Networking Named Content

Paper 178 (12 pages)

Van Jacobson D. K. Smetters James D. Thornton Michael Plass Nick Briggs
Rebecca L. Braynard

Palo Alto Research Center (PARC)
{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 can only speak 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 takes 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 have implemented the basic features of our architecture 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 the low storage costs have enabled access to a staggering amount of new content – 500 exabytes created in 2008 alone [12]. 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 access to content 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.
- **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. While many of the Internet's design principles remain valid, 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 (described in 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

ACM CoNEXT 2009, December 1–4, 2009, Rome, ITALY.

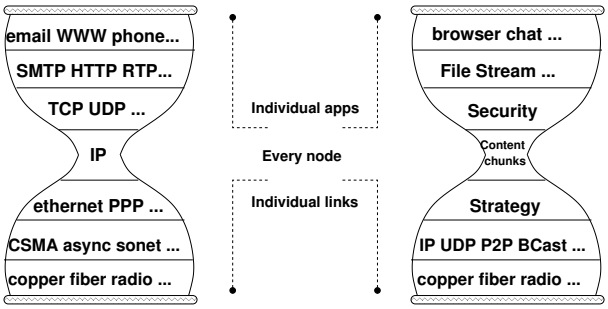

Figure 1: CCN moves the universal component of the network stack from IP to chunks of named content.

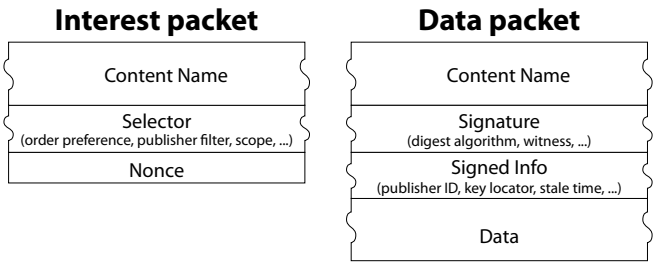

Figure 2: CCN packet types

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 3.1) 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 [2].

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 the same matching convention to resolve the $< net, subnet, host >$ hierarchical structure of IP addresses. Experience has shown that this 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. Content names may also be context-dependent.

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.[3] 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).

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

[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 [19] 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]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.

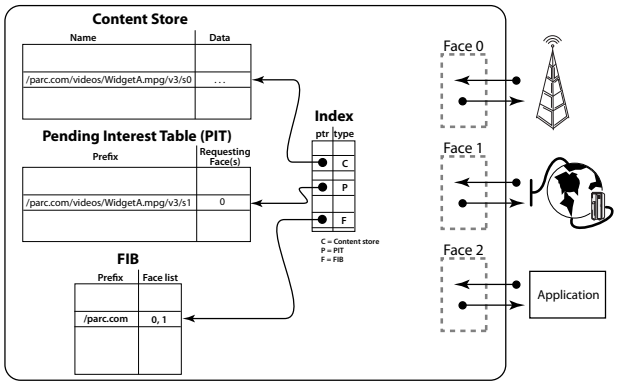

Figure 3: CCN forwarding engine model

Data packets as long as possible (LRU or LFU replacement).

The PIT keeps track of Interests that have been forwarded upstream toward content source(s) so that returned Data can be sent downstream to its requestor(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 requestor(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 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 will have been 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 doesn't have any matching data and does not know how to find any).

The processing of Data packets is relatively simple since Data is not routed, it simply follows the chain of PIT entries back to the original requestor(s). When a Data packet arrives a longest-match lookup of it's ContentName is done. 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's discarded[4] 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 network environments. Any node having access to multiple networks can serve as a content router between them. Using its cache, a mobile node can even serve as the network medium between disconnected areas, or provide delayed connectivity over sporadically connected links. Thus CCN transport provides Disruption Tolerant Networking [10]. The Interest/Data exchange also functions whenever there is local connectivity. For example, two business colleagues with laptops and ad-hoc wireless could continue to share corporate documents normally even in an isolated location with no connectivity to the Internet or rest of the organization.

3. TRANSPORT

3.1 Reliability and Flow Control

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 receiver (the application that originated the initial Interest) is responsible for maintaining a timer on unsatisfied Interests and resending them if it still wants the data. The receiver's strategy layer (see Figure 1) is responsible for retransmission. It

[4] 'Unsolicited' Data can arise from malicious behavior but also from data arriving from multiple sources or multiple paths from a single source. In the latter cases the first copy of the Data that arrives will consume the Interest so the duplicate(s) won't 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.

is also responsible such things 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 describe in the preceding section. But while data cannot loop in CCN, in richly connected topologies Interests can go round a loop and make it appear that 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 here and sequencing in the next section.

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 that there are many points between sender and receiver where congestion can occur from aggregation of conversations 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 [19]. 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 plan to 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

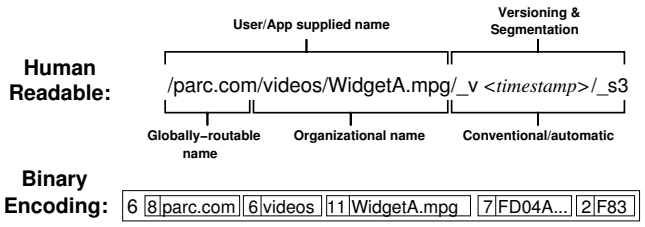

Figure 4: Example Data name

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 reflect some organizational structure of their origin, rather than just the sequence in an ephemeral conversation. But, despite this extra richness in CCN names, the transport function of their use in Interests is exactly the same as that of sequence numbers in TCP ACKs: an Interest specifies the next Data that 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. The final component of every Data packet name includes a SHA256 digest of the data itself. The digest component is not actually transmitted in packets since it is derivable from the data itself.

An Interest can specify precisely what content is required next with the full name of the Data (digest included) when it is known, as when dereferencing secure links. In most cases, however, the full name of the next Data will not be known in advance by the consumer, and the Interest will specify what is needed next *relative* to a collection of packets whose names are known. The names of all possible packets form a single *name tree*, and any *name prefix* identifies a sub-tree and thereby the collection of data with names sharing that prefix. Although the names are opaque to CCN transport, we apply a simple lexicographic ordering for sibling components in the tree, in which shorter components precede longer ones and those of equal length are ordered by their value as unsigned integers. With the addition of a traversal order rule (pre-order) we have a total ordering defined for the name tree. Thus an Interest can request the *next* Data packet within a collection according to the total ordering,

and the request can be satisfied by the CCN transport without knowledge of any semantic interpretations of the names.

Let us illustrate this with a concrete example from Figure 4. The name in the figure is a hypothetical name of one packet of video data. The prefix `/parc.com/videos/WidgetA.mpg` identifies the collection of data that makes up what humans would call a video file. This is the only name that a user would typically see, and would be the name presented to a player application. Starting from that name alone, an Interest can be generated requesting the next Data packet having exactly that prefix, which has the semantic effect of requesting the first available packet in the video. Suppose that this Interest retrieves a packet with the name as shown in Figure 4. In this example, the names of individual video packets are constructed with a version marker (`_v`) and an integer version number and a constant segment marker (`_s`) and an integer value which might be the frame number of the first video frame contained in the packet. Given this name, a new Interest can be constructed to request the packet with the next name in the tree *after* the one in Figure 4 but still having the same prefix (*i.e.,* still within the same collection/video file).

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 a very 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.

Interests, then, provide a form of restricted query mechanism over collections of content accessible 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 that are under development. It will be possible to restrict results in certain ways by publisher, not just by collection, and to exclude content already obtained when simple ordering is insufficient. We also plan optional name discovery mechanisms to be implemented on top of CCN transport that will be more efficient for exploring large name subtrees when the content itself is not required.

4. ROUTING

4.1 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') [15, 13]. These two functions are orthogonal—one describes links in the graph while the other describes what is available at particular nodes in the graph. It is common for these two functions to be performed in completely different information domains. For example, IS-IS [15] describes adjacencies in terms of IEEE 802.1 layer 2 MAC addresses but announces layer 3 IP4 and/or IP6 prefixes. As described in the previous section, 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.

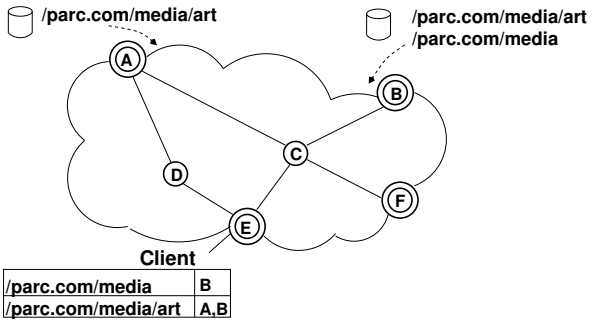

Figure 5: Routing Interests to a domain's media content

Since CCN prefixes are very different from IP prefixes, the main issue 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 very general TLV ('type label value') scheme [17, 18] that is quite suitable for distributing CCN content prefixes. The spec says that unrecognized types should be ignored, which means that content routers, implementing the full CCN Interest/Data 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 5 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 running 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. If a client adjacent to *E* expresses interest in /parc.com/media/art/impressionist-

history.mp4, this interest will get forwarded to both *A* and *B*, who will each forward it to their adjacent repository.

CCN, in general, 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 and, 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 5, 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 essentially 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's forced to construct loop-free forwarding topologies, *i.e.,* a sink tree rooted at the destination. Since a tree has only 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. Fortunately this semantic difference can be accommodated without changing the IGP because it's an implementation change, not a protocol change. *i.e.,* IP has to compute a spanning tree from the 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. However, if the IGP were a distance vector protocol such as RIP or EIGRP, the production of a routing announcement involves a Bellman-Ford calculation that presupposes spanning trees and will lose information. Such an IGP would not be suitable for CCN (based on a long history of routing problems, there are many who believe distance vector protocols are not even suitable for IP). A CCN deployment in such an environment might require its own routing protocol.

4.2 Inter-domain Routing

Current BGP inter-domain routing has the equivalent of the IGP TLV mechanism that would allow domains to advertise their 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. IP network clients often must retrieve content directly from the publisher, rather than efficiently sharing cached copies with their peers, since peers can only trust content they have retrieved directly from the original source. Embodying security in content, not hosts, also reduces the trust we need to place in network intermediaries. This opens the network to wide participation. In this paper, 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, are in preparation for submission in a separate paper.

5.1 Validating Content

Each CCN data packet (Figure 2) contains digital signature information, authenticating the *binding* between its name and its corresponding content. This allows publishers to securely bind arbitrary names to content, in contrast to previous approaches that rely on *self-certifying* name structures to achieve such a binding (*e.g.,* by using the cryptographic digest of the content as its name [23, 25, 11, 8]). The ability to directly use user- or application-meaningful names to securely obtain data enhances usability and eases transport; systems without it require an "indirection infrastructure" [3, 5] to map from the names humans care about to secure, opaque, self-certifying names.

CCN's per-Data packet signature information consists of a standard public key signature as well as (signed) supporting

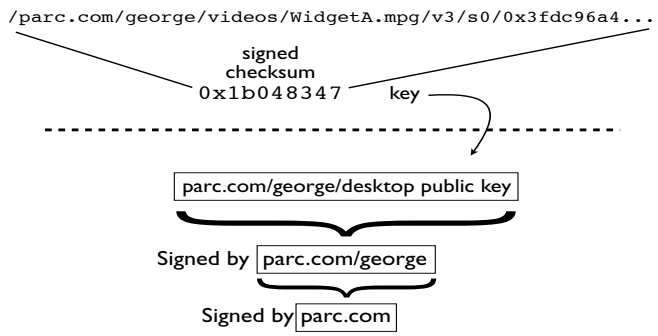

Figure 6: CCN trust establishment can associate content namespaces with publisher keys.

data helpful in verifying it. The use of public key signatures makes all CCN data *publicly authenticatable* – anyone, not just the endpoints for a communication stream, can verify that a name-content mapping was signed by a particular key.

The signature algorithm used may be 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. Data packets are designed to be individually verifiable, but the cost of signature generation may be amortized across multiple packets through the use of aggregation techniques such as Merkle Hash Trees [24]. Supporting data ("signed info" in Figure 2) includes the fingerprint of the public key used for signing, to select the correct key for verification (*e.g.,* from a local cache), and a *key locator*, which indicates where that key can be obtained; this can be the key itself, or a CCN name where the key can be retrieved.

5.2 Trust Establishment

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* for their individual purpose. Trust in content is narrowly determined, both by who signed it, and specifically what the content is (as indicated by its name), and what it will be used for (as determined by the consumer). This approach to *contextual* trust is much more flexible and easier to use than attempts to mandate a one-size-fits-all trust establishment mechanism (such as a global PKI).

At the lowest layers, CCN routers may simply verify that content was signed by the key it purports – without attaching any real-world identity or semantics to *who* owns that key. This can be surprisingly effective. This allows content consumers to request content by publisher key as well as name, and to get the content they intend in the face of spurious or malicious data. This most generic form of verification is used to defend against many types of network attack. Routers may choose to verify all, some or no data, as their resources allow; they may also dynamically adapt, verifying more data in response to detected attack.

Application-level CCN consumers must solve traditional key management problems – associating public keys with individuals and organizations, and determining what keys are acceptable signers for each type of content they are interested in. CCN simplifies this task in three 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.

Second, by organizing content in terms of hierarchical *namespaces*, CCN allows signing policy, and even keys, to attach to particular names; authorization at one level of a namespace is given by a signature from a key at a higher level. Figure 6 shows the key for `parc.com` authorizing that of the user `george`, who goes on to authorize 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` namespace. Such namespaces make up a *forest-of-trees* – a content consumer might trust that they have the right key for `parc.com` (or even directly for `/parc.com/george`) for any number of reasons from direct experience (*e.g.,* they are a PARC employee), to information provided by their 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.

CCN's signed bindings between names and content act in essence to certify that content – and when that content is a public key, as a certificate for that public key. It is trivial to represent a traditional certificate-based public key infrastructure (PKI), or PGP Web of Trust, directly in CCN data. Users are free to reuse these existing models for establishing trust in keys, or to embrace or define new ones more appropriate to CCN. A model particularly appropriate to CCN is that of SDSI/SPKI [30, 9, 1], wherein identities, and corresponding keys form local *namespaces* that can be mapped directly onto CCN names – *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).

Third, CCN enables *secure linkage* between content items – one content item can refer to another not only by its name, but also by the cryptographic digest of its contents (forming effectively a *self-certifying name* [23, 25, 11, 8]), or by the identity (key) of its publisher [8, 29, 22, 21]). This allows individual signed pieces of content to effectively certify other pieces of content they (securely) refer to; each piece of content the user encounters acts as a potential piece of *evidence* as to the validity of the content and keys they have en-

countered before. A user may also, having decided to trust content *A*, say a web page, automatically trusts the content *A* securely links to – *e.g.,* its images, ads, source material and so on, without additional management or configuration overhead; however that trust is very fine-grained – those materials are only considered valid within the context of *A*.

Together, these mechanisms allow users to leverage bootstrapped trust in a small number of public keys accepted using a variety of user-friendly mechanisms (*e.g.,* personal contact, organizational membership, public experience [28, 34]), to allow fine-grained validation of a wide range of content in context.

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. 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 standard classes of 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.

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 over anything beyond the local link; extra Data packets will not be forwarded. To ensure they get the content they want in the face of potential spurious alternatives, consumers can request it by publisher as well as by name. Routers belonging to an organization or service provider can enforce *policy-based routing*, where content forwarding policy is associated with content name and signer. A simple example of such might be a "content firewall" that only allows Interests from the Internet to be satisfied if they were requesting content under the `/parc.com/public` namespace. Finally, Interests may also be digitally signed, enabling policy routing to limit into what namespaces or how often particular signers may query.

6. EVALUATION

In this section, we evaluate the performance of our prototype CCN implementation in the context of two illustrative applications. Packets are encoded in the *ccnb* compact binary representation of XML, which uses dictionary-based tag compression. Our CCN forwarder, `ccnd` is implemented in C as a userspace deamon. Interest and Data packets are encapsulated in UDP for forwarding over existing networks as broadcast, multicast, or unicast.

Much of the "work" of using CCN – communicating with `ccnd`, key management, signing, basic encryption and much of trust management are embodied in a CCN library layer. This library layer (implemented in both Java and C) encapsulates a set of common *conventions* for names and data, covering things such as encoding fragmentation and versioning information in names, and representing keying information for encryption and trust management. These conventions are organized into topic-specific *profiles* representing application-level protocols layered on top of Interest-Data.

The host architecture has two notable implications. First, the security perimeter around sensitive data is pushed far into the application – content is decrypted only inside an application that has rights to it, not inside the networking stack or in cache or on disk. Second, much of the work of writing a CCN application consists of specifying a set of conventions to be agreed upon between publishers and consumers.

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

6.1 Voice over CCN

To demonstrate how CCN can be used to support arbitrary point-to-point protocols, we have implemented Voice over IP (VoIP) on top of CCN (VoCCN), and evaluated its performance. Implementing a point-to-point network protocol on top of CCN is done by taking advantage of the fact that Interests can be generated to retrieve Data that has not yet been created. Applications can register to receive Interests in response to which they are prepared to generate data. Such Interests are forwarded to the most likely source of corresponding data based on a longest prefix match. One can map the distinguishing fields in the original point-to-point protocol header directly into CCN name components, resulting in names that both parties can generate *a priori*. For example, an Interest in the next packet in a TCP exchange might ask for `/company/dst/<servername>/src/<client-name>/dport/25/sport/4567/seq/15238`.

We implemented a secure Voice over CCN client on top of Linphone (version 3.0), an open source Linux VoIP phone. VoCCN operates by encapsulating standard VoIP messages (SIP, RTP) directly inside CCN traffic, enabling interoperability with standard phones via a VoCCN-VoIP gateway.

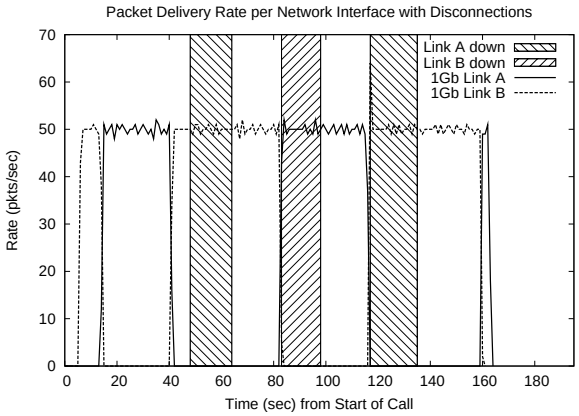

Figure 7: CCN automatic network interface failover.

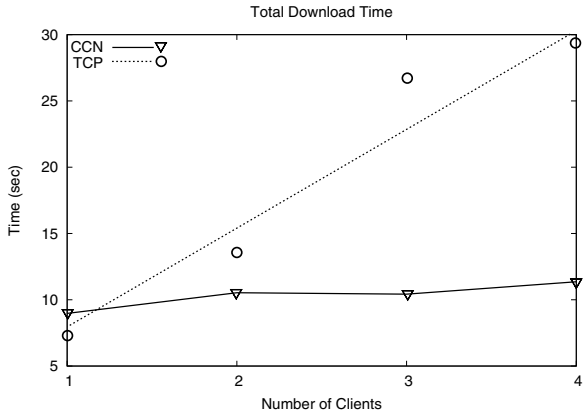

Figure 8: Mbps transfer vs. the number of sinks.

Each VoCCN client registers with its local `ccnd` its desire to receive Session Initiation Protocol (SIP) INVITE messages for its user. A caller encodes such an INVITE as a CCN name under a prefix specified by the phone's configured SIP identity: `/<sipdomain>/<sipuser>/sip-/INVITE/<contents>`, and sends that name as an Interest. On receipt of such an INVITE, the callee generates a signed Data packet with the INVITE name as its name, and the SIP response as its payload; thus completing SIP signalling in a single round trip. The VoCCN phones derive a set of paired name prefixes from information in the SIP messages under which to write their actual call data, in the form of RTP packets. These names can be cryptographically anonymized to unlink them from the SIP exchange, and provide user privacy. Our modified VoCCN-Linphone uses standard VoIP protocols to secure the call itself – MIKEY [16] to exchange session keys, and SRTP [14] to encrypt the media path; it uses native CCN encryption to protect the content of the SIP messages. All VoCCN Data packets, both SIP and SRTP, are signed with 1024-bit RSA keys; faster algorithms (*e.g.,* ESIGN [27]) can be used, if necessary, for video or other higher-rate data.

We measured the performance of our VoCCN client and compared it to an (insecure) stock Linphone. Our experimental data was collected from VoCCN calls between two machines, one an Intel P4 at 3.4 GHz and the other an Intel Core2 Duo at 2.66 GHz, both running Linux with a 2.6.27 kernel. Both machines were connected to two isolated wired 1 Gbs networks. We conducted failover tests by manually disconnecting and reconnecting network cables.

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 by Linphone for having arrived too late.

Figure 7 shows our simple CCN strategy layer's (Section 3.1) ability to automatically failover a call between network interfaces when one is disconnected. Instantaneous packet rate (expected 50 pps) is shown during the course of a VoCCN call between two hosts connected via two separate network interfaces (links `A` and `B`). The CCN strategy layer actively probes and measures the performance of multiple paths to a given destination. It dynamically selects the best available interface through which to send packets, adapting to changes in network performance. The call initially sends most of its packets over link `B`, but at 15 seconds into the call it spontaneously switches to link `A` in reaction to some small variance in measured response time, switching back to link `B` at 40 seconds. At 45 seconds into the call, link `A` is disconnected with no impact on packet throughput. At approximately 82 seconds into the call, link `B` is disconnected. The packet rate drops precipitously, until the CCN strategy layer discovers the other link automatically, and then increases to catch up. At 95 seconds into the call, link `B` is restored, with no measurable impact. At 120 seconds, link `A` is disconnected, and the CCN strategy layer switches the call back to link `B`. There is one more spontaneous switch at 160 seconds, and the call terminates at approximately 165 seconds.

There is no VoCCN-specific strategy code in our Linphone client to handle failover; this behavior arises entirely out of the CCN transport itself. The small delay for the third failover reflects the preliminary state of our current implementation; it is interesting to note that after failing over the client is able to retrieve the missing conversation data from CCN; packets are not lost, just delayed.

6.2 Content Distribution

We evaluate the network throughput and caching efficiency of CCN in the context of basic content distribution.

6.2.1 Throughput and Caching

To measure throughput, we compared the total time taken to retrieve multiple copies of a large data file simultaneously over a network bottleneck using CCN and TCP. The test configuration had a data source node connected over a 10 Mbs shared link to a cluster of 4 data sink nodes all intercon-

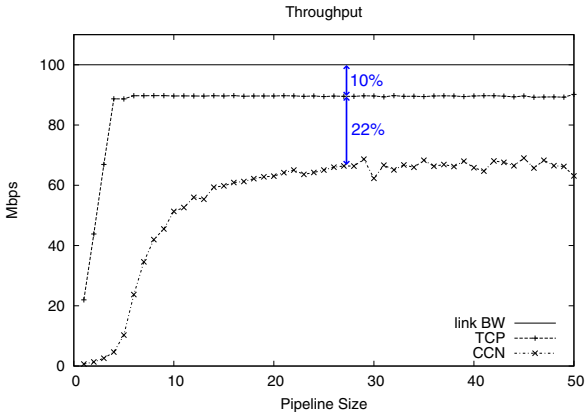

Figure 9: CCN pipelining provides high throughput.

nected via 1 Gbs links. We used a 10 Mbs link as the bottleneck to clearly show saturation behavior, even with only a small number of nodes. For the TCP tests, the sink nodes pulled the data individually from the source. For the CCN tests, nodes were arranged in a 2-hop hub and spoke topology, with the first node pulling data across the 10 Mbs link, and the others pulling data from it. The source node and two of the sinks were G5 Macs running Mac OS X (source, 10.4, sinks 10.5); the other two sinks were AMD and Intel machines running FreeBSD 7.1.

The source node was pre-loaded with an approximately 6 MB data file. For our TCP control condition this file was made available via a standard web server on the source, and retrieved by the sinks using `curl` or `fetch`. For the CCN test, this file was fragmented into 6,278 individually named, signed Data blocks of 1 kB each, representing approximately 1350 bytes on the wire. Prior to each test, the data blocks were loaded into the `ccnd` of the data source node. For each test, the contents of the entire file were retrieved by 1, 2, 3 or 4 sink nodes simultaneously, 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.

Median times are shown in Figure 8. With a single sink, TCP retrieves the data in slightly less time than CCN. As the number of sinks increases, TCP's performance dramatically declines, and it rapidly saturates the link. CCN's performance, on the other hand, is largely independent of the number of sinks participating – its built-in content caching ability allows the network to avoid sending any repeated copies of the same content, thus dramatically increasing throughput.

6.2.2 *Pipelining and Overhead*

We evaluated the ability of our CCN implementation to pipeline requests together by expressing multiple Interests simultaneously, increasing throughput much as the variable window size does in TCP (see Section 3.1). We measured data throughput (payload content, not packet size on the wire)

	HTTP	CCN/UDP	CCN/ETH
App Payload	6429	6429	6429
Packets Out	9	3	8
Bytes Out	723	325	811
Avg. Size Out	80.33	108.33	101.38
Packets In	9	5	6
Bytes In	7364	6873	8101
Overhead In	14.54 %	6.91 %	26.01 %
Avg. Size In	818.22	1374.6	1350.17

Table 1: Comparing CCN and HTTP

	HTTPS	CCN/UDP	CCN/ETH
App Payload	16944	16944	16944
Packets Out	16	5	16
Bytes Out	1548	629	1791
Avg. Size Out	96.75	125.8	111.94
Packets In	22	14	14
Bytes In	21232	18253	20910
Overhead In	25.31 %	7.73 %	23.41 %
Avg. Size In	965.09	1303.79	1493.57

Table 2: Comparing CCN and HTTPS

as a function of window size between the two Linux hosts used in Section 6.1, transferring the 6 MB file used above over a 100 Mb link. Pipeline size in TCP was defined as the TCP window size/1460 (a window size of 1 is a TCP window size of 1460 bytes), controlled with the SO_SNDBUF and SO_RECVBUF socket options; reads and writes were done in multiples of 1460 bytes.

Results can be seen in Figure 9. While the throughput of our (unoptimized) CCN implementation is not quite as good as that of TCP, it is close, and reasonably close to saturating the link. The 10% difference between TCP throughput and the theoretical link bandwidth is due to the overhead of the TCP headers; our prototype CCN implementation shows an additional 22% overhead in comparison with TCP.

A potentially more useful comparison can be seen in tables 1 and 2, which compares the overhead of CCN to that of HTTP and HTTPS. These tests retrieve a single HTML file (*i.e.,* the result of a single HTTP GET), not the the many different components (*e.g.,* images) that must be retrieved to render most web pages. In Table 1, we show retrieval of Google's home page, and in Table 2, we show retrieval of the larger home page of Wells Fargo over HTTPS. For web retrievals, the limiting factor affecting observed performance is generally the bandwidth of the return path, as that is the direction in which the largest quantity of data is flowing. The upstream bandwidth required for the ACKs, or Interests in the case of CCN, is much lower, so we show it separately. In the first CCN alternative (CCN/UDP column), we used UDP encapsulation with a large block size (7656 bytes), to amortize the cost of CCN names and signatures over a significant amount of content. In the second CCN alternative (CCN/ETH column), we considered direct encapsulation of CCN packets in Ethernet frames. Here we chose a much smaller block size (1230 bytes) so the resulting packets would fit but mostly fill the frames.

The results show that CCN does not necessarily add significant overhead compared to conventional web access for single retrievals, in either round-trips or packet sizes in the all-important data download direction. With UDP encapsulation and a large block size, we avoid overhead associated with a TCP conversation and effectively amortize the cost of signatures and multiple copies of names to achieve a total overhead *lower* than standard HTTP.

7. RELATED WORK

It is widely recognized that combining identity and location information into a single network address is not meeting 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-oriented to content-oriented networking to meet data-intensive application needs.

Previous work on content-oriented networking is dominated by the use of unstructured, opaque, usually self-certifying content labels. The challenge these systems face is first efficiently routing queries and data based on these "flat" names, and second, providing an indirection mechanism to go from user-meaningful names to these opaque labels.

The Data-Oriented Network Architecture [21] 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 be first 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 [6]. A circular namespace is created to ensure correct routing (as in Chord [33]), but additional pointers are added to shorten routes. In a similar approach, i3 [32] 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 [20] 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 [31] 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 insecure 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 betwen user-meaningful names (or currently, even their opaque labels) to content.

The 4WARD NetInf project [26] 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 of API. The NetInf Dictionary infrastructure uses a DHT for name resolution and location lookup.

TRIAD [7], 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 [4] 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 have implemented a prototype CCN network stack, and demonstrated its usefulness for both content distribution and point-to-point network protocols. We intend to release this implementation as open source.[5]

9. REFERENCES

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

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

[3] 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.

[4] 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.

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

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

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

[8] 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.

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

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

[11] 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.

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

[13] IETF. RFC 2328 – OSPF Version 2.

[14] IETF. RFC 3711 – The Secure Real-time Transport Protocol (SRTP).

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

[16] IETF. RFC 3830 – MIKEY: Multimedia Internet KEYing.

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

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

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

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

[21] 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.

[22] J. Kubiatowicz et al. OceanStore: An Architecture for Global-Scale Persistent Storage. *SIGPLAN Not.*, 35(11):190–201, 2000.

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

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

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

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

[27] T. Okamoto and J. Stern. Almost Uniform Density of Power Residues and the Provable Security of ESIGN. In *ASIACRYPT*, volume 2894, pages 287–301, Dec. 2003.

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

[29] B. C. Popescu, M. van Steen, B. Crispo, A. S. Tanenbaum, J. Sacha, and I. Kuz. Securely Replicated Web Documents. In *IPDPS*, 2005.

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

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

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

[33] 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.

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

[5] Estimated release Sept 2009; a definite release date and URL will be in the final version of the paper.