivity, Mobility and Strategy

Machines today typically have multiple network interfaces and are increasingly mobile. Since IP is restricted to forwarding on spanning trees, it is difficult for IP to take advantage of more than one interface or adapt to the changes produced by rapid mobility. CCN packets cannot loop so CCN can take full advantage of multiple interfaces. CCN talks *about* data, not *to* nodes, so it does not need to obtain or bind a layer 3 identity (IP address) to a layer 2 identity such as a MAC address. Even when connectivity is rapidly changing, CCN can always exchange data as soon as it is physically possible to do so. Furthermore, since CCN Interests and Data are paired, each node gets fine grained, per-prefix, per-face performance information for adaptively choosing the 'best' face for forwarding Interests matching some prefix (see Section 6.3).[8]

As described in Section 2, CCN explicitly models multiple connectivity via per-FIB-entry face lists. Since there is no one-size-fits-all strategy for using multiple faces, the design intent is for each CCN FIB entry to contain a program, written for an abstract machine specialized to forwarding choices, that determines how to forward Interests. The 'instructions' for this machine should include a small subset of the normal load/store, arithmetic, and comparison operators plus *actions* that operate on sets of faces such as *sendToAll, sendToBest, markAsBest*, and *triggers* such as *inter-*

[8]In IP, route asymmetry generally makes it impossible for an interior node to learn if an interface or route is actually functioning since it only sees one side of a conversation.

estSatisfied, interestTimedOut, faceDown that can be used to invoke lists of actions when significant events occur. Faces will have an (open-ended) set of *attributes* such as *BroadcastCapable, isContentRouter, UsageBasedCharging, PeakUseLimited* that can be used to dynamically construct the sets for use by the actions.

These actions, triggers and attributes are collectively called the CCN *Strategy Layer* and the program in a FIB entry is the *strategy* for obtaining Data associated with the FIB's prefix. Our current default strategy is to send an Interest on all BroadcastCapable faces then, if there is no response, to try all the other faces in sequence. Thus data that is available in the local environment, such as on the instructor's computer in a lecture or a colleague's laptop or phone in a business meeting, will be obtained directly and only data that is not found locally will use the routing machinery.

The other faces in a FIB prefix entry are learned in a variety of ways. Sources of data, such as the repositories in Figure 6, arrange to receive Interests for the prefixes they service by doing a *Register* operation to the local CCN core. This creates local FIB entries for the registered prefixes that have the repository application's face in their face lists. The registered prefixes have optional flags that indicate if they should be advertised outside the local machine. Announcement agents read the registered prefix table on the local node (via normal CCN Interest-Data to a namespace reserved for local node communication) and advertise the flagged prefixes that meet their policy constraints (see Section 5.4). The advertisements might be via CCN (*e.g.,* the agent services Interests in `/local/CCN/registrations`), via standard IP Service Location protocols, or via CCN or IP routing (see Section 4).

4. ROUTING

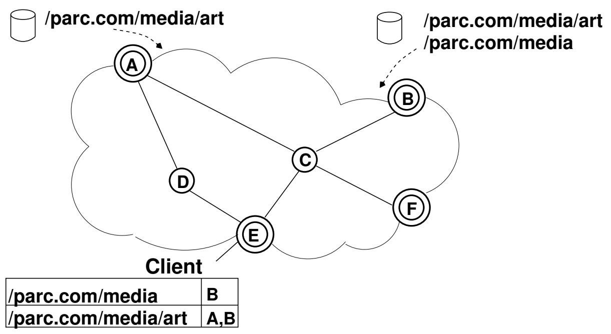
Routing has recently experienced a resurgence of research activity. Today there are a variety of interesting and effective candidate solutions for most routing problems. Any routing scheme that works well for IP should also work well for CCN, because CCN's forwarding model is a strict superset of the IP model with fewer restrictions (no restriction on multi-source, multi-destination to avoid looping) and the same semantics relevant to routing (hierarchical name aggregation with longest-match lookup). CCN provides an excellent vehicle to implement a routing protocol's transport: at the heart of most routing transport protocols is something very similar to CCN's information-oriented guided-diffusion flooding model since they have to function in the pre-topology phase of networking where peer identities and locations are not known. Since CCN provides a robust information security model (Section 5), using CCN as a routing transport can make routing infrastructure protection almost automatic.

To illustrate how CCN is mapped onto a routing scheme, the next section describes how CCN can be routed using unmodified Internet link-state IGPs (IS-IS or OSPF). This is intended both to show how CCN can use existing, conventional routing,[9] and to show that CCN is sufficiently compatible with IP that it can be deployed incrementally, using the existing infrastructure.

4.1 Link-state Intra-domain Routing

Intra-domain routing protocols provide a means for nodes to discover and describe their local connectivity ('adjacencies'), and to describe directly connected resources ('prefix announcements') [17, 16]. These two functions are orthogonal—one describes links in the graph while the other describes what is available at particular

[9]As opposed to more content-oriented routing such as Small-Worlds which also apply to CCN but have quite different implementation strategies.



/parc.com/media	B
/parc.com/media/art	A,B

Figure 6: Routing Interests to a domain's media content

nodes in the graph. It is common for these two functions to be performed in completely different information domains. For example, IS-IS [17] describes adjacencies in terms of IEEE 802.1 layer 2 MAC addresses but announces layer 3 IP4 and/or IP6 prefixes. As described in Section 2, IP forwarding and CCN forwarding are almost identical. They both use prefix-based longest match lookups (and use them for the same reason—hierarchical aggregation of detail) to find local neighbor(s) 'closer' to the identifier matched. Given the similarities of the two FIBs, one might suspect that the distributed routing machinery used to create IP FIBs might be easily adapted to create CCN FIBs. This is indeed the case.

CCN prefixes are very different from IP prefixes, so the main question is whether it is possible to express them in some particular routing protocol. Fortunately, both IS-IS and OSPF can describe directly connected resources via a general TLV ('type label value') scheme [18, 19] that is suitable for distributing CCN content prefixes. The specification says that unrecognized types should be ignored, which means that content routers, implementing the full CCN forwarding model, can be attached to an *existing* IS-IS or OSPF network with no modifications to the network or its routers. The content routers learn the physical network topology and announce their place in that topology via the adjacency protocol and flood their prefixes in prefix announcements using a CCN TLV.

For example, Figure 6 shows an IGP domain with some IP-only routers (single circles) and some IP+CCN routers. The media repository next to *A* is announcing (via a CCN broadcast in a local network management namespace) that it can serve Interests matching the prefix '/parc.com/media/art'. A routing application on *A* hears this announcement (since it has expressed interest in the namespace where such announcements are made), installs a local CCN FIB entry for the prefix pointing at the face where it heard the announcement, and packages the prefix into IGP LSA which is flooded to all nodes. When the routing application on *E*, for example, initially gets this LSA, it creates a CCN face to *A* then adds a prefix entry for '/parc.com/media/art' via that face to the local CCN FIB. When a different repository adjacent to *B* announces '/parc.com/media' and '/parc.com/media/art', *B* floods an IGP LSA for these two prefixes with the result that *E*'s CCN FIB is as shown in the figure. An interest in /parc.com/media/art/impressionist-history.mp4 expressed by a client adjacent to *E* will be forwarded to both *A* and *B*, who each forward it to their adjacent repository.

CCN dynamically constructs topologies that are close to optimal for both bandwidth and delay (*i.e.,* data goes only where there is interest, over the shortest path, and at most one copy of any piece of data goes over any link). But this delivery topology is clearly non-optimal since a client adjacent to *F* interested in the same movie would result in a second copy of the content crossing the *A*-*C* or *B*-*C* link. This happens when an incremental CCN deployment leaves some parts of the physical topology inaccessible to CCN (*C* is not a content router so it cannot cache). As soon as *C* gets the CCN software upgrade, *E* and *F* will forward their interests via it and the distribution will be optimal.

In the model described above, IGP LSA's are used as a transport for normal CCN messages which have full CCN content authentication, protection and policy annotation. Thus even though the IGP is not secure, the communication between CCN-capable nodes is secure. If all the nodes are evolved to being CCN-capable, the IGP topology infrastructure is automatically secured (see Section 5.1). The security of the externally originated prefix announcements is a function of the announcing protocol. CCN content prefixes, such as those announced by the media servers in Figure 6, are secured by CCN and have its robust trust model. IP prefixes announced from other IGPs or BGP would be untrusted.

There is a behavioral difference between IP and CCN in what happens when there are multiple announcements of the same prefix. In IP any particular node will send all matching traffic to exactly one of the announcers. In CCN all nodes send all matching interests to all of the announcers. This arises from a semantic difference: An IP prefix announcement from some IGP router says "all the hosts with this prefix can be reached via me". The equivalent announcement from a CCN router says "some of the content with this prefix can be reached via me". Since IP has no way of detecting loops at the content level, it is forced to construct loop-free forwarding topologies, *i.e.,* a sink tree rooted at the destination. Since a tree has a single path between any two nodes, an IP FIB has only one slot for 'outgoing interface'. So all the hosts associated with a prefix have to be reachable via the node announcing a prefix because all traffic matching the prefix will be sent to that node. Since CCN packets cannot loop, a prefix announcement does not have to mean that the node is adjacent to all the content covered by the prefix and CCN FIBs are set up to forward Interests to all the nodes that announce the prefix. This semantic difference can be accommodated without changing the IGP because it is an implementation change, not a protocol change. IP has to compute a spanning tree from prefix announcements and CCN does not, but this computation is done where the information is used, not where it is produced, so both protocols receive complete information.[10]

4.2 Inter-domain Routing

Once a few customers of an ISP start to use CCN, it is in the ISP's best interest to deploy content router(s) to reduce peering costs (only one copy of any piece of content needs to cross an inter-provider peering link, independent of how many customers request it) while lowering customers' average latency (all but the first copy come from the ISP's local content store). Thus there is an edge-driven, bottom-up incentive structure to grow CCN once it reaches some base deployment threshold. Since customers are directly connected to their ISP, it is trivial for them to learn about the ISP's content router via a service discovery protocol run over the customer-ISP peering link(s). This protocol could use CCN-based registration announcements (Section 3.3) for a 'wildcard' (0-component) prefix. Or it could use Anycast (*e.g.,* all content routers have IP address 10.0.96.95), DNS convention (*e.g.,* all content routers are named *ccn.isp.net*), DNS SRV[14], SLP[15], etc., and none of these options require any inter-domain distribution of content prefixes.

[10]Strictly speaking, this statement is true for link-state IGPs like IS-IS or OSPF but not for distance vector IGPs like RIP or EIGRP. The production of their routing announcement involves a Bellman-Ford calculation that presupposes spanning trees and suppresses information on alternatives. Such an IGP would require small modifications to the scheme described here.

The central problem with this type of bottom-up deployment is to bridge the gap between domains that have content routers but are separated by ISP(s) that do not. For example, a content router at *parc.com* would like to obtain content with the prefix *mit.edu* and there are no content routers between PARC and MIT. Using the prefix in a (heuristic) DNS lookup to locate the IP address of content server(s) at MIT (either via a *_ccn._udp.mit.edu* SRV lookup and/or a *ccn.mit.edu* address lookup) works perfectly well to automatically and on-demand build a UDP-tunneled 'face' connecting the content routers. But this scheme does not work as well if the gap is not at the edges. If PARC and MIT's ISPs both support content routing but they are connected via ISPs that do not, there is no way for PARC's ISP to learn of the relevant content router in MIT's ISP so it will forward Interests directly to MIT. Thus without additional mechanism, ISP routers benefit inbound content (content requested by their customers) but not outbound (content created by their customers). This partially negates a major long term CCN advantage of making traffic near the root of a content distribution tree independent of the popularity of the content; today that traffic grows linearly with the popularity (Section 6.2).

This problem can be fixed by integrating domain-level content prefixes into BGP. Current BGP inter-domain routing has the equivalent of the IGP TLV mechanism that would allow domains to advertise their customer's content prefixes. The BGP AS-path information also lets each domain construct a topology map equivalent to the one constructed in the IGP case, but at the Autonomous System (AS) rather than network prefix level. This map is functionally equivalent to the IGP case (one learns which domains serve Interests in some prefix and what is the closest CCN-capable domain on the paths to those domains) so the same algorithms apply.

5. CONTENT-BASED SECURITY

CCN is built on the notion of *content-based security*: protection and trust travel with the content itself, rather than being a property of the connections over which it travels. In CCN, *all* content is authenticated with digital signatures, and private content is protected with encryption. This is a critical enabler for CCN's dynamic content-caching capabilities – if you are to retrieve content from the closest available copy, you must be able to *validate* the content you get. Current IP networks trust content based on where (from what host) and how (over what sort of pipe) it was obtained; clients must therefore retrieve content directly from the original source to trust it. Embodying security in content, not hosts, reduces the trust we must place in network intermediaries, opening the network to wide participation. In this section, we give an overview of CCN's core security design, and highlight novel aspects of its security processing. Detailed analysis of the CCN security model, including topics like revocation, will be the subject of a separate paper; additional background and motivation are described in [34].

5.1 Content Validation

CCN authenticates the *binding* between names and content; the signature in each CCN data packet (Figure 2) is over the name, the content, and a small amount of supporting data useful in signature verification ("signed info" in Figure 2). This allows content publishers to securely bind arbitrary names to content. In contrast, many previous approaches require names to be *self-certifying* to securely name content (*e.g.,* by using the cryptographic digest of the content as its name [26, 28, 12, 9]). The ability to directly, and securely use user- or application-meaningful names enhances usability and eases transport. Systems without it require an "indirection infrastructure" [4, 6] to map from the names humans care about to secure, opaque, self-certifying names. The security of the resulting system is then limited to the security of the (often unsecured) indirection infrastructure.

CCN data is *publicly authenticatable* – per-packet signatures are standard public key signatures, and anyone, not just the endpoints of a communication stream, can verify that a name-content binding was signed by a particular key. The signature algorithm used is selected by the content publisher from a large fixed set, and chosen to meet the performance requirements of that particular data – *e.g.,* to minimize the size of the verification data, or the latency or computational cost of signature generation or verification. Though Data packets are designed to be individually verifiable, the computational cost of signature generation may be amortized across multiple packets through the use of aggregation techniques such as Merkle Hash Trees [27].

Each signed CCN Data packet contains information to enable retrieval of the public key necessary to verify it. Its supporting information includes the cryptographic digest, or fingerprint, of that public key, as a shorthand identifier for the publisher, and to enable fast retrieval of that key from a local cache. It also includes a *key locator*, which indicates where that key can be obtained; this can contain the key itself, or a CCN name to retrieve the key.

At the lowest layers, CCN content validation is purely *syntactic* – it simply verifies that content was signed by the key it purports (the key whose fingerprint is specified as the content publisher). It does not attach any real-world meaning to that key – who it belongs to, or if a signature by that key indicates whether or not the user should trust this particular piece of content. Even this minimal verification can be surprisingly useful, particularly in defending against many types of network attack. For example, it allows content consumers to request content by publisher as well as by name, and to get the content they intend in the face of spurious or malicious data. CCN content routers may choose to verify all, some or none of the Data they handle, as their resources allow. They may also dynamically adapt, verifying more data in response to detected attack.

5.2 Managing Trust

Although CCN moves data in a peer-to-peer fashion, it provides *end-to-end* security between content publisher and content consumer. CCN content consumers must determine whether received content is acceptable, or *trustworthy*. CCN's notion of trust is *contextual*, *i.e.,* narrowly determined in the context of particular content and the purpose for which it will be used. For example, one might require a legal document be signed by someone authorized by the courts, and a blog post only be signed by the same person who signed the other entries in that blog – and perhaps not even that. This is more flexible and easier to use than attempts to mandate a one-size-fits-all approach to trust, for example marking publishers as uniformly "good" or "bad".

The basic primitive of content-based security – authenticated bindings from names to content – can be used to implement mechanisms for establishing higher-level trust. CCN's signed bindings between names and content act in essence to *certify* that content. When that name refers to an individual or organization, and that content is a public key, the result is essentially a digital certificate. This allows CCN to easily support traditional mechanisms for establishing trust in keys. More interestingly, by allowing content to securely link, or refer, to other content we can allow content to certify other content. This provides a powerful mechanism by which we can leverage trust in a small number of keys into trust in a large forest of interconnected content.

5.2.1 Trusting Keys

Application-level CCN consumers must solve traditional key management problems – associating public keys with individuals and organizations, as it is these real-world identities that largely determine who is an acceptable signer for a given piece of content. CCN simplifies this task in several ways: first, it directly addresses the practical problem of merely obtaining the keys necessary to verify a piece of content. Keys are just another type of CCN Data, and simple naming *conventions* enable them to be easily found.

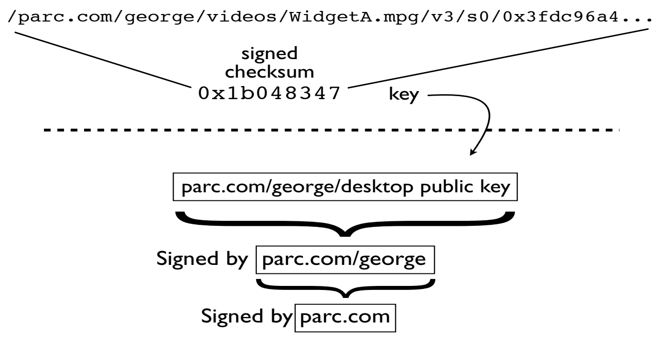

/parc.com/george/videos/WidgetA.mpg/v3/s0/0x3fdc96a4...
signed checksum
0x1b048347
key
parc.com/george/desktop public key
Signed by parc.com/george
Signed by parc.com

Figure 7: CCN trust establishment can associate content namespaces with publisher keys.

Second, as mentioned above, merely publishing a key as CCN content effectively generates a certificate for it – binding a (CCN) name to that key as authenticated by the signer (publisher). This building block can be used to represent arbitrary trust relationships between keys directly in CCN – from simple trees, as in a traditional certificate-based public key infrastructure (PKI), to the arbitrary graphs used by the PGP Web of Trust.

Third, CCN does not mandate a one-size-fits-all trust model. Trust is between publishers and content consumers, and what is appropriate for one application might not be appropriate for another. Users are free to reuse existing models (*e.g.,* PKI) for establishing trust in keys, or to define new ones more appropriate to CCN.

A model particularly suited to CCN is that of SDSI/SPKI [32, 10, 2]. In this model keys are mapped to identities via locally-controlled *namespaces*; *e.g.,* the members of an organization might be recognized because their keys are certified by the organization itself, not because they are validated by some source of external, third-party trust (*e.g.,* Verisign). Knowing the key for `parc.com`, we can then authenticate the keys of its employees. More powerfully, if we know and trust one of `parc.com`'s employees, we might look in his SDSI namespace for the identity, and key, of `parc.com`. Starting from a small number of public keys authenticated using a variety of user-friendly mechanisms (*e.g.,* personal contact, organizational membership, public experience [30, 37]), one can use SDSI's model to infer trust in a large number of publishers.

We can map SDSI identities directly into CCN names, and express SDSI trust relationships directly in CCN content. Such namespaces make up a *forest-of-trees* – a content consumer might trust that they have the right key for `parc.com` (or for `/parc.com/george`) for any number of reasons from direct experience (*e.g.,* they are a PARC employee), to information provided by friends, to its presence in a trusted directory of keys. It is not required, or even expected, that all such trees will be joined in a single (or small number of) root(s) as happens in traditional global or commercial PKIs. Most notably, it is the *consumer* who decides why they trust a particular key, using many types of information, not the publisher in obtaining a certificate from a particular vendor.

Further, by organizing content in terms of hierarchical *name-*

spaces, CCN allows signing policy, and even keys, to attach to particular content names; authorization at one level of a content namespace is given by a signature from a key at a higher level. Figure 7 shows the key for `parc.com` authorizing that of user `george`, who then authorizes the key for his `desktop` computer. These trust statements, represented as CCN data, help a consumer evaluate whether or not he is an acceptable publisher of `WidgetA.mpg` in the `parc.com/george` namespace.

5.2.2 Evidence-Based Security

We can add to our notion of structured names a representation of *secure reference*, much like a trusted hyperlink or bookmark. One CCN content item can refer to another (the link target) not only by the target's name, but also by the cryptographic digest of its contents (forming effectively a *self-certifying name* [26, 28, 12, 9]), or by the identity (key) of its publisher [9, 31, 25, 24]). Such references can be used to express *delegation*, saying that the publisher P of a link named N with target (N', P') intends the name N to refer to whatever publisher P' refers to by target name N'.

Such references can express traditional forms of delegation, but they can also be used to build up a network of trust in content – individual signed pieces of content effectively certify the other pieces of content they (securely) refer to. For example, having decided to trust content *A*, say a web page, users may automatically trust the content *A* securely links to – *e.g.,* its images, ads, source material and so on, without additional management or configuration overhead. That trust is very fine-grained – those materials are only considered valid within the context of *A*.

Each piece of content the user encounters also acts as a potential piece of *evidence* as to the validity of content it refers to. If many publishers that we trust all say that they believe in P''s value for N', we are much more likely to believe it as well. If an attacker subverts a single publisher, *e.g.* obtaining the key for P'' and using it to forge a malicious value for N' the attack will fail, as the preponderance of the evidence will still point to the correct value. With each piece of additional signed support for trustworthy content, it becomes harder and harder for an attack to succeed, as an attacker simply cannot subvert all of the available evidence.

5.3 Content Protection and Access Control

The primary means of controlling access to CCN content is encryption. CCN does not require trusted servers or directories to enforce access control policies; no matter who stumbles across private content, only authorized users are able to decrypt it.

Encryption of content, or even names or name components, is completely transparent to the network – to CCN, it is all just named binary data (though efficient routing and data sequencing may require that some name components remain in the clear). Decryption keys can be distributed along with their content, as CCN Data blocks. Name conventions, encapsulated in programmer-friendly libraries, can make it easy to find the decryption key necessary for an authorized user to decrypt a given piece of content. CCN does not mandate any particular encryption or key distribution scheme – arbitrary, application-appropriate access control models can be implemented simply by choosing how to encode and distribute decryption keys for particular content.

5.4 Network Security and Policy Enforcement

CCN's design protects it from many classes of network attack. Authenticating all content, including routing and policy information, prevents data from being spoofed or tampered with. The fact that CCN messages can talk only *about* content, and simply cannot talk *to* hosts makes it very difficult to send malicious packets to a particular target. To be effective, attacks against a CCN must focus on denial of service: "hiding" legitimate content (*e.g.,* simply not returning an available later version), or "drowning" it – preventing its delivery by overwhelming it in a sea of spurious packets.

To ensure they get the content they want in the face of potential spurious alternatives, consumers can place constraints on the publishers whose content can satisfy their Interests. Available constraints attempt to strike a balance between network efficiency and content consumer ease of use. At a minimum, consumers (or library software acting on their behalf) can specify the specific publisher (public key) they want to have signed their desired content, or a key that must have signed (certified) the key of the content publisher. This level of indirection avoids the brittleness of systems that require content consumers to know what specific key signed the content they want *a priori*.

CCN incorporates a number of mechanisms to prevent excessive forwarding of unwanted traffic. Flow balance between Interests and Data prevents brute force denial of service by Data flooding over anything beyond the local link. Flow balance operates in a hop-by-hop fashion. Only the number of Data packets requested by downstream Interests will ever be forwarded across a given link, no matter how many are provided. Even if an Interest is forwarded across many networks in search of matching Data (unlikely, but dependent on routing), at each aggregation point only a single Data packet will be forwarded towards the content consumer. Data-based distributed denial of service (DDoS) attacks are simply not possible.

As Interests can be generated by consumers at a rate of their choosing, it is theoretically possible to mount an *Interest flooding attack* – distributed generation of huge numbers of Interests in hopes of overwhelming the available bandwidth to the node(s) or networks CCN routers believe are the most likely sources of matching content. Multiple Interests requesting the same Data will be combined by CCN routers, and only a single copy will be forwarded upstream. So to attempt an Interest flood, an attacker must generate Interests for names that begin with a prefix supplied by the target, but which contain unique name components (to prevent Interest combining). As CCN routing allows transmission of Interests for content that does not yet exist, such Interests would normally be forwarded to their intended target.

Two features of CCN routing make it easy to mitigate such attacks: first, as Data packets follow the Interests they satisfy back to the consumer, every CCN intermediary node is able to see, for each Interest it forwards, whether or not that Interest successfully retrieves Data. (This is not the case in IP networking, where forward and return paths through the network are frequently disjoint.) Such randomly generated Interest flood packets will in general never result in Data responses. A simple adaptive algorithm allows intermediary routers to limit the number of Interests they will forward under a certain prefix as a function of how many previous Interests for that prefix have been successful (resulted in Data). Second, the attacked domain can ask downstream routers to throttle the number of Interests they forward by name prefix, much as an IP network might ask its upstream IP to throttle or block attempts to access unused addresses. However, CCN's ability to attach policy to content namespaces allows semantically selective control.

CCN also provides tools that allow an organization to exercise control over where their content will travel. Routers belonging to an organization or service provider can enforce *policy-based routing*, where content forwarding policy is associated with content name and signer. One simple example is a "content firewall" that only allows Interests from the Internet to be satisfied if they were requesting content under the `/parc.com/public` namespace. An organization could publish its policies about what keys

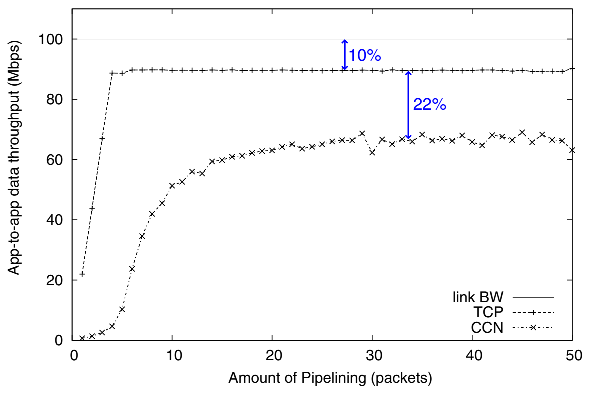


Figure 8: Bulk data transfer performance

can sign content under a particular name prefix (*e.g.,* all keys signed by `ccnx.org`), and have their content routers automatically drop content that does not meet those requirements, without asking those routers to understand the semantics of the names or organizations involved. Finally, Interests could in certain cases be digitally signed, enabling policy routing to limit into what namespaces or how often particular signers may query.

6. EVALUATION

In this section we describe and evaluate the performance of our prototype CCN implementation. Our current implementation encodes packets in the *ccnb* compact binary XML representation using dictionary-based tag compression. Our CCN forwarder, `ccnd`, is implemented in C as a userspace daemon. Interest and Data packets are encapsulated in UDP for forwarding over existing networks via broadcast, multicast, or unicast.

Most of the mechanics of using CCN (`ccnd` communication, key management, signing, basic encryption and trust management) are embodied in a CCN library. This library, implemented in Java and C, encapsulates common conventions for names and data such as encoding fragmentation and versioning in names or representing information about keys for encryption and trust management. These conventions are organized into *profiles* representing application-specific protocols layered over basic CCN Interest–Data.

This architecture has two implications. First, the security perimeter around sensitive data is pushed into the application; content is decrypted only inside an application that has rights to it and never inside the OS networking stack or on disk. Second, much of the work of using CCN in an application consists of specifying the naming and data conventions to be agreed upon between publishers and consumers.

All components run on Linux, Mac OS X$^{\text{TM}}$, Solaris$^{\text{TM}}$, FreeBSD, NetBSD and Microsoft Windows$^{\text{TM}}$. Cryptographic operations are provided by OpenSSL and Java.

6.1 Data Transfer Efficiency

TCP is good at moving data. For bulk data transfer over terrestrial paths it routinely delivers app-to-app data throughput near the theoretical maximum (the bottleneck link bandwidth). TCP can 'fill the pipe' because its variable window size allows for enough data in transit to fill the bandwidth$\times$delay product of the path plus all of the intermediate store-and-forward buffer stages[21]. CCN's ability to have multiple Interests outstanding gives it the same capability (see Section 3.1) and we expect its data transfer performance to be similar to TCP's.

	Bytes (packets)		Overheads	
	Sent	Received	Encap	Transact
Web page (6429 bytes)				
HTTP	723 (9)	7364 (9)	15%	11%
CCN/ETH	811 (8)	8101 (6)	26%	13%
CCN/UDP	325 (3)	6873 (5)	7%	5%
Secured Web page (16944 bytes)				
HTTPS	1548 (16)	21232 (22)	25%	9%
CCN/ETH	1791 (16)	20910 (14)	23%	11%
CCN/UDP	629 (5)	18253 (14)	8%	4%

Table 1: Web content efficiency

To test this we measured the time needed to transfer a 6MB file as a function of the window size (TCP) and number of outstanding Interests (CCN). The tests were run between two Linux hosts connected by 100Mb/s links to our campus ethernet. For the TCP tests the file was transferred using the test tool *ttcp*. For the CCN tests the file was pre-staged into the memory of the source's *ccnd* by requesting it locally.[11] This resulted in 6,278 individually named, signed CCN content objects each with one KB of data (the resulting object sizes were around 1350 bytes).

Results can be seen in Figure 8.[12] CCN requires five times the pipelining of TCP, 20 packets vs. 4, to reach its throughput asymptote. This is an artifact of the additional store-and-forward stages introduced by our prototype's totally unoptimized task-level implementation vs. Linux TCP's highly optimized in-kernel implementation. TCP throughput asymptotes to 90% of the link bandwidth, reflecting its header overhead (payload to packet size ratio). CCN asymptotes to 68% of the link bandwidth. Since CCN was encapsulated in IP/UDP for this test, it has all the overhead of the TCP test plus an additional 22% for its own headers. Thus for this example the bulk data transfer efficiency of CCN is comparable to TCP but lower due to its larger header overhead.[13]

Bulk data transfer performance is important for things like downloading large multimedia files, but users' perception of the speed of the net is driven by how quickly it can deliver (much smaller) web page content items. We compared the relative performance of CCN with HTTP and HTTPS to retrieve a single HTML file. For the unsecure (HTTP) example we used Google's home page. For the secure (HTTPS) example we used Wells Fargo Bank's home page. Two different CCN encapsulations are measured: directly into 1500 byte ethernet packets (no IP or UDP headers) using a payload size of 1230 bytes, and into UDP datagrams using a max payload size of 7656 bytes.[14]

The results are summarized in Table 1. The first two columns give the total number of bytes and packets sent and received by the client (including all protocol headers and control traffic such as SYNs and ACKs). The last two columns are computed from the first two and give overhead percentages for each protocol: *Encap* measures the data encapsulation overhead and is the ratio of overhead bytes (total bytes received – data bytes received) to data bytes. *Transact* measures the transaction overhead (cost of soliciting the

[11]This was done so the measurement would reflect just communication costs and not the signing cost of CCN content production.

[12]Since CCN transacts in packet-sized content chunks, the TCP window size was divided by the amount of user data per packet to convert it to packets.

[13]Most of the CCN header size increase vs. TCP is due to its security annotation (signature, witness and key locator).

[14]Since these datagrams were larger than the 1500 byte path MTU they were IP fragmented. The packet and byte counts include the fragments and fragment headers.

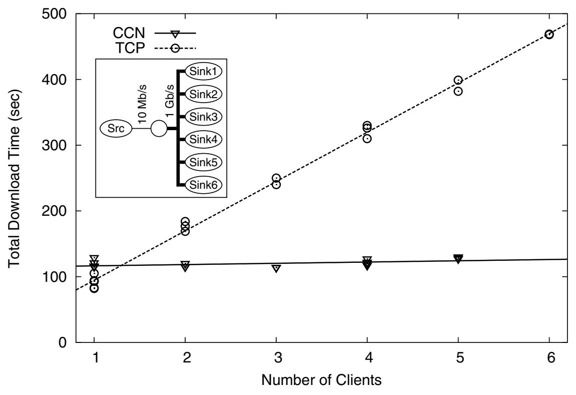


Figure 9: Total transfer time vs. the number of sinks.

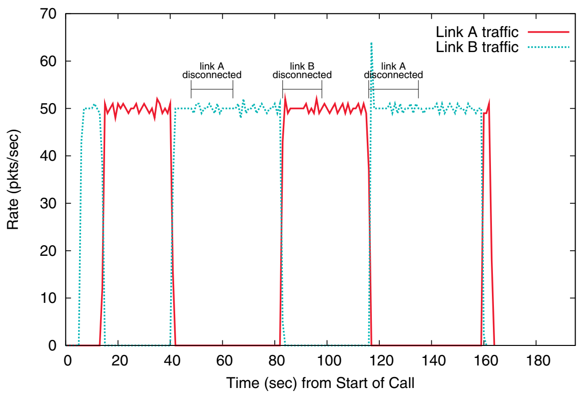


Figure 10: CCN automatic failover.

content) and is the ratio of total bytes sent to data bytes received.

Content transfer via CCN is always secure, yet the results show that it matches the performance of unsecured HTTP and substantially outperforms secure HTTPS. CCN-over-ethernet is essentially the same as HTTP (more bytes but fewer packets and round-trip times) and twice as efficient as HTTPS (half the number of packets). CCN-over-jumbo-UDP is twice as efficient as HTTP and three times more efficient than HTTPS in both overhead and packets.

6.2 Content Distribution Efficiency

The preceding sections compared CCN vs. TCP performance when CCN is used as a drop-in replacement for TCP, *i.e.,* for point-to-point conversations with no data sharing. However, a major strength of CCN is that it offers automatic, transparent sharing of all data, essentially giving the performance of an optimally situated web proxy for all content but requiring no pre-arrangement or configuration.

To measure sharing performance we compared the total time taken to simultaneously retrieve multiple copies of a large data file over a network bottleneck using TCP and CCN. The test configuration is shown in the inset of Figure 9 and consisted of a source node connected over a 10 Mbps shared link to a cluster of 6 sink nodes all interconnected via 1 Gbps links.[15] The machines were of various architectures (Intel, AMD, PowerPC G5) and operating systems (Mac OS X 10.5.8, FreeBSD 7.2, NetBSD 5.0.1, Linux 2.6.27).

The sinks simultaneously pulled a 6MB data file from the source. For the TCP tests this file was made available via an http server on the source and retrieved by the sinks using `curl`. For the CCN tests this file was pre-staged as described in Section 6.1. For each test, the contents of the entire file were retrieved and we recorded the elapsed time for the last node to complete the task. Multiple trials were run for each test configuration varying the particular machines which participated as sinks.

Test results are shown in Figure 9. With a single sink TCP's better header efficiency allows it to complete faster than CCN. But as the number of sinks increases TCP's completion time increases linearly while the CCN performance stays constant. Note that since the performance penalty of using CCN vs. TCP is around 20% while the performance gain from sharing is integer multiples, there is a net performance win from using CCN even when sharing ratios / hit rates are low. The win is actually much larger than it appears from this test because it applies, independently, at every link in the network and completely alleviates the traffic concentrations we now see at popular content hubs and major peering points. For example, today a popular YouTube video will traverse the link between youtube.com and its ISP millions of times. If the video were distributed via CCN it would cross that link once. With the current architecture, peak traffic loads at aggregation points scale like the total consumption rate of popular content. With CCN they scale like the popular content creation rate, a number that, today, is exponentially lower.

[15]We used a 10 Mbps bottleneck link to clearly show saturation behavior, even with only a small number of nodes.

6.3 Voice-over-CCN and the Strategy Layer

To demonstrate how CCN can support arbitrary point-to-point protocols we have implemented Voice-over-IP (VoIP) on top of CCN (VoCCN). Complete details and performance measurements are given in [22]. In this section we describe a test that uses a VoCCN call to demonstrate the behavior and advantages of CCN's strategy layer.

As described in Section 3.3, when the FIB contains multiple faces for a content prefix, the strategy layer dynamically chooses the best. It can do this because CCN can send the same Interest out multiple faces (since there is no danger of looping) and because a CCN node is guaranteed to see the Data sent in response to its Interest (unlike IP where the request and response paths may be almost entirely disjoint). These two properties allow the strategy layer to run experiments where an Interest is occasionally sent out all faces associated with the prefix. If a face responds faster than the current best, it will become the new best and be used exclusively for the prefix's Interests, until it is time for the next experiment (*e.g.,* after 200 packets, when there is a change in carrier or SSID, or when an Interest does not get a response and times out).

To test this mechanism we ran our *linphone*-based VoCCN client between two Linux 2.6.27 machines (a 3.4 GHz Intel P4 and a 2.66 GHz Intel Core2 Duo) each connected to two isolated wired 1 Gbps ethernet networks. The *linphone* default is to packetize audio into 20ms frames so audio activity resulted in a constant 50pps source of RTP packets. As measured by voice quality, the performance of our secure VoCCN prototype was equivalent to that of stock *linphone*. No packets were lost by either client, however a small number of VoCCN packets ($< 0.1\%$) were dropped for arriving too late.

We conducted failover tests by manually disconnecting and reconnecting network cables. Figure 10 shows the traffic on both links during one of these tests. The strategy layer initially picks link `B` but at 15 seconds into the call it switches to link `A` in reaction to some small variance in measured response time, then switches back to link `B` at 40 seconds. At 45 seconds we unplugged link `A` but this had no impact since link `B` was being used at the time. At 60 seconds link `A` was reconnected. At 82 seconds link `B` was disconnected. The strategy layer switches to link `A`, and there is a small excursion above 50pps in the link `A` traffic rate as the packets produced during the failure detection time are retrieved from the upstream *ccnd*. At 95 seconds link `B` is reconnected; traffic remains on link `A`. At 120 seconds link `A` is disconnected, and the CCN strategy layer switches back to link `B`, but the failure detection took longer this time as shown by the large traffic burst on `B` immediately following the switch. There is one more spontaneous switch at 160 seconds then the call terminates at 165 seconds.

The failover behavior is not coded into our client but arises entirely from the CCN core transport. The small delay for the final failover reflects the preliminary state of our current implementation (it does not listen for 'carrier lost' notifications from the ethernet driver so failure detection is timeout rather than event driven). It is interesting to note that after failing over the client is able to retrieve the missing conversation data from CCN: a few packets were delayed but none were lost.

7. RELATED WORK

It is widely recognized that combining identity and location information into a single network address does not meet the demands of today's applications and mobile environments. Proposed remedies implement functionality above the current Internet architecture, replace it in a "clean slate" approach, or combine aspects of both. Like CCN, these proposals aim to switch from host- to content-oriented networking to meet data-intensive application needs.

Prior content-oriented networking research is dominated by the use of unstructured, opaque, usually self-certifying content labels. The challenges these systems face are efficiently routing queries and data based on the "flat" names, and providing an indirection mechanism to map user-meaningful names to the opaque labels.

The Data-Oriented Network Architecture [24] replaces DNS names with flat, self-certifying names and a name-based anycast primitive above the IP layer. Names in DONA are a cryptographic digest of the publisher's key and a potentially user-friendly label – however, that label is not securely bound to the content, allowing substitution attacks. Unlike CCN, data cannot be generated dynamically in response to queries – content in DONA must first be published, or *registered*, with a tree of trusted resolution handlers (RHs) to enable retrieval. Each resolution handler must maintain a large forwarding table providing next hop information for every piece of content in the network. Once the content is located, packets are exchanged with the original requester using standard IP routing. If the location of a piece of content changes, new requests for it will fail until the new registration propagates through the network. CCN, in contrast, can forward requests to all the places a piece of content is likely to be.

A number of systems make use of distributed hash tables (DHTs) to route queries for opaque content names. ROFL (Routing on Flat Labels) evaluates the possibility of routing directly on semantic-free flat labels [7]. A circular namespace is created to ensure correct routing (as in Chord [36]), but additional pointers are added to shorten routes. In a similar approach, i3 [35] separates the acts of sending and receiving by using a combination of packet identifiers and a DHT. Receivers insert a trigger with the data identifier and their address into the DHT. The trigger is routed to the appropriate sender, who fulfills the request by responding with the packet containing the same id and the requested data. SEATTLE [23] utilizes flat addressing with a one-hop DHT to provide a directory service with reactive address resolution and service discovery. Unlike CCN, all of these systems require content be explicitly published to inform the DHT of its location before it can be retrieved. Also unlike CCN, this retrieval is largely free of locality – queries might retrieve a cached copy of data along their routed path, but are not guaranteed to retrieve the closest available copy.

Instead of routing end-to-end based on an identifying name, the PSIRP project [33] proposes using rendezvous as a network primitive. Each piece of data has both a public and private label used for verifying the publisher and making routing decisions. Consumers receive content by mapping the desired, user-friendly name to an opaque public label via an unsecure directory service. The label is then used to subscribe to the piece of data, triggering the system to locate and deliver the corresponding content. Though motivated by the same problems as CCN, PSIRP suffers from its use of unstructured identifiers and lack of strong cryptographic binding between user-meaningful names (or currently, even their opaque labels) to content.

The 4WARD NetInf project [29] has similar goals to CCN but focuses on higher level issues of information modeling and abstraction. It currently uses DONA-style names for Data and Information Objects and provides a publish/subscribe style API. The NetInf Dictionary infrastructure uses a DHT for name resolution and location lookup.

TRIAD [8], like CCN, attempts to name content with user-friendly, structured, effectively location-independent names. TRIAD uses URLs as its names using an integrated directory to map from the DNS component of the URL to the closest available replica of that data. It then forwards the request to that next hop, continuing until a copy of the data is found. Its location is returned to the client, who retrieves it using standard HTTP/TCP. TRIAD relies on trusted directories to authenticate content lookups (but not content itself), and suggests limiting the network to mutually trusting content routers for additional security.

Research into content-aware routing protocols also attempts to improve delivery performance and reduce traffic overhead. For example, Anand et. al [5] studied the benefits of large-scale packet caching to reduce redundant content transmission. In this work, routers recognize previously forwarded content and strip the content from packets on the fly, replacing the content portion with a representative fingerprint. Downstream routers reconstruct the content from their own content cache before delivering to the requester.

8. CONCLUSIONS

Today's network *use* centers around moving content, but today's *networks* still work in terms of host-to-host conversations. CCN is a networking architecture built on IP's engineering principles, but using named content rather than host identifiers as its central abstraction. The result retains the simplicity and scalability of IP but offers much better security, delivery efficiency, and disruption tolerance. CCN is designed to replace IP, but can be incrementally deployed as an overlay – making its functional advantages available to applications without requiring universal adoption.

We implemented a prototype CCN network stack, and demonstrated its usefulness for both content distribution and point-to-point network protocols. We released this implementation as open source and it is available from [1].

Acknowledgements

We thank Paul Stewart, Paul Rasmussen and Simon Barber for substantial help with implementation. We also thank Ignacio Solis, Marc Mosko, Eric Osterweil and Dirk Balfanz for fruitful discus-

sion and advice.

9. REFERENCES

[1] Project CCNx^{TM}. `http://www.ccnx.org`, Sep. 2009.

[2] M. Abadi. On SDSI's Linked Local Name Spaces. *Journal of Computer Security*, 6(1-2):3–21, October 1998.

[3] B. Adamson, C. Bormann, M. Handley, and J. Macker. *Multicast Negative-Acknowledgement (NACK) Building Blocks*. IETF, November 2008. RFC 5401.

[4] W. Adjie-Winoto, E. Schwartz, H. Balakrishnan, and J. Lilley. The Design and Implementation of an Intentional Naming System. *SIGOPS Oper. Syst. Rev.*, 33(5):186–201, 1999.

[5] A. Anand, A. Gupta, A. Akella, S. Seshan, and S. Shenker. Packet Caches on Routers: The Implications of Universal Redundant Traffic Elimination. In *SIGCOMM*, 2008.

[6] H. Balakrishnan, K. Lakshminarayanan, S. Ratnasamy, S. Shenker, I. Stoica, and M. Walfish. A Layered Naming Architecture for the Internet. In *SIGCOMM*, 2004.

[7] M. Caesar, T. Condie, J. Kannan, K. Lakshminarayanan, I. Stoica, and S. Shenker. ROFL: Routing on Flat Labels. In *SIGCOMM*, 2006.

[8] D. Cheriton and M. Gritter. TRIAD: A New Next-Generation Internet Architecture, Jan 2000.

[9] I. Clarke, O. Sandberg, B. Wiley, and T. W. Hong. Freenet: A Distributed Anonymous Information Storage and Retrieval System. *Lecture Notes in Computer Science*, 2009:46, 2001.

[10] C. M. Ellison, B. Frantz, B. Lampson, R. Rivest, B. M. Thomas, and T. Ylonen. *SPKI Certificate Theory*, September 1999. RFC2693.

[11] S. Farrell and V. Cahill. *Delay- and Disruption-Tolerant Networking*. Artech House Publishers, 2006.

[12] K. Fu, M. F. Kaashoek, and D. Mazières. Fast and secure distributed read-only file system. *ACM Trans. Comput. Syst.*, 20(1):1–24, 2002.

[13] J. F. Gantz et al. IDC - The Expanding Digital Universe: A Forecast of Worldwide Inform ation Growth Through 2010. Technical report, March 2007.

[14] A. Gulbrandsen, P. Vixie, and L. Esibov. *A DNS RR for specifying the location of services (DNS SRV)*. IETF - Network Working Group, The Internet Society, February 2000. RFC 2782.

[15] E. Guttman, C.Perkins, J. Veizades, and M. Day. *Service Location Protocol*. IETF - Network Working Group, The Internet Society, June 1999. RFC 2608.

[16] IETF. RFC 2328 – OSPF Version 2.

[17] IETF. RFC 3787 – Recommendations for Interoperable IP Networks using Intermediate System to Intermediate System (IS-IS).

[18] IETF. RFC 4971 – Intermediate System to Intermediate System (IS-IS) Extensions for Advertising Router Information.

[19] IETF. RFC 5250 – The OSPF Opaque LSA Option.

[20] V. Jacobson. Congestion Avoidance and Control. In *SIGCOMM*, 1988.

[21] V. Jacobson, R. Braden, and D. Borman. *TCP Extensions for High Performance*. IETF - Network Working Group, The Internet Society, May 1992. RFC 1323.

[22] V. Jacobson, D. K. Smetters, N. Briggs, M. Plass, P. Stewart, J. D. Thornton, and R. Braynard. VoCCN: Voice-over Content-Centric Networks. In *ReArch*, 2009.

[23] C. Kim, M. Caeser, and J. Rexford. Floodless in SEATTLE: A Scalable Ethernet Architecture for Large Enterprises. In *SIGCOMM*, 2008.

[24] T. Koponen, M. Chawla, B.-G. Chun, A. Ermolinskiy, K. H. Kim, S. Shenker, and I. Stoica. A Data-Oriented (and Beyond) Network Architecture. In *SIGCOMM*, 2007.

[25] J. Kubiatowicz et al. OceanStore: An architecture for global-scale persistent storage. *SIGPLAN Not.*, 35(11):190–201, 2000.

[26] D. Mazières, M. Kaminsky, M. F. Kaashoek, and E. Witchel. Separating Key Management from File System Security. In *SOSP*, 1999.

[27] R. C. Merkle. *Secrecy, authentication, and public key systems*. PhD thesis, 1979.

[28] R. Moskowitz and P. Nikander. *Host Identity Protocol Architecture*. IETF - Network Working Group, May 2006. RFC 4423.

[29] B. Ohlman et al. First NetInf architecture description, April 2009. `http://www.4ward-project.eu/index.php?s=file_download&id=39`.

[30] E. Osterweil, D. Massey, B. Tsendjav, B. Zhang, and L. Zhang. Security Through Publicity. In *HOTSEC '06*, 2006.

[31] B. C. Popescu, M. van Steen, B. Crispo, A. S. Tanenbaum, J. Sacha, and I. Kuz. Securely replicated web documents. In *IPDPS*, 2005.

[32] R. L. Rivest and B. Lampson. SDSI - A Simple Distributed Security Infrastructure. Technical report, MIT, 1996.

[33] M. Särelä, T. Rinta-aho, and S. Tarkoma. RTFM: Publish/Subscribe Internetworking Architecture. In *ICT-MobileSummit*, 2008.

[34] D. K. Smetters and V. Jacobson. Securing network content, October 2009. PARC Technical Report.

[35] I. Stoica, D. Adkins, S. Zhuang, S. Shenker, and S. Surana. Internet Indirection Infrastructure. In *SIGCOMM*, 2002.

[36] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A Scalable Peer-To-Peer Lookup Service for Internet Applications. In *SIGCOMM*, 2001.

[37] D. Wendlandt, D. Andersen, and A. Perrig. Perspectives: Improving SSH-style host authentication with multi-path probing. In *USENIX*, 2008.