Networking Named Content

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

ABSTRACT

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

Categories and Subject Descriptors

C.2.1 [**Computer Systems Organization**]: Network Architecture and Design; C.2.2 [**Computer Systems Organization**]: Network Protocols

General Terms

Design, Experimentation, Performance, Security

1. INTRODUCTION

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

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

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

- **Availability**: Fast, reliable content access requires awkward, pre-planned, application-specific mechanisms like CDNs and P2P networks, and/or imposes excessive bandwidth costs.
- **Security**: Trust in content is easily misplaced, relying on untrustworthy location and connection information.
- **Location-dependence**: Mapping content to host locations complicates configuration as well as implementation of network services.

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

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

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

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

CoNEXT'09, December 1–4, 2009, Rome, Italy.

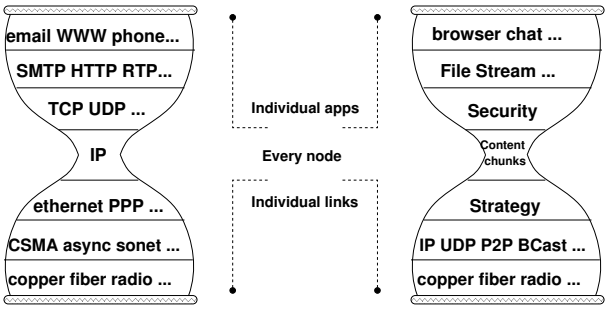


Figure 1: CCN moves the universal component of the network stack from IP to chunks of named content.

2. CCN NODE MODEL

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

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

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

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

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

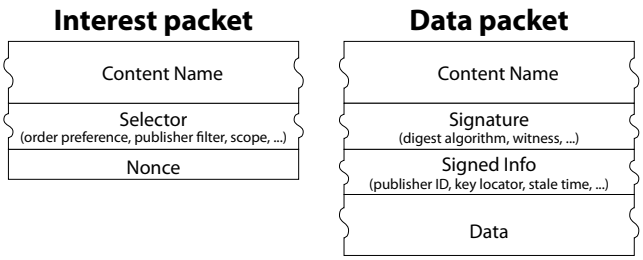


Figure 2: CCN packet types

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

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

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

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

When an Interest packet arrives on some face, a longest-match lookup is done on its ContentName. The index structure used for lookup is ordered so that a ContentStore match will be preferred over a PIT match which will be preferred over a FIB match.

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

Otherwise, if there is an exact-match PIT entry the Interest's arrival face will be added to the PIT entry's RequestingFaces list and the Interest will be discarded. (An Interest in this data has already

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

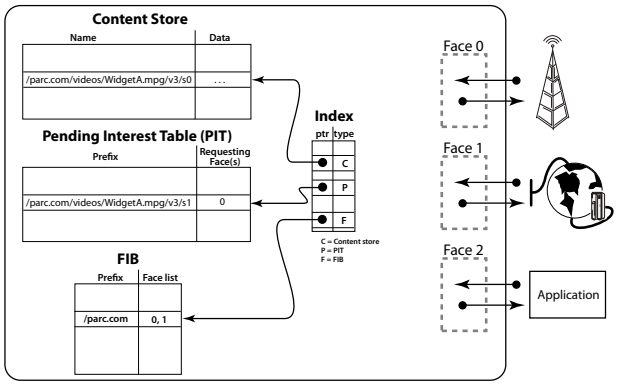


Figure 3: CCN forwarding engine model

been sent upstream so all that needs to be done is to make sure that when the Data packet it solicits arrives, a copy of that packet will be sent out the face that the new Interest arrived on.)

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

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

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

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

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

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

3. TRANSPORT

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

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

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

3.1 Reliability and Flow Control

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

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

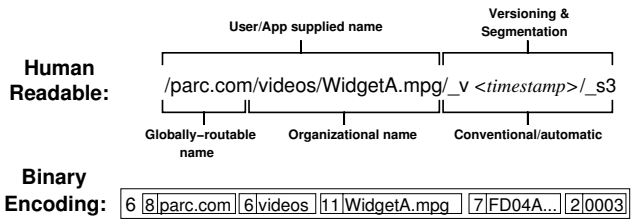


Figure 4: Example Data name

3.2 Sequencing

In a TCP conversation between hosts, data is identified by simple sequence numbers. CCN needs something more sophisticated because consumers are requesting individual pieces from large collections of data and many recipients may share the same Data packets. Locating and sharing data is facilitated by using hierarchical, aggregatable names that are at least partly meaningful to humans and reflect some organizational structure of their origin, rather than just the sequence in an ephemeral conversation. Despite this extra richness in CCN names, their transport function in Interests is exactly the same as that of sequence numbers in TCP ACKs: specifying the next Data the recipient requires.

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

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

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

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

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

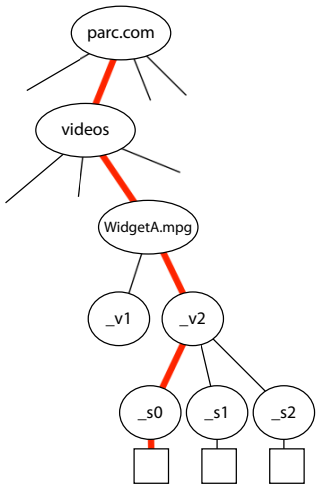


Figure 5: Name tree traversal

LeftmostRightSibling annotation or by simply computing the `_s1` portion of the name since the segmentation rules are known (and determined) by the application.

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

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

3.3 Rich Connectivity, Mobility and Strategy

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

As described in Section 2, CCN explicitly models multiple connectivity via per-FIB-entry face lists. Since there is no one-size-

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

fits-all strategy for using multiple faces, the design intent is for each CCN FIB entry to contain a program, written for an abstract machine specialized to forwarding choices, that determines how to forward Interests. The 'instructions' for this machine should include a small subset of the normal load/store, arithmetic, and comparison operators plus *actions* that operate on sets of faces such as *sendToAll, sendToBest, markAsBest*, and *triggers* such as *interestSatisfied, interestTimedOut, faceDown* that can be used to invoke lists of actions when significant events occur. Faces will have an (open-ended) set of *attributes* such as *BroadcastCapable, isContentRouter, UsageBasedCharging, PeakUseLimited* that can be used to dynamically construct the sets for use by the actions.

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

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

4. ROUTING

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

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

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

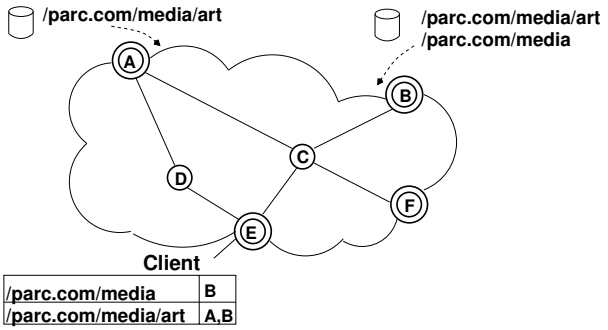


Figure 6: Routing Interests to a domain's media content

4.1 Link-state Intra-domain Routing

Intra-domain routing protocols provide a means for nodes to discover and describe their local connectivity ('adjacencies'), and to describe directly connected resources ('prefix announcements') [17, 16]. These two functions are orthogonal—one describes links in the graph while the other describes what is available at particular nodes in the graph. It is common for these two functions to be performed in completely different information domains. For example, IS-IS [17] describes adjacencies in terms of IEEE 802.1 layer 2 MAC addresses but announces layer 3 IP4 and/or IP6 prefixes. As described in Section 2, IP forwarding and CCN forwarding are almost identical. They both use prefix-based longest match lookups (and use them for the same reason—hierarchical aggregation of detail) to find local neighbor(s) 'closer' to the identifier matched. Given the similarities of the two FIBs, one might suspect that the distributed routing machinery used to create IP FIBs might be easily adapted to create CCN FIBs. This is indeed the case.

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

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

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

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

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

4.2 Inter-domain Routing

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

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

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

5. CONTENT-BASED SECURITY

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

5.1 Content Validation

CCN authenticates the *binding* between names and content; the signature in each CCN data packet (Figure 2) is over the name, the content, and a small amount of supporting data useful in signature verification ("signed info" in Figure 2). This allows content publishers to securely bind arbitrary names to content. In contrast,

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

many previous approaches require names to be *self-certifying* to securely name content (*e.g.,* by using the cryptographic digest of the content as its name [26, 28, 12, 9]). The ability to directly, and securely use user- or application-meaningful names enhances usability and eases transport. Systems without it require an "indirection infrastructure" [4, 6] to map from the names humans care about to secure, opaque, self-certifying names. The security of the resulting system is then limited to the security of the (often unsecured) indirection infrastructure.

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

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

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

5.2 Managing Trust

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

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

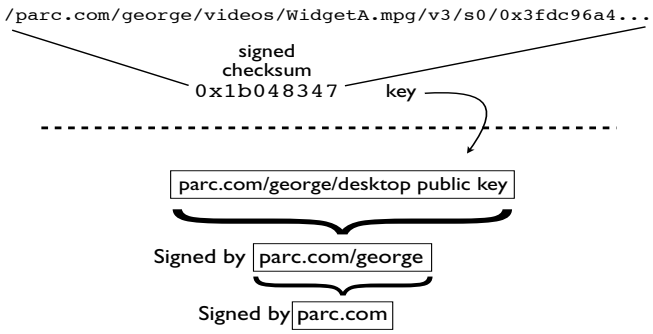


Figure 7: CCN trust establishment can associate content namespaces with publisher keys.

5.2.1 Trusting Keys

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

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

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

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

We can map SDSI identities directly into CCN names, and express SDSI trust relationships directly in CCN content. Such namespaces make up a *forest-of-trees* – a content consumer might trust that they have the right key for `parc.com` (or for `/parc.com/george`) for any number of reasons from direct experience (*e.g.,* they are a PARC employee), to information provided by friends,

to its presence in a trusted directory of keys. It is not required, or even expected, that all such trees will be joined in a single (or small number of) root(s) as happens in traditional global or commercial PKIs. Most notably, it is the *consumer* who decides why they trust a particular key, using many types of information, not the publisher in obtaining a certificate from a particular vendor.

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

5.2.2 Evidence-Based Security

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

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

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

5.3 Content Protection and Access Control

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

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

5.4 Network Security and Policy Enforcement

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

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

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

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

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

CCN also provides tools that allow an organization to exercise control over where their content will travel. Routers belonging

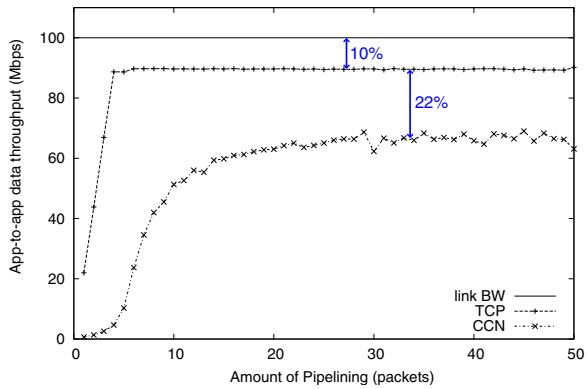


Figure 8: Bulk data transfer performance

to an organization or service provider can enforce *policy-based routing*, where content forwarding policy is associated with content name and signer. One simple example is a "content firewall" that only allows Interests from the Internet to be satisfied if they were requesting content under the `/parc.com/public` namespace. An organization could publish its policies about what keys can sign content under a particular name prefix (*e.g.,* all keys signed by `ccnx.org`), and have their content routers automatically drop content that does not meet those requirements, without asking those routers to understand the semantics of the names or organizations involved. Finally, Interests could in certain cases be digitally signed, enabling policy routing to limit into what namespaces or how often particular signers may query.

6. EVALUATION

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

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

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

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

6.1 Data Transfer Efficiency

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

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

Table 1: Web content efficiency

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

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

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

The results are summarized in Table 1. The first two columns give the total number of bytes and packets sent and received by

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

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

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

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

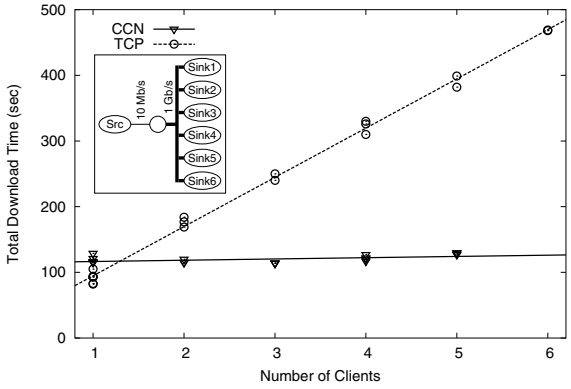


Figure 9: Total transfer time vs. the number of sinks.

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

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

6.2 Content Distribution Efficiency

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

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

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

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

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

6.3 Voice-over-CCN and the Strategy Layer

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

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

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

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

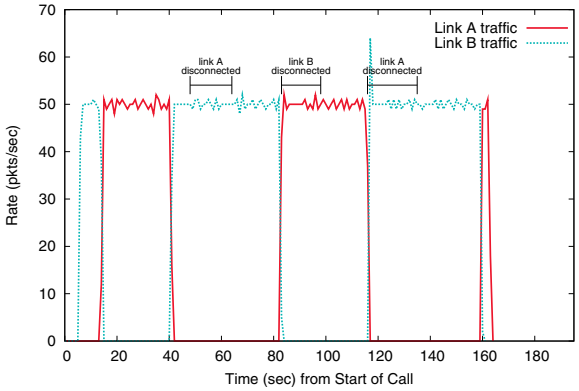


Figure 10: CCN automatic failover.

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

7. RELATED WORK

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

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

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

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

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

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

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

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

8. CONCLUSIONS

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

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

Acknowledgements

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

9. REFERENCES

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

[16] IETF. RFC 2328 – OSPF Version 2.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

用P4对数据平面进行编程

作者：尼克·麦克欧文(Nick McKeown)[1,3]
金昶勳(Changhoon Kim)[2,3]
[1]斯坦福大学
[2]Barefoot Networks
[3]P4.org
译者：高荣新(Ron Kao)

关键词：P4 语言联盟　可编程数据平面

引言

软件定义网络因其使网络拥有者和运营商能够对网络行为进行编程而取得了巨大的成功。然而，其可编程性目前仅局限于网络控制平面，其转发平面在很大程度上受制于功能固定的包处理硬件。P4语言联盟 (www.P4.org) [1] 及其开源活动旨在完全摆脱网络数据平面的束缚，让网络拥有者、工程师、架构师及管理员可以自上而下地定义数据包的完整处理流程。

灵活的网络数据平面将加速网络和计算在不同子领域的创新。事实上，这是计算机历史上几次相同技术变革模式的又一次重现：廉价的可编程器件的出现必然会推动其上层应用的创新和发展，给业界带来新一轮革新。

在网络领域，这个故事才刚刚开始，并在新一代高性能可编程数据包处理芯片的助力下成为可能。在高性能领域，内置协议无关的交换架构 PISA 芯片能提供每秒 Tb[2] 级别的数据包处理速度，以及完全可编程的数据包解析与通用的“匹配 - 动作”能力。在中低性能领域，服务器级或嵌入式的中央处理器 (CPU)、图形处理器 (GPU)、现场可编程门阵列 (FPGA) 和网络处理器 (NPU) 早已能够提供每秒几十到几百 Gb 速度级别的数据包处理速度以及灵活的处理过程。

除了可编程转发芯片，我们还需要“P4”(www.p4.org)[1,3] 这样的高级语言，以不受限于具体目标设备（目标无关）的方式控制转发行为。程序员首先用 P4 定义数据包的处理流程，然后利用编译器在不受限于具体协议（协议无关）的交换机或网卡上生成具体的配置，从而实现用 P4 表达的数据包处理逻辑。程序员通过编程，可以将交换机变为一个架顶交换机 (Top-Of-Rack, TOR)、一道防火墙或一个负载平衡器，或者支持新的自动诊断功能和新的拥塞控制算法等。

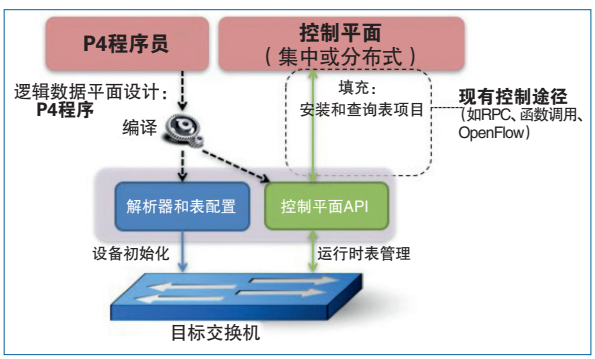


图1　P4是编程交换机数据平面的语言

图 1 显示了 P4（编程交换机数据平面，从而说明数据包如何被处理）与控制平面应用程序（在运行时生成转发表）之间的关系。需要注意的是，通

过编译 P4 程序，不仅可以在可编程的数据包处理硬件上生成具体的配置，还可以生成运行时应用编程接口，来帮助控制和数据平面之间的交互。

本文将从比较抽象的层面介绍 P4 语言和可编程数据平面技术。如想了解更多技术细节，建议从《P4 的语言规范》[4]（最初是 2014 年在 *SIGCOMM CCR* 上发表的 P4 论文 [3]）和《开源 P4 开发工具》[5] 入手。

可编程数据平面将达到什么目标?

可编程数据平面将带来广泛而巨大的影响。我们预计几乎所有的网络领域参与者都将从中获益。

可编程数据平面有助于网络系统供应商进行更快速的迭代开发，迅速推出新的功能，甚至直接通过打补丁修复现有产品中发现的数据平面程序漏洞。数据平面的灵活性使得各个供应商能够在系统功能和性能上差异化，以有别于其他厂商。最根本的是，这些系统供应商可以从软件产业过去几十年已发展成熟的软件编程理论、实践和工具中受益。

可编程数据平面也可以帮助网络拥有者（例如在线服务提供商、运营商和企业）实现最适合其自身需求的具体网络行为。此外，许多大型运营商拥有的大量软件开发人员能够轻松学会对数据平面的网络设备进行编程、测试和调试，将网络变成一个可编程的平台，以一个完全可编程的方式来管理网络。例如，他们可以开发自定义的网络监控、分析和诊断系统，实现前所未有的网络可视化和相关性，从而大幅度降低网络的运营成本。他们还可以协同优化网络以及在网络上运行的应用程序，来保证最佳的用户体验。

对于网络芯片供应商，可编程数据平面使他们能专注于设计并改进那些可重用的数据包处理架构和基本模块，而不是纠缠特定协议里错综复杂的细节和异常行为。而且，一旦证明这些架构和基本模块可行，供应商就可以在多代交换芯片的设计中重复使用它们，不必为客户不断产生的新需求而反复修改。

对于网络和分布式系统的研究人员，可编程的数据平面为他们实现并验证新想法（尤其是那些引入新的数据平面行为的项目）提供了新的契机。因为只有“合适”的硬件才能真正应对实际部署中所有具有挑战性的需求，如高端口速度、全线速发送小数据包、低延迟、高端口数等。

可编程的数据平面技术具有如下优点：

新功能 可以快速和频繁地开发新的网络功能，比如新的自定义功能或标准的数据包头规范和转发行为。

降低复杂性 可以去掉冗余的而只保留网络必需的数据平面功能。例如，大多数大型数据中心网络通常只使用L2 & L3转发、ECMP（等价多路径）、LAG（链路聚合）和 ACL（访问控制列表）。其他复杂的协议，比如 MPLS（多协议标签交换）、QoS（服务质量）和 IP 多播等，都是多余的负担。在设备的数据和控制平面中仅保留有用的协议，可以减少潜在的漏洞、被攻击的可能性和运维的复杂性。

有效利用资源 通过去掉不必要的功能，可以释放所耗费的物理资源（例如内存、纵横式交换矩阵和计算逻辑单元），然后重新分配给必要的功能，以实现最大的效能。

增强可视化 当前，只具有固定功能的数据包处理硬件并且能够生成的网络监控信息的数量和质量都非常有限。这是因为监测、分析和诊断等功能往往被认为是一种事后的补救措施，所以其优先级总是低于数据包转发功能。一个无法根据特定协议转发数据包的交换机芯片在市场上是毫无竞争力的，而一个能转发数据包但只提供较少或低质量检测信息的交换机芯片在市场上则有一定的竞争力。因此，考虑到要在有限的硬件资源（如尺寸和功耗）上实现各种数据平面的功能，固定功能硬件的设计者要始终优先设计转发功能，而后才是监控功能。有了可编程数据平面，网络拥有者可以控制转发和监控功能之间的平衡，因为其真正想要的转发功能往往只是硬件厂商提供的功能列表中的一个很小的子集。此外，网络拥有者还可以明确地定义所需的监控功能的语义，以保证监控信息的相关性。

模块化 我们可以通过重用别人开发的 P4 代码库来实现自己想要的转发行为。这将大大简化和方便个人用户的 P4 开发进程。

可移植性 当开发了一个 P4 程序后，我们可以为不同目标硬件重新编译代码，将此程序重复用于不同类型的目标上。这可以显著提升编程、测试和调试整个网络行为的效率，因为我们只要在所有的目标设备上使用相同的 P4 程序就可以保证数据和控制平面之间接口的统一。因此，控制平面可以被重复使用，而无须或只须最少的改动。

拥有自己的知识产权 P4 程序员不需要与硬件厂商分享转发逻辑（功能）。这使得开发一个 P4 程序能得到相应的回报，从而激发网络拥有者们自主创新的热情。

协议无关的交换机架构

可编程的数据平面技术具有很多优点，但所有这些都是基于廉价的可编程数据包处理器。虽然中低端性能的数据包处理器——CPU、现场可编程门阵列和网络处理器——早已存在，并已被用来实现具有复杂数据平面功能的网络系统，但这还远远不够。如果要想具备和现有的功能固定的数据包处理硬件相似的性价比，我们还需要实现高性能的系统芯片级可编程数据包处理器。

网络硬件行业的最新技术趋势表明，实现这样的硬件是可行的[2,6,7]，并且相对于固定功能设计，可编程性带来的额外开销可以做到足够低。这种数据包处理器的一种通用架构是协议无关的交换机架构 (Protocol-Independent Switch Architecture, PISA)。

图 2 展示了 PISA 的高层架构。PISA 是一种具有完全可编程数据包解析器和通用“匹配 - 动作”单元的高速数据包处理器。关于可编程解析器的技术在近期已有广泛研究[8]，且最早的 P4 论文中也提供了一个案例来说明如何使用三元匹配 (ternary matching) 能力实现有限状态机。通用的“匹配 - 动作”逻辑是通过静态随机存储器 (SRAM)、三态内容寻址存储器 (TCAM)、寄存器、灵活的哈希 (Hash) 生成器和支持通用数据包处理指令集的计算逻辑单元来实现的。博斯哈特 (Bosshart) 等人最近提出了一种通用“匹配 - 动作”逻辑的详细设计[2]。PISA 中有大量“匹配 - 动作”单元，且所有这些单元都是相同的。这种同质化使得 PISA 成为一个极佳的编译器目标，编译器可以灵活编程每个“匹配 - 动

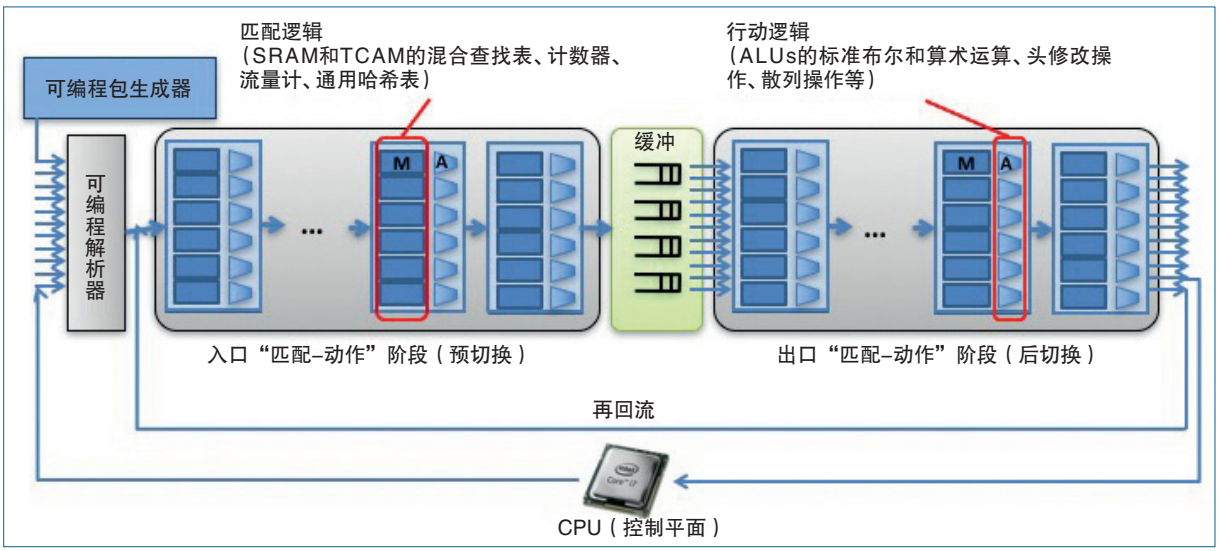


图2 协议无关的交换机架构。到达PISA交换机的数据包先由可编程的解析器解析，再通过入口侧一系列的“匹配-动作”阶段，然后经由队列系统交换，由出口“匹配-动作”阶段再次处理，最后重新组装发送到输出端口

作”单元来实现需要的数据包处理功能。需要注意的是，解析器和“匹配 - 动作”单元必须在被编程后才能做具体的数据包处理工作；实际上，没有配置的 PISA 芯片不会实现任何数据平面协议。这就是为什么我们称之为协议无关架构。

PISA 从两个并行的维度保证高速率的数据包处理。第一，多个数据包处理阶段以流水线的方式工作。也就是说，每一个阶段可以相互独立地处理少量的数据包，因此在特定时间内多个数据包可以在流水线的不同阶段同时存在，从而提高芯片的处理速度。由于这些阶段都循序安排，PISA 还可以处理彼此依赖的数据包任务（如表格查找和包头修改）。例如，假设用户希望实现的是一个包头字段的先写后读的操作，编译器可以定义写的操作给 i 阶段的“匹配 - 动作”单元，另外定义“读”的操作到 i+1 阶段的单元。第二，在每个阶段，有大量的“匹配 - 动作”单元。这让编译器可以分配独立(即“可并行”) 的数据包处理操作到同一阶段的“匹配 - 动作”单元，同样提高了处理速度。

除了可编程解析器和通用的“匹配 - 动作”流水线，PISA 还有一些有价值的基本模块。经流水线修改过的数据包头在传输到输出端口之前需要重新组合。这个逆向解析过程同样受益于灵活性，因此 PISA 引入了可编程的逆解析操作。为了提高可编程性，PISA 还提供了一条回流路径使一些特殊数据包能够被多次反馈到解析器和流水线。编译器也可以使用这个回流功能来实现传统芯片因只有固定的查找修改单元数量而无法完成的复杂逻辑。PISA 还可以嵌入可编程数据包生成器，使 CPU（控制平面）可以将频繁或周期性的数据包生成操作转移到数据包生成器。特定的包头值或实例的出现也可以触发数据包生成，从而进一步丰富可编程性。这与一个通用 CPU 事件驱动的处理能力（例如中断）相似。PISA 还提供了连接数据和控制平面之间的高带宽通道。

P4语言简介

要编程 PISA 和其他类型的可编程数据包处理器，需要网络工程师和运营商所熟悉的领域和一种专用的编程语言。P4 就是这样一种语言，其目标如下：

协议无关性 网络设备不应该与任何特定的网络协议捆绑在一起。相反，对于任何所需要的网络数据平面协议和数据包处理行为，程序员都可以使用 P4 进行表达。

目标无关性 程序员无须关心底层硬件细节，即可描述数据包处理功能。

现场可重配置能力 程序员应在部署交换机后能对数据包的处理方式进行再次修改。

P4程序定义

P4 程序主要由如下组件构成：

包头 定义描述一系列字段的顺序和结构，包括字段的宽度和字段值的限制。

解析器 定义如何识别数据包内的包头和有效的包头序列。

表 “匹配 - 动作”表是执行数据包处理的机制，P4 程序定义了在表内可以匹配的字段和可以执行的操作。

动作 基于一系列预先定义的和协议目标无关的简单基本操作，P4 可以构建复杂的自定义动作，并且可以在“匹配 - 动作”表中使用。

控制程序 决定以什么顺序用“匹配 - 动作”表处理数据包，一个简单的命令式程序就可以描述“匹配 - 动作”表之间的控制流。

以下是关键组件的一些简单实例。

包头类型实质上是由成员字段组成的有序列表，每个字段有其名称和长度。每一种包头类型都有对应的包头实例来存储具体的包头数据。P4 也允许建立模型将数据包元数据（metadata，任何不包含在该数据包内，而是衍生自数据包包头或底层硬件的数据）作为包头类型。下面是以太网和 IPv4 包头类型及其实例。

```
header_type ethernet_t {
  fields {
    dst_addr : 48; // width in bits
    src_addr : 48;
```

```
        ethertype : 16;
    }
}
header ethernet_t ethernet;
header_type ipv4_t {
    fields {
        version : 4;
        ihl : 4;
        diffserv : 8;
        totalLen : 16;
        identification : 16;
        flags : 3;
        fragOffset : 13;
        ttl : 8;
        protocol : 8;
        hdrChecksum : 16;
        srcAddr : 32;
        dstAddr: 32;
    }
}
header ipv4_t ipv4;
```

一旦定义了包头实例，即可表达数据包的解析逻辑：硬件将输入的字节序列转换为有效的包头实例集合（也称为数据包的解析后表达）的过程。有限状态机的概念很适合模拟该过程，所以 P4 借用了这个概念。在有限状态机的模型下，底层的数据包处理器经过多个解析状态，从预先定义的启动状态开始，在预先定义的终止状态结束。下面的例子说明了如何描述以太网和 IPv4 包头的解析逻辑。

```
parser start {
    return parse_ethernet;
}
parser parse_ethernet {
    extract(ethernet)
    return switch(latest.ethertype) {
        case 0x8100: parse_vlan;
        case 0x0800: parse_ipv4;
        // Other cases
    }
}
parser parse_ipv4 {
    extract(ipv4);
    return select(latest.protocol) {
        IP_PROTOCOLS_IPHL_ICMP :parse_icmp;
        IP_PROTOCOLS_IPHL_TCP :parse_tcp;
        IP_PROTOCOLS_IPHL_UDP :parse_udp;
        IP_PROTOCOLS_IPHL_GRE :parse_gre;
        IP_PROTOCOLS_IPHL_IPV4 : parse_ipv4_in_ip;
        IP_PROTOCOLS_IPHL_IPV6 : parse_ipv6_in_ip;
        // Other cases
        default: ingress;
    }
}
```

P4 的表和动作是“匹配 - 动作”过程模型中的关键概念。P4 表是一个包含很多条目的逻辑结构。包头字段（如匹配键）与表中的条目按照与包头字段相关联的特定匹配语义进行匹配。匹配的语义包括完全匹配、三元匹配、最长前缀匹配和范围匹配。一经匹配或失配（没有匹配项），就执行一个与数据包及相关元数据对应的动作。

在 P4 中，所有动作都是用户自定义的。一个动作其实就是一个自定义的执行序列，包含若干个基本模块指令，配合在运行时由控制平面传递给动作的参数并以此来处理包头字段或元数据。P4 语言规范 [4] 中定义了标准的基本模块指令集，而下层目标设备还可以提供更多的指令。下面是如何定义一个 IPv4 路由表和相关的动作来实现 IPv4 路由的示例。

P4 语言规范规定的基本模块指令集包括：数据包处理运算符（如添加、删除或修改包头）、基本的算术运算符、哈希运算符和统计跟踪运算符（如计数、测量）。

```
#define IPV4_LPM_TABLE_SIZE 65536
table ipv4_fib_lpm {
    reads {
        metadata.vrf : exact;
        ipv4.dstAddr :lpm;
```

```
    }
    actions {
      on_miss;
      fib_hit_nexthop;
    }
    size : IPV4_LPM_TABLE_SIZE;
  }
  action on_miss() {
      // no op
  }
  action fib_hit_nexthop() {
    modify_field(metadata.fib_hit, TRUE);
    modify_field(metadata.fib_nexthop, nexthop_
index);
    add_to_field(ipv4.ttl, -1);
  }
```

一旦定义好包头、解析器、表和动作，剩下的任务则是指定从一个表到下一个表的控制流。控制流是一个由条件语句和表的引用组成的命令式程序。下面是如何实现一个自定义的控制流的例子：数据包首先经过 L2 转发表 (12_fwd)，然后可能经过 L3 路由表（ipv4_fib_lpm 和 ipv6_fib_lpm），这取决于数据包的以太网目的地址是否匹配路由器自身的 MAC 地址（通过查找所属路由器的 router_mac 表）。根据 IP 包头类型（IPv4 或 IPv6），数据包将经过不同的 L3 路由表。最后采用访问控制列表来决定是否允许数据包通过。

```
control ingress {
  // L2 forwarding
  apply(l2_fwd);
  // L3 routing
  apply(router_mac) {
    hit {
      if (valid(ipv4)) {
        apply(ipv4_fib_lpm);
      } else {
        if (valid(ipv6)) {
          apply(ipv6_fib_lpm);
        }
      }
    }
  }
  // ACL
  apply(acl);
}
```

编译器功能

P4 编译器本质上是将在 P4 程序中表达的数据平面的逻辑翻译成一个在特定可编程数据包处理硬件上的具体物理配置。因此，编译器后端部分自然与其支持的硬件目标紧密结合，而其前端部分则可以在各个 P4 可编程目标之间通用。这就意味着一个 P4 程序的具体实现可根据被编译的目标而改变。

PISA 编译器可以从 P4 程序的包头和解析器定义中导出解析器和重组器的配置。P4 采用的状态机概念使得此映射变得相对容易。PISA 编译器可以从表、动作和控制流的定义中导出“匹配 - 动作”阶段的配置。编译器首先分析每个包头字段、元数据和状态对象在表和动作之间的所有依赖关系。基于这个结果，识别出可放置在相同阶段同时运行的表和动作，以及那些由于依赖性而必须顺序执行的表和动作。同时编译器还应考虑到其他由特定目标带来的相关限制，如可用的表内存、计算逻辑单元和携带数据包包头的寄存器等。

阿尼路达 (Anirudh) 等人最近推出了一些可重用的技术，用于分析由状态对象（寄存器）造成的复杂的依赖关系，介绍了如何将一系列指令映射到不同功能的计算逻辑单元上 [9]。拉万亚 (Lavanya) 等人的文章介绍了如何将所需的表放置在有资源限制的特定目标设备上，同时使这些表能够符合其 P4 程序中所描述的依赖关系 [10]。

现状和展望

P4 自首次发表 [3] 以来就引起了工业界和学术界的极大关注。业内专家和主要学术团队受其启发

组建了 P4 语言联盟 [1]，并正在推广该语言，以期形成全行业通用的数据平面编程框架 [4]。目前已有许多具体的 P4 案例研究和语言演进的技术评论，发表在相关论文和报告 [11~13] 中。

P4语言联盟

P4 语言联盟是一个开源社区，由工业界和学术界成员组成。它有两个目标：(1) 定义 P4 语言的正式规范；(2) 维持开放源码的 P4 开发工具和 P4 的参考程序 [5]。所有资料都在非常宽松的 Apache 2.0[14] 许可下通过 www.P4.org 免费使用和重新发布。

P4 语言联盟通过语言设计工作组产生正式的语言规范，组织成员的代表参与工作组的活动。当前得到广泛支持的语言规范是 1.0.2 版本。P4 语言联盟已经用此版本发布了一个 P4 参考程序（取名为“switch.p4”），能实现各种流行的标准数据平面协议和功能，包括 L2 和 L3（IPv4 和 IPv6）转发、虚拟局域网 (VLAN)、生成树协议 (STP)、等价多路径、链路聚合、虚拟路由和转发 (VRF)、IP 组播、多协议标签交换、各类隧道协议（如虚拟可扩展局域网、通用路由封装、IP-in-IP 和 Q-in-Q）、数据包镜像、服务质量控制、访问控制、RPF 验证、传输协议（TCP、UDP 等），甚至还有一个 OpenFlow 功能的数据平面。尽管具有如此丰富的功能 [1]，截止到 2016 年 3 月，switch.p4 只有约 7000 行的 P4 代码。这表明，P4 在可读性和抽象性之间达到了很好的平衡，这对网络工程师和网络系统设计者极具吸引力。阿尼路达等人描述了在原有的 P4 论文基础上，如何增加语言结构以实现许多有实际应用的数据平面协议 [11]。

最新的预发布的语言规范为 1.1.0 版 [15]，相对 1.0.2 版本增加了以下功能：

强类型 语言中的所有数据类型、有效的运算符和操作规则都已明确定义，动作参数的类型亦经明确分类以避免歧义。

表达式 最新版本支持通用（一元、二元、三元、按位运算及逻辑运算）运算符组成的表达式。这使得语言有更强的表达能力且使用方便，无须支持更多的基本模块操作。

顺序执行语义 在 1.0.2 版本，用户自定义的动作是用并行执行的语义来解释的，这使得 P4 的开发者很难推理动作的实际行为。而 1.1.0 版本改用顺序执行语义，但这并不妨碍在实际数据包处理的时候并发执行用户定义的动作。我们应该让编译器，而不是 P4 的开发者，利用 P4 程序和硬件的能力并行执行用户定义的动作。

除了维护语言规范和开源参考实例外，P4.org 还举办各种活动（如 P4 教程 [16]、研讨会 [17,18] 和编程马拉松）以推广 P4 并支持 P4 的开发社区。

开发环境

由 P4.org 发布的开源 P4 开发工具包括参考编译器、软件交换机（称为行为模型）、参考程序以及测试和调试框架 [5]。在掌握了这些工具以后，开发者就可以编写、编译自己的 P4 程序，使用行为模型来运行程序，并用测试框架对其进行测试。测试框架既支持单一设备，也支持使用 Mininet[2] 进行网络级测试 [19]。

在基于 P4 的数据平面编程里，一个微妙而重要的方面是生成运行时 API 的方式。对于传统的固定功能包转发设备来说，每个设备都有其自己的运行时 API（常被误称为软件开发工具包 (SDK)）。这些 API 本质上是一些非常复杂的库，是设备供应商手工编写的，并且只适用于该供应商的设备。这些 API 由于来自不同设备供应商而缺乏一致性和互相兼容性，妨碍了我们在网络里部署异构（不同厂商）的数据包转发设备。

对于 P4 来说，API 是由编译器在编译时根据一个给定的 P4 程序自动生成的。因此，P4 程序的开发者也完全定义了运行时 API。这种做法有一个重要的好处：跨平台可移植性。只要网络拥有者为其

[1] 有关支持该参考程序的最新协议和功能的列表，请访问https://github.com/p4lang/switch/tree/master/p4src。

[2] Mininet是一个轻量级软件定义网络和测试平台。

设备使用相同的 P4 程序，这些设备产生的 API 就是相同的。所以 P4 的应用程序可以横跨各类型的 P4 可编程设备，实现可移植性。

最近已经推出多个 P4 可编程的目标设备，还有更多即将出台。一些厂商已经展示了其基于 FPGA 或基于 GPU 的 P4 可编程网卡 [20]。P4 也可以作为一个领域专用语言来扩展基于软件的可编程的转发设备。OVS、eBPF 和 DPDK 都是很好的例子 [21,22]。如果程序员想要修改这些目标以增添新的数据平面协议，他们需要具备丰富的软件开发经验，熟悉大型的、复杂的代码库。P4 能大幅度地降低这类任务的复杂度，因为 P4 可以让程序员只关注网络协议设计，而不用去理解、扩展和测试一个大型而复杂的代码库，比如内核网络栈。这种方法也使得基于软件的包处理器的部署和维护更加方便。例如，在 OVS 中实现一个新的数据平面协议可能需要修改 Linux 内核中的代码，而使用 P4 可编程的 OVS，可以将特定协议的实现和 OVS 的底层实现完全脱离。

语言演化路线

随着 P4 被广泛应用于编程各种类型的可编程数据包处理器，我们确定了两个额外的目标：兼容异质性和代码重用。

兼容异质性包含功能异质性和架构异质性两个方面。功能异质性源于不同类型的数据包处理器的不同功能，如校验、验证和更新、先进的模式匹配、加密 / 解密、状态操作等。架构异质性源于不同类型的数据包处理器的不同设备架构。

语言设计者所面临的挑战是：以前后一致和统一的方式解决这一异质性挑战，同时保持语言核心部分的简洁。语言设计工作组一直在探索一些新的语言结构（如 extern type 和架构模型）来解决这个问题。最新预发布的 P4 规范的附录（第 17.7 节）详细解释了这些概念 [15]。

代码重用对于建设一个繁荣的语言生态系统，以及提升该语言的整体开发效率至关重要。我们确定了提高 P4 重用性的三个具体子目标：**可移植性、架构和语言的分离、组合性**。

研究问题

随着 P4 带来了可编程的数据平面，许多令人兴奋的研究课题也随之产生：

- P4 应该如何演变？对于“语言演化路线”一节提出的问题，如何找到正确的方向来解决，这将是一个重要的研究贡献。
- 如何测试两个 P4 程序在编译前后的等效性？
- 给定一个 P4 程序，不论表目是否被填充，能否自动地产生一组完备的测试用例，来确认它具有某些性质？
- 如何证明 P4 程序的正确性？
- 能否直接从更高级别的网络策略产生 P4 程序？
- 利用可编程数据平面，能实现什么样新颖的网络应用程序？在近期的论文和讲座中已经出现了一些出色的和具有广阔前景的应用，包括先进的网络监控和分析 [23~25]、把中间层功能嵌入到交换机 [26]、通过在网络里的处理来协助分布式应用 [27,28]、协同设计网络数据平面和在其上运行的分布式应用程序 [29]。我们相信，学术界和产业界已开始触及由可编程数据平面技术带来的无穷可能。

结论

我们都知道如何在 CPU 上编程，使之易于创建新的程序，试验新的想法，构建新的应用，并与他人分享。但今天，常见的不只是可编程的 CPU，还有 GPU、智能手机的应用处理器，甚至是数字信号处理器 (DSP)。程序员用高级语言描述所期望的行为，然后编译到领域专用的处理器上运行。我们已经多次在计算机历史上看到这种事例，而在网络领域里这样的故事才刚刚开始。

在 P4 和新兴的可编程数据包处理硬件（如 PISA）的基础上，P4 语言联盟提出要开发更灵活的交换机，其功能在生产环境中可被指定或改变。程序员可以决定转发平面如何处理数据包，而无须担心实施细节。编译器将命令式的程序转化为表依赖关系图 (table dependence graph)，而表依赖关系图可

以映射到许多具体的目标交换机，包括优化的硬件实现。

我们预测网络在未来几年内将成为一个可编程平台，重现已经发生在计算和存储行业的演化模式。这种趋势将为网络拥有者、经营者和研究人员提供前所未有的灵活性和可视性，为网络控制、管理和服务的创新提供重要的研发机会。

我们大力鼓励研究界为 P4.org 贡献全新的和改进的技术、验证和软件。P4 将很好地帮助研究人员参与数据包处理逻辑的相关研究，而他们无须将工作和一个特定的专用芯片捆绑起来，也无须花费大量精力去学习错综复杂的 API。■

致谢：

感谢白巍、李宇亮、李小舟、陈力、成建晖、李少波在本文中文版的翻译中给予的专业意见。

作　者：

尼克・麦克欧文(Nick McKeown)

斯坦福大学教授。主要研究方向为互联网架构、软件定义网络、可编程转发平面、网络中立性、拥塞控制、计算机科学教育。nickm@stanford.edu

金昶勳(Changhoon Kim)

Barefoot Networks公司系统架构总监。主要研究方向为可编程网络数据平面、网络监控和诊断、网络验证、自编程/配置网络和大规模分布式系统的调试与诊断。chang@barefootnetworks.com

译　者：

高荣新

从事网络芯片开发30余年。目前在Barefoot Networks公司负责亚太事务。ron@barefootnetworks.com

参考文献

[1]P4 Language Consortium. http://p4.org.

[2]P. Bosshart, G. Gibb, H.-S. Kim, and et al.. Forwarding metamorphosis: Fast programmable match-action processing in hardware for SDN, in ACM SIGCOMM, 2013.

[3]P. Bosshart, D. Daly, G. Gibb, et al.. P4: Programming protocol-independent packet processors, ACM SIGCOMM Computer Communication Review, vol. 44, no. 3, 87~95, 2014.

[4]The P4 Language Specification. http://p4.org/wp-content/uploads/2015/04/p4-latest.pdf, Mar. 2015. Version 1.0.2.

[5]P4 Open Source Repositories. https://github.com/p4lang.

[6]R.Ozdag, White paper- intel ethernet switch fm6000 series - software defined networking, 2012. Intel Corporation.

[7]XPliant packet architecture press release. http://cavium.com/newsevents-Cavium-and-XPliant- Introduce-a-Fully-Programmable-Switch-Silicon- Family.html, 2014.

[8]G. Gibb, G. Varghese, M. Horowitz, and N. McKeown, Design principles for packet parsers, in ANCS, 13~24, 2013.

[9]A. Sivaraman, M. Budiu, A. Cheung, C. Kim, S. Licking, G. Varghese, H. Balakrishnan, M. Alizadeh, and N. McKeown, Packet transactions: High-level programming for line-rate switches. http://arxiv.org/abs/1207.0016, Jan. 2016.

[10]L. Jose, L. Yan, G. Varghese, and N. McKeown, Compiling packet programs to reconfigurable switches, in *12th USENIX Symposium on Networked Systems Design and Implementation* (NSDI 15), 103~115, 2015.

[11]A. Sivaraman, C. Kim, R. Krishnamoorthy, A. Dixit, and M. Budiu, Dc. p4: programming the forwarding plane of a data-center switch, in *Proceedings of the 1st ACM SIGCOMM Symposium on Software Defined Networking Research*, p. 2, ACM, 2015.

[12]C. Kim, Programming the Network Dataplane using P4. http://schd.ws/hosted_files/p4workshop2015/4b/ChangK-P4-Workshop-June-04-2015.pdf, July 2015.

[13]C. Kim, P4 Status Update: Where are we now and what's next? http://schd.ws/hosted_files/2ndp4workshop2015/33/ Chang%2C%20P4%20Workshop%20Nov%2018%202015.pdf, Nov. 2015.

[14]Apache License version 2.0. http://www.apache.org/licenses/LICENSE-2.0.

[15]The P4 Language Specification – pre-release. http://p4.org/wp- content/uploads/2016/03/p4_v1.1.pdf, Jan. 2016. Version 1.1.0.

[16]P4 Tutorial at SIGCOMM'15.http://conferences.

sigcomm.org/sigcomm/2015/tutorial- p4.php.

[17]1st P4 Workshop. http://p4.org/p4/1st-p4-workshop-on-june-4-2015/.

[18]2nd P4 Workshop. http://p4.org/p4/2nd-p4-workshop-on-november-18-2015/.

[19]Mininet – An Instant Virtual Network on Your Laptop. http://mininet.org/.

[20]OpenFlow's Possible Successor P4, Gets Into Hardware. https://www.sdxcentral.com/articles/news/p4- openflows-possible-successor-gets-into-hardware/2015/11/.

[21]P4 and OpenvSwitch. http://p4.org/p4/p4-and-open-vswitch/.

[22]D. Daly, P4 Applied to a vSwitch Data Plane. http://schd.ws/hosted_files/p4workshop2015/67/DanD-P4-Workshop-June-04-2015.pdf.

[23]Improving Network Monitoring and Management with Programmable Data Planes. http://p4.org/p4/inband-network-telemetry/.

[24]C. Kim, A. Sivaraman, N. Katta, A. Bas, A. Dixit, and L. J. Wobker, In-band network telemetry via programmable dataplanes, in ACM SIGCOMM Industrial Demo, 2015.

[25]Y. Li, R. Miao, C. Kim, and M. Yu, Flowradar: A better netflow for data centers, in 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), (Santa Clara, CA), 311~324, USENIX Association, 2016.

[26]Adding L4 Load-balancing to Every Switch. http://p4.org/wp-content/uploads/2015/12/USC- Barefoot-Demo-Poster.pdf.

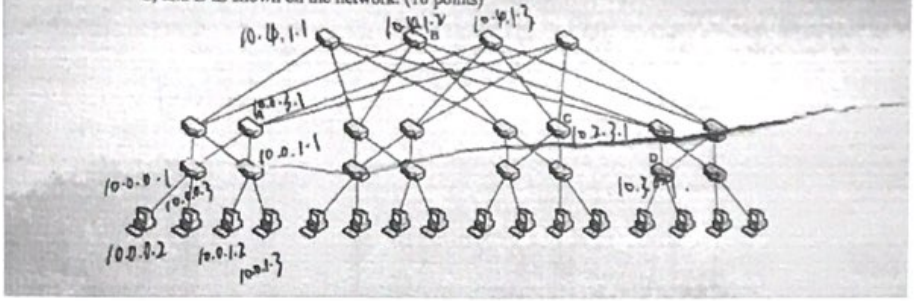
[27]D. R. K. Ports, J. Li, V. Liu, N. K. Sharma, and Krishnamurthy, Designing distributed systems using approximate synchrony in data center networks, in 12th USENIX Symposium on Networked Systems Design and Implementation (NSDI 15), (Oakland, CA), 43~57, USENIX Association, 2015.

[28]H. T. Dang, M. Canini, F. Pedone, and et al.. Paxos made switch-y, CoRR, vol. abs/1511.04985, 2015.

[29]X. Li, R. Sethi, M. Kaminsky, D. G. Andersen, and M. J. Freedman, Be fast, cheap and in control with switchkv, in 13th USENIX Symposium on Networked Systems Design and Implementation (NSDI 16), (Santa Clara, CA), 31~44, USENIX Association, 2016.

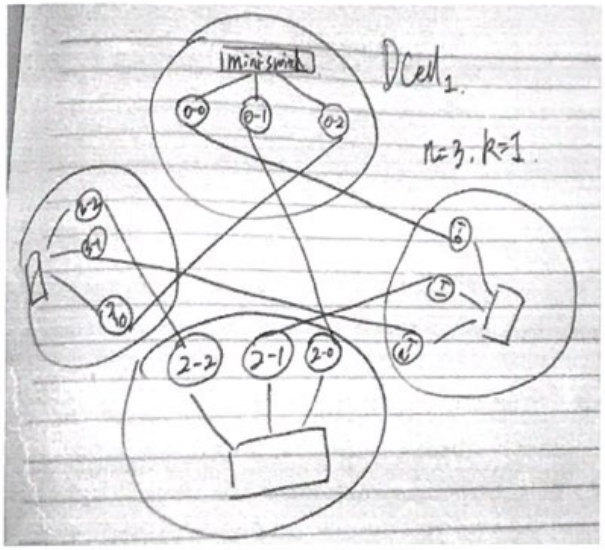
1. Which one is NOT the desired property of the datacenter network? (5 points)
   (1) Scalable interconnection bandwidth;
   (2) Revolutionary clean-slate architecture;
   (3) Economies of scale;
   (4) Backward compatibility

2. According to the addressing scheme of FatTree, give the IP addresses for the switches of A, B, C, and D as shown on the network. (10 points)



A: 10.0.3.1
B: 10.4.1.2
C: 10.2.3.1
D: 10.3.0.1

3. Draw a DCell network structure with n=3, k=1, and name each node.(10 points)



4. How many layers are there in a software defined network? Where is the OpenFlow located? List at least three OpenFlow flow entry actions, and for each action, explain how the OpenFlow switch will handle a packet that matches the entry? (15 points)

1. Three layers
   - application layer
   - control layer

- infrastructure layer

2. Between the control layer and the infrastructure layer
3. three basic actions:
   - Forward this flow's packets to a given port (or ports).
   - Encapsulate and forward this flow's packets to a controller.
   - Drop this flow's packets.

如果匹配OpenFlow flow entry，并且action是转发，则按照entry转发该flow，如果action是drop，则drop该flow，如果不匹配entry，则压缩并转发至控制器，一般情况是转发该flow的第一个packet，也有转发所有packet的情形。

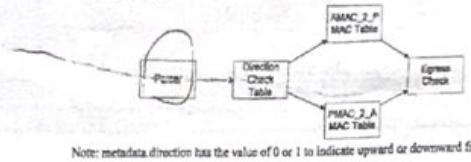
这个第四问的最后一个解释

5. Fill the blanks in the script written in the Ethane policy language, so that:
   Rule 1: Members in "students" cannot communicate with each other; (5 points)
   Rule 2: Members in "students" can communicate with members in "tutors" using protocol of SSH; (5 points)
   Rule 3: Members in "tutors" can communicate with each other; (5 points)
   Rule 4: Members in "students" can use HTTP through the "http-proxy" device. (5 points)

```
# Rule 1
[(usrc=in("students")^(udst=in("students"))] : deny;
# Rule 2
[(usrc=in("students")^(protocol="ssh")^(udst=in("tutors"))] : allow;
[(usrc=in("tutors")^(protocol="ssh")^(udst=in("students"))] : allow;
# Rule 3
[(usrc=in("tutors")^(udst=in("tutors"))] : allow;
# Rule 4
[(usrc=in("students"))^(protocol="http)] : waypoints("http-proxy");
```

6. In PortLand, edge switches are responsible for translating AMAC to PMAC for the upward Ethernet frames and translating PMAC to AMAC for the downward Ethernet frames. We seek to implement PortLand with P4, following is the flow control chart, please fill the blanks in the amac_to_pmac and pmac_to_amac action specifications, and write out the skeleton of the other parts of the P4 program, including the packet parser, the table specification, and the control program. (25 pt.)

Flow control chart:



Note: metadata.direction has the value of 0 or 1 to indicate upward or downward frames respectively.

Action specifications

```
action set_direction(direction){
    //Set the metadata direction
    set_field(metadata.direction, direction);
}
```

```
action amac_to_pmac(ethernet, src_pmac, dst_pmac, egr_spec){
    //Translate the address
    copy_field(_____);
    copy_field(_____);
    //Set the destination egress port
    set_field(metadata.egress, egr_spec);
}
```

```
action pmac_to_amac(ethernet, src_amac, dst_amac, egr_spec){
    //Translate the address
    copy_field(_____);
    copy_field(_____);
    //Set the destination egress port
    set_field(metadata.egress, egr_spec);
}
```

```
action_amac_to_pmac(…){
    …
    copy_field(ethernet.src_adrr,src_pmac)
    copy_field(ethernet.dst_adrr,dst_pmac)
    …
}
action_pmac_to_amac(…){
    …
    copy_field(ethernet.src_adrr, src_amac)
    copy_field(ethernet.dst_adrr ,dst_amac)
    …
}
```

Packet Parser（不一定对）

```
parser start {
    ethernet;
}

parser ethernet {
    switch(ethertype) {
        case 0x8100: vlan;
        case 0x9100: vlan;
        case  0x800: ipv4;
        // Other cases
    }
}
```

```
parser vlan {
    switch(ethertype) {
        case 0xaaaa: mTag;
        case  0x800: ipv4;
        // Other cases
    }
}

parser mTag {
    switch(ethertype) {
        case 0x800: ipv4;
        // Other cases
    }
}
```

Table Specification（不一定对）

```
table mTag_table {
    reads {
        ethernet.dst_addr : exact;
        vlan.vid : exact;
    }
    actions {
        // At runtime, entries are programmed with params
        // for the mTag action.  See below.
        add_mTag;
    }
    max_size : 20000;
}

table source_check {
    // Verify mtag only on ports to the core
    reads {
        mtag : valid; // Was mtag parsed?
        metadata.ingress_port : exact;
    }
    actions { // Each table entry specifies *one* action

        // If inappropriate mTag, send to CPU
        fault_to_cpu;

        // If mtag found, strip and record in metadata
        strip_mtag;

        // Otherwise, allow the packet to continue
        pass;
    }
    max_size : 64; // One rule per port
}
```

```
table local_switching {
    // Reads destination and checks if local
    // If miss occurs, goto mtag table.
}

table egress_check {
    // Verify egress is resolved
    // Do not retag packets received with tag
    // Reads egress and whether packet was mTagged
}
```

Control Program

```
control main() {
    // Verify mTag state and port are consistent
    table(source_check);

    // If no error from source_check, continue
    if (!defined(metadata.ingress_error)) {
        // Attempt to switch to end hosts
        table(local_switching);

        if (!defined(metadata.egress_spec)) {
            // Not a known local host; try mtagging
            table(mTag_table);
        }

        // Check for unknown egress state or
        // bad retagging with mTag.
        table(egress_check);
    }
}
```

补充（不用写）：

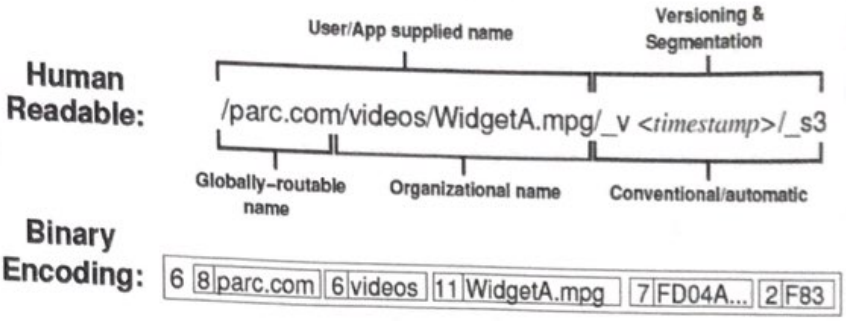
```
header ethernet {
    fields {
        dst_addr : 48; // width in bits
        src_addr : 48;
        ethertype : 16;
    }
}

header vlan {
    fields {
        pcp : 3;
        cfi : 1;
        vid : 12;
        ethertype : 16;
    }
}
```

```
header mTag {
    fields {
        up1 : 8;
        up2 : 8;
        down1 : 8;
        down2 : 8;
        ethertype : 16;
    }
}
```

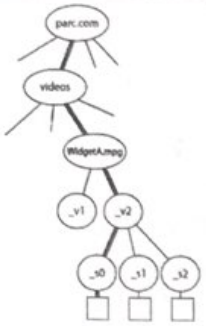
7. How CCN names a content Data? On CCN network, when you receive a Data with a name of "cn/edu/ustc/cs/tian/exampaper.doc/v2/page1/line10", how can you validate the data? (15 points)

Transport: Naming

- CCN is based on hierarchical, aggregatable names at least partly meaningful to humans
- The name notation used is like URI



Transport: Naming



- An Interest can specify the content exactly
- Content names can contain automatically generated endings used like sequence numbers
- The last part of the name is incremented for the next chunk (e.g. a video frame)
- The names form a **tree** which is traversed in preorder
- In this way, the receiver can ask for the next *Data* packet in his *Interest* packet