DOI:10.1145/2063176.2063204

# Networking Named Content

By Van Jacobson, Diana K. Smetters, James D. Thornton, Michael Plass, Nick Briggs, and Rebecca Braynard

Current network use is dominated by content distribution and retrieval yet current networking protocols are designed for conversations 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 uses content chunks as a primitive—decoupling location from identity, security and access, and retrieving chunks of content by name. Using new approaches to routing named content, derived from IP, CCN simultaneously achieves scalability, security, and performance. We describe our implementation of the architecture's basic features and demonstrate its performance and resilience with secure file downloads and VoIP calls.

## 1. INTRODUCTION

The engineering principles and architecture of today's Internet were created in the 1960s and 1970s. 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—500EB were created in 2008 alone.[7] 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.

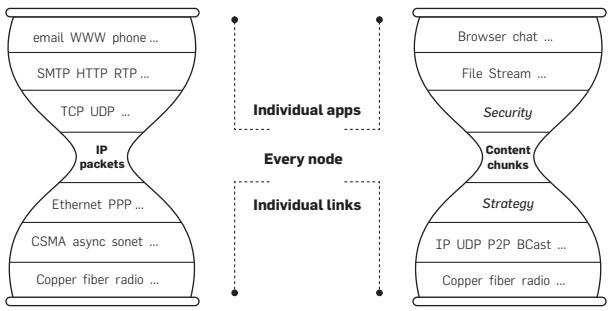
- *Availability*: Fast, reliable content access requires awkward, pre-planned, application-specific mechanisms like CDNs and P2P networks, and/or imposes excessive bandwidth costs.
- *Security*: Trust in content is easily misplaced, relying on untrustworthy location and connection information.
- *Location-dependence*: 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 1960s. 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 4), rather than the connections over which it travels, thereby avoiding many of the host-based vulnerabilities that plague IP networking.

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




A previous version of this paper was published in *Proceedings of ACM's CoNEXT Conference 2009* (Rome, Italy, Dec. 1–4, 2009), ACM, NY.

## 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 interfaces. Any node hearing the `Interest` and having data that satisfies it can respond with a `Data` packet (content chunk). `Data` is transmitted only in response to an `Interest` and consumes that `Interest`.[a] Since both `Interest` and `Data` identify the content chunks 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` (CCN name) in the `Interest` packet is a prefix of the `ContentName` in the `Data` packet. CCN names are opaque, binary objects composed of a (explicitly identified) sequence of components (see Figure 3). 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 ⟨*net, subnet, host*⟩ 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.[b] 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 (e.g., broadcast) environment.[c]

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.[d] The core CCN packet forwarding engine has three main data structures: the `FIB` (Forwarding Information Base), `ContentStore` (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 `ContentStore` 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 after forwarding (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` tracks `Interests` forwarded upstream toward content source(s), so 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 immediately after being used to forward a matching `Data` packet (`Data` "consumes" an `Interest`). `PIT` entries for `Interests` that never find a matching `Data` are eventually timed out (a "soft state" model—consumers are responsible for re-expressing an `Interest` if they still want the `Data`).

**Figure 2. CCN packet types.**

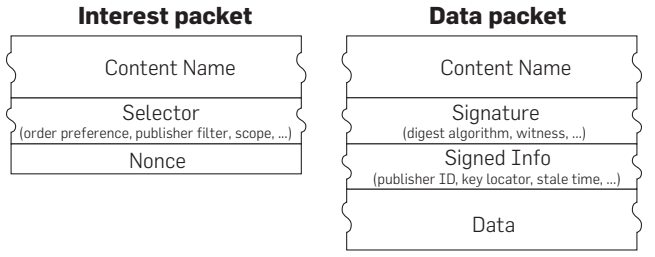
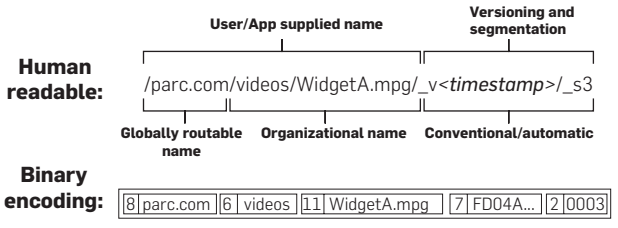



**Figure 3. Example data name.**




[a] `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[8] but, unlike TCP, CCN's model works for many-to-many multipoint delivery (see Section 3.1).

[b] 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.

[c] 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."

[d] 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 5.

When Interest packets arrive on a face, a longest-match lookup is done on the ContentName. (Full details for Interest and Data packet processing are described in Jacobson et al.,[11] we provide a short overview of Interest processing here.) When an Interest arrives on a face, if there is a matching ContentObject in the ContentStore, it will be returned on the same face the Interest arrived on. If there is no ContentObject available, the PIT is checked for an existing pending Interest. If there is already a matching entry, the arrival face for the new Interest is added to the list of faces in the PIT entry used to send ContentObjects to requesters when they arrive. If there is not already an existing PIT entry, the FIB table is checked for forwarding information. If there is a corresponding entry, the Interest is forwarded and added to the PIT.

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.[e] 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 4.1) then added to the ContentStore. A new list of faces is created with the union of the matching PIT entry face lists minus the Data packet arrival face. The Data packet is then sent out each face on this new list.

The multipoint nature of Interest-based data retrieval 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. The Interest/Data exchange also functions whenever there is local connectivity. For example, two colleagues with laptops and ad hoc wireless could share corporate documents normally in an isolated location with no connectivity to the Internet or their organization.

## 3. TRANSPORT AND ROUTING

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 not satisfied in some reasonable period of time must be retransmitted. Unlike TCP, CCN senders are stateless and the final consumer is responsible for reexpressing unsatisfied Interests if it still wants the data. A receiver's strategy layer 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.

### 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 changing the number of Interests it issues. We show the effect of such pipelining later in Section 5.2. 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 operates 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.[8] 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 congestion control techniques 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 link FIFO queues but rather LRU memory (cache) decoupling hop-by-hop feedback control loops and damping oscillations.

### 3.2. Sequencing

TCP conversations between hosts identify data by simple sequence numbers. CCN needs something more sophisticated because consumers are requesting individual content chunks 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. 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,

[e] "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.

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 3, 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.[f]

An `Interest` can specify precisely what content chunk 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) and thus relations like *next* and *previous* can be unambiguously interpreted by the CCN transport without any knowledge of name semantics.

### 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 (e.g., IP address) to a layer 2 identity such as a MAC address. Even when connectivity is rapidly changing, CCN exchanges packets 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 5.3).[g]

### 3.4. 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 needed 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 where peer identities and locations are not known. CCN's robust information security model enables almost automatic routing infrastructure protection.

## 4. CONTENT-BASED SECURITY

CCN is built on the notion of *content-based security:* protection and trust are properties of the content itself, not of the connections over which it travels. Current IP networks trust content based on the server id and the connection type used for retrieval. This intertwines content and network infrastructure security, thereby forcing clients to retrieve content from the original source to trust it. In CCN all content is digitally signed and private content is encrypted. Embodying security in content, not hosts, reduces the trust burden on network intermediaries, opening the network to wide participation.[h] This section gives an overview of CCN's core security design. Additional background and motivation are described in Smetters and Jacobson.[19]

### 4.1. Content validation

Rich and robust content-based security models can be constructed if a consumer can assess just three things about any content object[2, 5, 18]: *integrity* (is the object intact and complete), *pertinence* (what question does it answer), and *provenance* (who claims this is an answer). TCP/IP's a 16-bit checksum provides explicit, though weak, integrity, but pertinence and provenance are, at best, implicit in the source and destination addresses. CCN's pertinence is explicit in the `ContentName` and provenance explicit in the key used to sign the packet ("signed info" in Figure 2). Thus the packet's signature both checks integrity and authenticates the binding between name, key, and content. Standard public key signatures are used, so anyone, not just the communication endpoints, can validate them.

Consumers generally validate content in the context of an application and user dependent trust model (Section 4.2). Anything can validate that a `Data` was signed by the key it claims was used to generate its signature, even without attaching any real-world meaning to that key. This minimal verification is useful to detect packet corruption and to defend against network attacks. For example, consumers can request content by publisher as well as by name in the face of spurious or malicious data. To prevent work-factor attacks, no node is obligated to validate every `Data`.

[f] The digest component is not transmitted since it is derivable. It exists so that an `Interest` or a `ContentName` can unambiguously and exactly name any content chunk.

[g] 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.

[h] CCN signing and encryption may require consumers to retrieve keys. For TCP/IP this requires side information and special purpose application protocols but in CCN keys are simply `ContentObjects` like everything else and, in general, the mechanism and network state required to get a content chunk are sufficient to also get all of its keys.

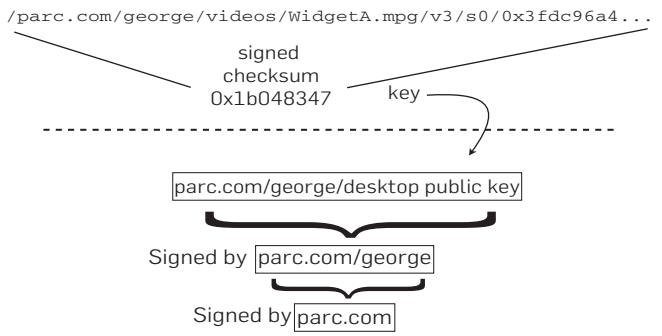
In particular, CCN content routers may choose to check all, some or none, as their resources allow, and dynamically adapt in response to detected attack or consumer advice.

### 4.2. Managing trust

Although CCN moves data in a peer-to-peer fashion, it provides *end-to-end* security between content publisher and consumer. Consumers determine whether received content is acceptable or trustworthy. CCN's trust is *contextual*, narrowly determined in the context of particular content and the purpose for which it will be used. Different consumers can even use different trust models for the same content. For example, a bank might require that a deed be signed by someone authorized by the courts while a backup server might only care that its signature validates. This model is more flexible and scalable than mandating a one-size-fits-all approach such as trusting any publisher who has paid an annual fee to some root certifying authority.

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 `Data`. For example, when a name refers to an individual or organization and the content is a public key, the result is essentially a digital certificate. This allows CCN to support traditional mechanisms such as hierarchical PKI for establishing trust in keys. CCN also supports more general, non-hierarchical models such as SDSI[2, 5, 18] where keys are mapped to identities via locally controlled *namespaces*. For example, the members of an organization might be recognized because their keys are certified by the organization itself, not because they are validated by a third-party like Verisign. If we know and trust one of parc.com's employees, we get their key any time we receive content from them and by following its key locators (certification chain) can easily get the key of parc.com (Figure 4). Thus, starting from a small number of public keys authenticated using a variety of user-friendly mechanisms (e.g., personal contact, organizational membership, public experience[16, 20]), one can use SDSI's model to infer trust in a large number of publishers.

**Figure 4. CCN trust establishment can associate content namespaces with publisher keys.**




*Evidence-based security*. The CCN content model creates many secured relationships: Each piece of data is bound to its name and publisher's key; names are implicitly bound to other names by the naming hierarchy and application-level naming conventions; publishers are bound to other publishers by the key certification graph. As a consumer accumulates content, this web of trusted relationships can be used to automatically infer trust in entire collections of content. For example, once trust has been established for the block of video shown in Figure 4, that decision can be applied to any block of any version of /parc.com/george/videos/WidgetA.mpg signed by the same key. Similarly, a block of the video signed by a different key is automatically suspicious.

This notion can be extended beyond collections defined by naming hierarchy. As Section 3.2 notes, the full name of a `ContentObject` contains a cryptographic digest of its contents (a *self-certifying name*[4, 6, 14, 15]) and the identity (key) of its publisher. Embedding a link using the full name of the target `ContentObject` creates a *secure reference*[4, 12, 13, 17] that can resolve to only one object. 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 $N'$. They can also be used to build up a network of trust in content where signed `Data` certify the other content chunks they securely reference. For example, having decided to trust the content of web page $A$, browsers may automatically trust the images, ads, etc., that $A$ securely links to. This trust is fine-grained since the linked content is only considered valid within the context of $A$.

### 4.3. 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 may require that some name components remain in the clear). Decryption keys are distributed as `ContentObjects`. Name conventions, encapsulated in programmer-friendly libraries, make it easy to find and decrypt the key needed by 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.

### 4.4. Network security and denial-of-service

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. It is difficult to target malicious packets at host vulnerabilities since CCN `Interests` address content, not hosts. Effective attacks against CCN must use denial of service to either hide legitimate content or drown it in a sea of spurious packets.

TCP/IP content originates at some server and an attacker can easily hide it by placing a filter anywhere on the single

shortest path between the targeted set of clients and that server. CCN content can be supplied by anything that has a copy and every CCN node can use any and all of its interfaces simultaneously to locate and retrieve a copy. Thus hiding content is exponentially more difficult for the attacker. It must establish a filtering perimeter around its targets that covers all paths to all possible copies of the consent. Any copy that makes it through the perimeter immediately becomes a new source that will virally propagate the content to all interested clients.

Drowning (DDoS) attacks can be mounted against sources of CCN content but are substantially more difficult than they are with TCP/IP. The flow balance between `Interest` and `Data` prevents any sort of `Data` flooding, so attackers must attack via `Interest` packets. Say an attacker marshals a horde of zombies to simultaneously generate interests in some `ContentName`. If they all use the same name, the `Interests` will be aggregated (at most one pending `Interest` in a name is ever forwarded over any link) and no flood will result. So they must all use different names under the targeted prefix. If the different names refer to actual `ContentObjects`, those objects will be cached everywhere along the paths from the content source(s) to the zombies; thus the flood near the source will quickly clear as `Interests` are satisfied by downstream cached copies of the `Data`.[i] If the zombie's names are randomly generated then their `Interests` will never be satisfied by a matching `Data` and will time out. Thus every intermediate node learns that many bogus `Interests` are being generated for the targeted prefix. Nodes can decide to temporarily rate limit such `Interests` (similar to the push-back strategy used against TCP/IP DDoS) or simply prioritize them lower than `Interests` that are resulting in `Data` responses.[j] In either case the effect of the attack on legitimate traffic is minimized.

### 4.5. Policy controls

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. For example, PARC might have a "content firewall" that only allows `Interests` from the Internet to be satisfied if they are requesting content under the /parc.com/public namespace. An organization could also publish its policies about what keys can sign content under a particular name prefix (e.g., PARC could require that all content in the /parc.com namespace be signed by a key certified by a /parc.com root key), 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 what namespaces or how often particular signers may query.

## 5. 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 segmentation and versioning in names and 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™, Solaris™, FreeBSD, NetBSD, and Microsoft Windows™. Cryptographic operations are provided by OpenSSL and Java.

### 5.1. 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 "fills the pipe" because its variable window size allows for enough data in transit to fill the bandwidth × delay product of the path plus all of the intermediate store-and-forward buffer stages.[9] 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.

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 100 Mbps 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.[k] This resulted in 6278 individually named, signed CCN `Data` packets (`ContentObjects`) each with 1KB of data (the resulting object sizes were around 1350 bytes).

[i] This does result in a distributed cache poisoning attack that must be addressed in the CCN node's cache replacement policy.

[j] A CCN content router has a limit on the number of pending `Interests` it will allow on any link (generally related to the bandwidth × delay product of that link). The router can choose if it wants to hold or discard arriving `Interests` over the limit and how it selects `Interests` up to the limit.

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

[l] 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.

**Figure 5. Bulk data transfer performance.**

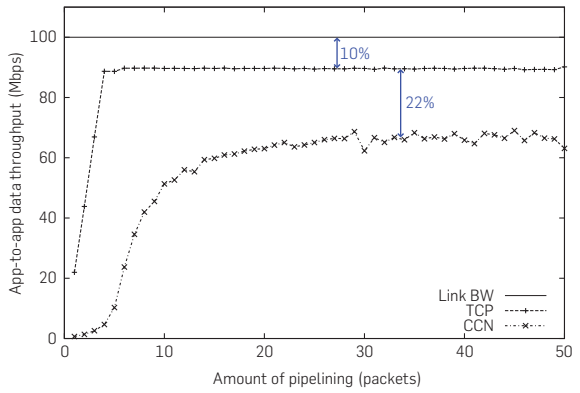
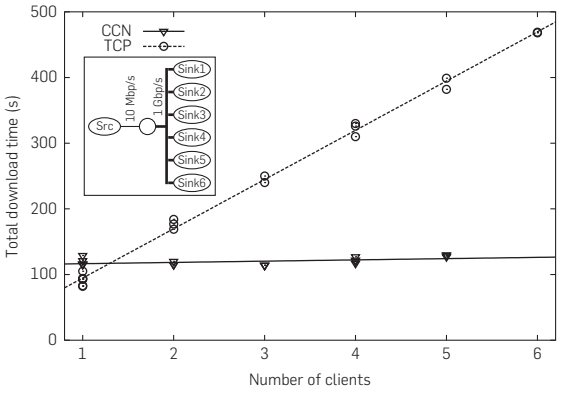



**Figure 6. Total transfer time vs. the number of sinks.**




Results can be seen in Figure 5.[l] 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 user-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 this test encapsulates CCN in IP/UDP, 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.[m]

### 5.2. 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 6 and consisted of a source node connected over a 10 Mbps shared link to a cluster of six sink nodes all interconnected via 1 Gbps links.[n] 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 5.1. 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 6. 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.

### 5.3. 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 Jacobson et al.[10] 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

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

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

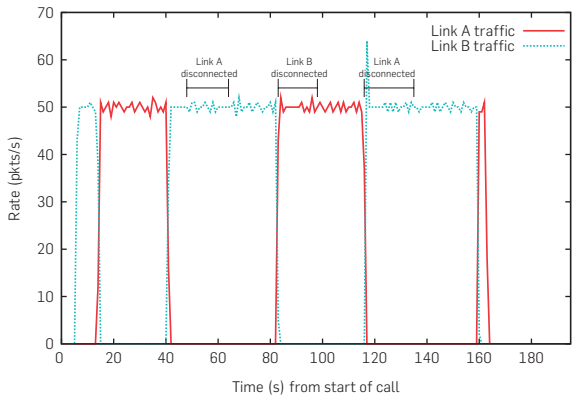
exclusively for the prefix's `Interests`, until it is time for the next experiment (e.g., after 200 packets, when there is a change in carrier or SSID, or when an `Interest` does not get a response and times out).

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 20 ms 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 few VoCCN packets (<0.1%) were dropped for arriving too late.

We conducted failover tests by manually disconnecting and reconnecting network cables. Figure 7 shows the traffic on both links during one of these tests. The strategy layer initially picks link `B`, but at 15 s 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 s. At 45 s we unplugged link `A` but this had no impact since link `B` was being used at the time. At 60 s link `A` was reconnected. At 82 s 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 s link `B` is reconnected; traffic remains on link `A`. At 120 s 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 s then the call terminates at 165 s.

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 retrieves the missing data from CCN: a few packets were delayed but none were lost.

**Figure 7. CCN automatic failover.**




## 6. CONCLUSION

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 Project CCNx™.[1]

## Acknowledgments

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 discussion and advice. C

**References**

1. Project CCNx™. http://www.ccnx.org, Sep. 2009.
2. Abadi, M. On SDSI's linked local name spaces. *J. Comput. Secur. 6*, 1–2 (Oct. 1998), 3–21.
3. Adamson, B., Bormann, C., Handley, M., Macker, J. *Multicast Negative-Acknowledgement (NACK) Building Blocks*. IETF, Nov. 2008. RFC 5401.
4. Clarke, I., Sandberg, O., Wiley, B., Hong, T.W. Freenet: A distributed anonymous information storage and retrieval system. In *Lecture Notes in Computer Science 2009* (2001), 46.
5. Ellison, C.M., Frantz, B., Lampson, B., Rivest, R., Thomas, B.M., Ylonen, T. *SPKI Certificate Theory*, Sep. 1999. RFC2693.
6. Fu, K., Kaashoek, M.F., Mazieres, D. Fast and secure distributed read-only file system. *ACM Trans. Comput. Syst. 20*, 1 (2002), 1–24.
7. Gantz, J.F. et al. IDC—The expanding digital universe: A forecast of worldwide information growth through 2010. Technical report, Mar. 2007.
8. Jacobson, V. Congestion avoidance and control. In *SIGCOMM*, 1988.
9. Jacobson, V., Braden, R., Borman, D. *TCP Extensions for High Performance*. IETF—Network Working Group, The Internet Society, May 1992. RFC 1323.
10. Jacobson, V., Smetters, D.K., Briggs, N., Plass, M., Stewart, P., Thornton, J.D., Braynard, R. VoCCN: Voice-over content-centric networks. In *ReArch*, 2009.
11. Jacobson, V., Smetters, D.K., Thornton, J.D., Plass, M., Briggs, N., Braynard, R. Networking named content. In *CoNext*, 2009.
12. Koponen, T., Chawla, M., Chun, B.-G., Ermolinskiy, A., Kim, K.H., Shenker, S., Stoica, I. A data-oriented (and beyond) network architecture. In *SIGCOMM*, 2007.
13. Kubiatowicz, J. et al. OceanStore: An architecture for global-scale persistent storage. *SIGPLAN Not. 35*, 11 (2000), 190–201.
14. Mazières, D., Kaminsky, M., Kaashoek, M.F., Witchel, E. Separating key management from file system security. In *SOSP*, 1999.
15. Moskowitz, R., Nikander, P. *Host Identity Protocol Architecture*. IETF—Network Working Group, May 2006. RFC 4423.
16. Osterweil, E., Massey, D., Tsendjav, B., Zhang, B., Zhang, L. Security through publicity. In *HOTSEC '06*, 2006.
17. Popescu, B.C., van Steen, M., Crispo, B., Tanenbaum, A.S., Sacha, J., Kuz, I. Securely replicated web documents. In *IPDPS*, 2005.
18. Rivest, R.L., Lampson, B. SDSI—A simple distributed security infrastructure. Technical report, MIT, 1996.
19. Smetters, D.K., Jacobson, V. Securing network content.. PARC Technical Report, Oct. 2009.
20. Wendlandt, D., Andersen, D., Perrig, A. Perspectives: Improving SSH-style host authentication with multi-path probing. In *USENIX*, 2008.

**Van Jacobson** (van@parc.com), PARC, Palo Alto, CA.

**James D. Thornton** (jthornton@parc.com), PARC, Palo Alto, CA.

**Michael Plass** (plass@parc.com), PARC, Palo Alto, CA.

**Nick Briggs** (briggs@parc.com), PARC, Palo Alto, CA.

**Rebecca Braynard** (rbraynar@parc.com), PARC, Palo Alto, CA.

**Diana K. Smetters** (smetters@alum.mit.edu), Now at Google. Work was done while at PARC.