CCNx 1.0 Protocol Introduction

Marc Mosko[1]*

Abstract
The CCNx 1.0 protocols deliver an efficient, flexible Information Centric Networking implementation. This paper describes the CCNx 1.0 protocols, from the wire format up through the message semantics. We detail protocol use cases and many examples.

Keywords
Content Centric Networks

[1] *Palo Alto Research Center*
***Corresponding author**: marc.mosko@parc.com

Contents

Introduction

CCNx 1.0 protocols use protocol layering to deliver efficient, flexible Information Centric Networking (ICN). The core protocol uses two message types: Interest and Content Object. An Interest message requests a Content Object from the network and is forwarded along routing paths. A Content Object is a chunk of authenticated user data sent along the Interest reverse path, consuming the state left by the Interest. This document describes these messages in detail and specifies how nodes process them. We give several examples of how to build useful services using Interest and Content Object exchanges, pointing out the advantages to this form of transport over traditional UDP or TCP.

A CCNx network has Content Producers, Content Publishers, Content Consumers, Caches, and Forwarders. Each of these actors fulfills specific roles. In our use cases, we describe how each of them interacts with Interest and Content Objects.

A Content Producer creates user data. It could be an individual taking photos, a sensor producing a reading, or a media corporation creating movies. A Content Publisher converts user data into Content Objects, with an associated cryptographic identity. A publisher may pre-produce the content objects, such as pre-coding a movie, or run an on-demand service such as an on-line merchant. A Content Consumer uses the CCNx 1.0 protocols to retrieve user data, authenticated by the publisher.

The two key network elements in CCNx are the forwarder and in-network caches. The forwarder executes the core protocols to forward Interests and Content Objects according to the core protocol rules and the routing tables. A forwarder may run additional protocols, depending on its role in the network. LAN routers and edge routers will run service and content discovery protocols. Core routers will typically run only the core protocols. In-network caches temporarily store Content Objects. LAN routers or edge routers may have large caches,

populated opportunistically or pre-populated, while core routers typically do not use opportunistic caching. In this document, we discuss the so-called Content Store, which is an optional part of every CCNx forwarder. A network operator may use other forms of in-network caching governed by different rules than the forwarder's Content Store.

1. Names, Content Publishers and Routing

A core feature of CCNx is that units of data, called Content Objects, have user-given names. The name serves two purposes: the first parts of the name match routing tables to seek out the Content Objects, and the later parts of the name identify a particular Content Object along that routing. In many respects, a CCNx name looks like a routable URI. A Content Object has two additional identifiers: a KeyId and a Content Object Hash. The KeyId identifies the publisher's signing key, and thus cryptographically identifies the publisher. If an Interest carries a KeyId in addition to a Name, it limits the universe of Content Objects that match the name to those published by that specific publisher. The Content Object Hash is a cryptographic hash of the entire Content Object. If used in an Interest, in addition to a name, it selects one specific Content Object of that name.

1.1 Names

A CCNx name is an absolute URI without an authority. Each URI path segment has a label and a value. The label identifies the purpose of the name component, such as a general name component used for routing, or a specialized component used to sequence numbers, timestamps, or content chunk numbers. There are also application-specific labels.

(1) `lci:/Name=foo/Name=bar/SerialNumber=7/ChunkNumber=30`

For example, in Name 1, the prefix `lci:/Name=foo/Name=bar` corresponds to a user-given name. The path segment `SerialNumber=7` indicates the revision of the document using a serial number. The path segment `ChunkNumber=30` indicates that the user data was split into multiple chunks, and this name identifies chunk number 30 in the series.

1.2 Publishers and Routing

A Content Publisher advertises its authoritative namespace via routing [1]. The routing protocol applies policies and security mechanisms to ensure that a publisher only advertises authorized namespaces. Service providers may, additionally, limit the locations from which a publisher may advertise their service.

A home user, for example, will have a peering relationship with their ISP. They might have a personal namespace, or more likely a namespace subordinate to their ISP. The personal namespace routes traffic to the home user, such that the home user could receive Interest messages.

A business would have a well-known namespace, similar to how it has a DNS name. It may very well have two or more well-known names, where one name points towards "cloud" content hosted off-site and another points to local names such that users at the business site can receive Interest messages. The local name may be tied to the business' ISP.

A large business or network service provider would have well-known names that are multiply hosted throughout the network.

1.3 Working with collections

A publisher creates a single content object, named for example `/foo/manifest` that names a consistent collection. For example, it could say to use the collection `/foo/manifest/v7`, so a client would then begin downloading the collection `/foo/manifest/v7/s0`, `/foo/manifest/v7/s1`, and so on. The single content object could enumerate the cryptographic hash codes of each of the collection members so a client could download them by exact, self-certified names.

The collection naming object will change over time, as the collection contents change. Therefore, it would have a short cache lifetime in the network. This causes the initial Interest message for the collection name to travel to an authoritative source, to the extent the collection naming object expires from the network.

2. Network Messages

This section describes the two CCNx messages used to transfer user data. An Interest message is a request from a client for data, transferred in a Content Object

[1] There are some experimental, small-network applications that use foraging and and epidemic information flow models rather than authoritative routing. These experimental networks are outside the scope of this document.

message. One Interest message receives, at most, one Content Object message, so there is flow balance in messages. An Interest message could also be considered a type of flow control, as a client could issue a set of Interest messages for different Content Objects. The client opens up a window for responses. Unlike most data networking protocols, the window is measured in messages, not bytes.

A Content Object binds a user-assigned name to the user-data (payload) via a cryptographic signature. The publisher of a Content Object identifies itself via a field called the KeyId, which is usually the SHA-256 hash of the signer's public key. A Content Object is a balance between being sized for network transport and amortizing the overhead of signing and other control information overhead, such as embedding cryptographic keys and signatures.

2.1 Interest Message

An Interest message always carries a Name. That is the only mandatory field. The Name identifies a Content Object to return. Any content object with an equal name satisfies the interest. An Interest may also carry a KeyId restriction and a Content Object Hash restriction. These two fields limit the set of content objects with matching names. The KeyId specifies a publisher (by its cryptographic signing key) and the Content Object Hash limits the response to a Content Object with the given cryptographic hash.

An Interest also carries a few directives that affect how the network handles responses. The Scope attribute determines if the Interest should be sent only to local applications, only 1-hop neighbors, or is unlimited distribution. The Lifetime field gives the minimum time before a node will ask for the same name again. This allows the network to gauge how long it should keep the interest pending, but there is no contract that the network must keep it for the requested lifetime.

If a protocol desires that an Interest only be answered by an authoritative source, it should include a nonce in the name. The authoritative source must also understand the use of nonces in names. So long as the nonce is unique among requesters, there will be no cache hits for it and only an authority will reply. It is not sufficient to use a flag like "Answer Origin Kind" (from the 0.x protocol design), because there is no way to determine if a content object was answered from cache or from an authoritative source. For example, a Interest goes along the routing and asks for cached data. A second Interest comes along and asks only for non-cached data, so a forwarder sends it upstream too. When a Content Object response comes, there is no way to tell if the response was for the first, cached allowed, or the second, cache not allowed, interest.

An Interest message sometimes carries state. For example, a name component could encode data like a session identifier. In this example name, an application-specific type, known to the application as "SessionId", identifies the session.

(2) `lci:/Name=foo/Name=bar/App:SessionId=0x5512334`

A name component could also identify a computation task, such as retrieving the current account balance:

(3) `lci:/Name=foo/Name=bar/App:SessionId=0x5512334/App:Task=AccountBalance`

Carrying state in an Interest may be more complex than the simpleton attributes given above. For example, one could serialize a complex data structure into a name component. A JSON data record could be encoded as a string in a name segment, or a binary record could be included as-is in a binary path segment.

(4) `lci:/Name=foo/Name=bar/App:State={SessionId:"0x5512334",Task:"AccountBalance",...}`

Some experimental networks use a new payload field inside the interest to separate out state from the name. The Interest name caries a hash or nonce to indicate the embedded state. This is required so returning Content Objects correctly match the appropriate Interest with payload if there are multiple outstanding at a forwarder. The embedded state – as it is not part of the name – does not need to be kept at each hop to facilitate reverse path routing, so the state is smaller and the memory bandwidth is lower.

2.2 Content Object Message

The Content Object Hash is the SHA-256 hash of the content object's wire format, from the opening ContentObject tag to the end of the Content Object. It does not include the initial packet header or per-hop TLVs. The Content Object Hash is not an explicit field in the packet, but must be calculated. This calculation provides an assurance that, for correctly behaving nodes, within the network, if a user requests a content object by hash, the network delivers the correct packet.

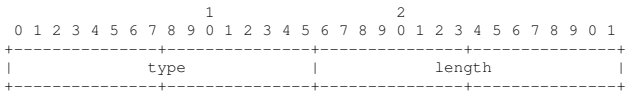

Figure 1. TLV Type and Length format

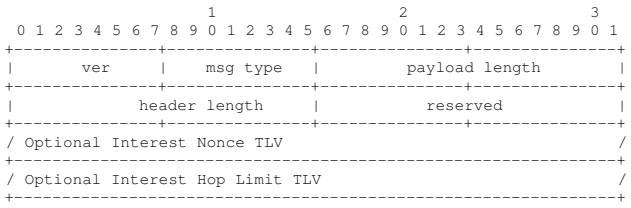

Figure 2. Fixed Header and Per-Hop fields

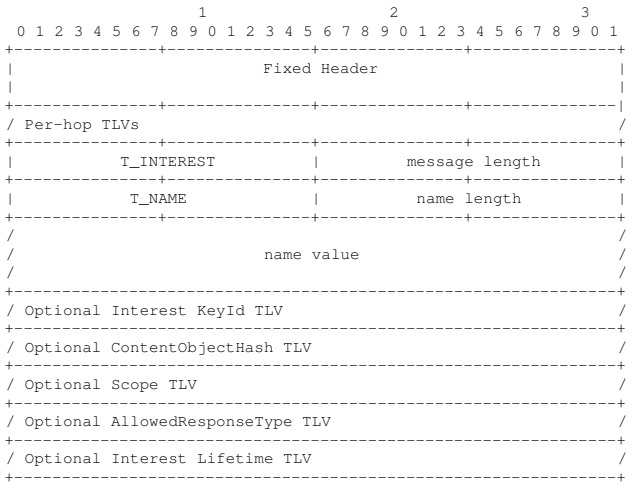

Figure 3. TLV Interest

2.3 Wire Format

CCNx 1.0 uses a Type Length Value format. The type and length fields are both exactly two bytes, as shown in Fig 1. Using a fixed type and length size avoid issues with aliases and is simple for high speed equipment to parse. In this section, we describe the TLV format, the fixed packet header, and then sketch out the skeletons of Interest and Content Object messages. We cover the important fields of the Interest and Content Object messages.

Aliasing problems can occur if the type or length fields can vary in size. For example, if a certain type, such as "1", must it always be specified in a 1-byte type? If the length field may be different sizes, then how does one encode values? For certain operations, such as hashes over a Content Object, the value of that hash will depend on the length of a value. If one Content Object encodes "0" as 2-bytes and another, equivalent, Content Object encodes it as 1-byte, then they will not evaluate to the same hash. Therefore, a TLV system would need specific rules on how to encode values, and then validate that encoding. Using a fixed 2-byte type and 2-byte length removes these alias problems and reduces the number of validation checks.

Each CCNx 1.0 packet begins with a fixed header followed by optional per-hop TLV fields. The per-hop TLV fields are not considered part of the CCNx message, but rather communicate in-network state, such as a hop limit or DSCP class. The `ver` field indicates the overall protocol version for the TLV encoding. The `msgtype` field indicates type type of message that follows the per-hop fields. `payloadlength` is the octets of the payload after the per-hop headers. `headerlength` is the bytes of header, including the fixed header and all per-hop headers.

Fig. 3 shows an Interest message in TLV format. The value `T_INTEREST` is a protocol constant to identify an Interest message. The value `T_NAME` is a protocol constant to identify a CCNx Name, which is a nested TLV structure. The remainder of the Interest message are optional nested TLV containers, such as the KeyId restriction or the Content Object Hash restriction. The Scope determines if the Interest is machine-local, link-local, or unlimited dissemination. The Allowed Response Type specifies the kinds of answers the requester accepts, such as if cached responses are allowed. The Interest Lifetime is an upper limit on how long the Interest should persist in the network. The requester should not re-express the Interest more frequently than the Lifetime. The network, however, may store unanswered Interests for less than the specific lifetime.

Fig. 4 shows a TLV encoded Content Object. It begins exactly the same as an Interest, with a fixed header, optional per-hop headers, the message container (`T_OBJECT`), and the CCNx Name. The remaining containers are specific to a content object. The Name Authenticator container encloses nested TLV structures needed to authenticate the Content Object. It specifies, for example, the cryptographic signature algorithm and the KeyId used to sign the Content Object. A publisher can also embed the public key, X.509 certificate, or an indirection key locator in this structure. The Protocol Information container encloses nested TLV structures for related protocols, such as a chunking scheme, a versioning scheme. For example, a chunking scheme could

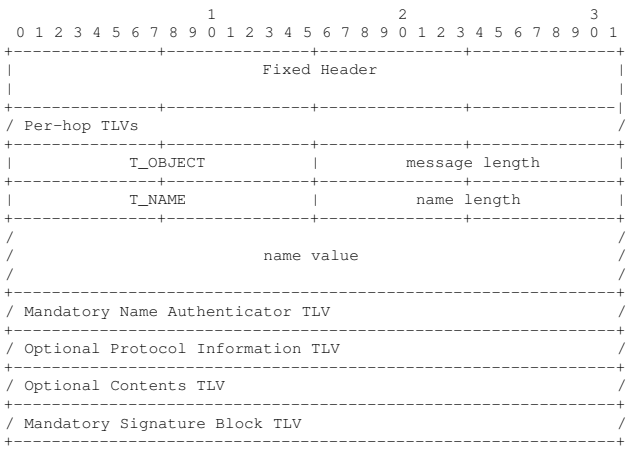

Figure 4. TLV Content Object

include a field here to indicate the last chunk number. The Contents TLV contains an opaque value that is the Content Object payload. The Signature Block holds the cryptographic signature for the content object.

2.4 Fragmentation

A CCNx message may be larger than a specific networking technology allows. The limitation, known as the Maximum Transmission Unit (MTU), typically varies from 1280 octets (for some IPv6 tunnels) to 1500 (Ethernet) to 9000 (Ethernet Jumbo frames), with 1500 being the most common [2]. CCNx messages, however, need to accommodate potentially large Names and have key and signing overhead. A Content Object could be large, 8KB to 64 KB, so it must be fragmented over specific network media.

Today's network layer (L3) protocols all use some form of fragmentation. IPv4 uses so-called "mid-to-end" fragmentation, where the first network device unable to fit a packet on a media begins fragmenting it. IPv6 uses end-to-end fragmentation.

There are currently two proposals for fragmenting large content objects to fit specific network media without using an intermediate format, such as IPv4 or IPv6. One approach is "hop-by-hop" and the other is "end-to-end". We will describe each of these approaches in turn with attention to their advantages and disadvantages.

2.5 Hop-by-hop fragmentation

The hop-by-hop fragmentation proposal uses a technique similar to PPP. On a peer-to-peer basis, nodes negotiate the maximum allowable MTU, or default to the media's MTU. Each CCNx message is split into an ordered set of packets that carry a sequence number, and flags to indicate the Beginning (B) packet and the End (E) packet in the set (for a 1 packet set, both B and E are set). One could also use an existing technology like multi-link PPP [1].

One significant advantage of hop-by-hop fragmentation is that the forwarder operates at the CCNx message level. It has complete access to the entire packet, so a long name would not be split over multiple messages above the hop-by-hop fragmenter.

One significant disadvantage of hop-by-hop fragmentation is that each hop must have buffer space for entire CCNx messages – up to 64 Kbyte – and has the added complexity of packet assembly. If the underlying media can multiplex packets, such as in Ethernet, then the reassembler must rely on timeouts or some other message constructs to detect that a reassembly buffer will not complete.

For a given amount of packet buffer, hop-by-hop reassembly has a much lower concurrent-session limit than end-to-end. If a switch has B bytes of packet buffer, each message is M bytes, then one may have at most $\lfloor B/M \rfloor$ active flows. For example, a switch with 2 GByte of packet buffer and 64 Kbyte messages can have at most 32768 active flows.

According to Appenzeller, in 2004, a core router has between 10,000 and 100,000 simultaneous flows [2]. If we take the upper limit, as things have only increased since 2004, then 100,000 flows at 64 Kbyte per flow means a core router needs 6.250 GByte of packet memory, in the worst case.

Fig. 5 illustrates the diminishing number of simultaneous flows on a forwarder, assuming fixed memory and interleaved flows. The figure uses the equation $f = m/(s+k)$, where f is the number of simultaneous flows, m is the fixed memory size (i.e. 4GB), and s is the Content Object segment size. The factor k is the PIT overhead of end-to-end fragmentation to keep state beyond hop-by-hop fragmentation between packets in a flow. It is 8 bytes in the figure.

2.6 End-to-end fragmentation

A different approach is to use end-to-end fragmentation, somewhat like IPv6. The principle of operation for fragmentation is that intermediate systems should not have to fragment packets. This is achieved by an Interest always fragmented to the minimum MTU and recording the for-

[2] Other common MTUs are 1380, 1400, and 1428. These are widely used in tunnels, such as 3GPP cellular networks and VPNs.

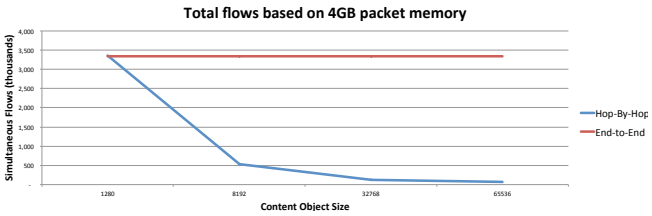

Figure 5. Simultaneous flows for fixed memory

ward path's MTU in the Interest, so a system sending back a Content Object may fragment it to the path's size, or smaller. An intermediate system's content store may store only pre-fragmented objects and respond only if those fragments satisfy an Interest's MTU, otherwise it may be considered a cache miss.

Because the Interest is the path discovery mechanism for content objects, all interests must carry an Interest Fragmentation Header so the reverse path MTU is known at the node responding with a Content Object. The minimum MTU required is 1280 octets. Any system implementing a physical layer with a smaller MTU must implement link fragmentation, for example using a PPP layer over the small MTU network.

Systems must create fragment streams in the most compressed packing. That is, all fragments except the last must be fragmented to the same size. This means that all interests are fragmented to 1280 bytes except the last fragment. A Content Object may be fragmented to any size no more the path MTU discovered, but all fragments except the last must be the same size. This ensures that any two fragment streams of the same content object with the same MTU have the same representation.

When an end system creates a fragment stream, it generates a 64-bit number for the Fragment Stream ID. This number identifies a contiguous stream of fragments. An end system uses the Fragment Stream ID for reassembly. An intermediate system uses the Fragment Stream ID of a Content Object to ensure that only one stream of Content Object fragments follow a reverse PIT entry. If the Maximum Path MTU of a Content Object fragment is larger than the supported MTU on an egress interface, the fragment stream should be dropped on that interface, even if some fragments fit within the MTU.

At an end system re-assembling fragments, it should timeout reassembly if all fragments are not received within a system-dependent timeout. If the re-assembly of an Interest times out before the PIT entry, the PIT entry on the local system should be removed to allow a new fragment stream to arrive. If the re-assembly of a Content Object times out, the received fragments bitmask of the PIT should be cleared to allow a new stream of Content Objects to arrive.

As an Interest goes through the FIBs, it records the minimum path MTU based on the egress interface's MTU. A Content Object sent in response must be fragmented to less than or equal to the minimum path MTU. A forwarder may choose to put 1280 in the Minimum Path MTU field even if it supports larger MTUs.

Interests follow the FIB and all fragments of an Interest (i.e. the same fragment id) should follow the same FIB choices. If at a later time a similar interest arrives with a smaller minimum path MTU, it should be forwarded even though it is similar, to ensure that a returned Content Object is fragmented to a size that satisfies the Interest's path.

A significant disadvantage of end-to-end fragmentation is that a forwarding node must examine the Interest name to determine its forwarding. This requires that the forwarding node re-assemble the front of the Interest to examine the name. In a typical case, this means that the node must receive fragment 0 to have enough prefix name components to compute the route. A system may discard out-of- order fragments after fragment 0 during this re-assembly, and once fragment 0 arrives and the system constructs a PIT entry with the routing, it should send a control message along the Fragment Stream ID's reverse path to cause the source to resend the interest stream, which can now be forwarded out of order. Or, it may buffer out-of- order fragments.

3. Message Forwarding

The core protocol specifies that an Interest name must exactly match a Content Object Name. In addition, if the Interest has a KeyId restriction, that must be numerically equal to the Content Object's publisher KeyId; it does not imply a cryptographic operation. If the interest has a Content Object Hash restriction, then the forwarder must verify that the SHA-256 hash of the content object (minus some per-hop headers and its signature) equals the hash value in the Interest. This is a strong restriction that involves a cryptographic hash.

A forwarder has three notional tables: Forwarding Information Base (FIB), Pending Interest Table (PIT), and Content Store (CS). An actual implementation may use a different information organization so long as the external behavior is the same. The FIB is the routing table. It is a longest-matching-prefix name table pop-

ulated by a routing protocol or with static routes. The PIT records Interest state such that Content Objects may follow the reverse Interest path. It also serves for Interest aggregation, so multiple similar interests do not get forwarded upstream. The Content Store is the in-network Content Object cache. The Content Store is optional.

Definition 3.1. (Satisfy Interest) *A Content Object satisfies an Interest if and only if (a) the Content Object name exactly matches the Interest name, and (b) the Content Object KeyId exactly equals the Interest KeyId restriction, if given, and (c) the Content Object Hash equals the Interest Content Object hash restriction, if given.*

As a Content Object moves through the "fast path" of the network, it matches pending interests at each hop according to the rules in Definition 3.1. Only end systems verify cryptographic signatures. Each hop may need to compute the Content Object Hash, if the pending interest includes the Content Object Hash restriction.

The Content Store has more restrictive matching rules. The reason for more restrictive rules is that the content store persists over time, so if incorrect content gets into the cache, it may cause persistent failures unless strong rules govern responses.

Definition 3.2. (Content Store) *If an Interest has a KeyId restriction, then the Content Store must verify a cached object's signature before returning it to the requester. This means that either the Interest must carry the desired public key or the Content Object itself must carry its own public key. If an interest has a Content Object Hash restriction, then the Content Store must verify a content object's hash prior to returning it to the requester.*

The CCNx 1.0 node behavior is as follows.

1. Receive Interest
   (a) If the interest may be satisfied from the Content Store, return that object to the previous hop, then discard the Interest.
   (b) Add to PIT (or aggregate)
   (c) Lookup the Interest name in the FIB, and forward it out all interfaces, excluding the ingress interface and those interfaces previously used if the PIT entry was already pending.
   (d) If the new Interest extends the lifetime of the PIT entry, then the forwarder should re-express the interest after the current lifetime expires and adjust the Interest's lifetime to the difference.
   (e) If a PIT entry's lifetime expires without being satisfied, the PIT entry is silently discarded.
   (f) If the Interest has a Scope "0" it is only forwarded to "local" applications.
   (g) If the Interest has a Scope "1" it is forwarded to "local" applications, and if it arrived from a "local" interface, it is forwarded out "remote" interfaces.
   (h) If the Interest has a hop count per-hop header, that header field is decremented on reception and it is only forwarded to "remote" interfaces if the difference is greater than zero.
2. Receive Content Object
   (a) A node looks in the PIT to find all PIT entries satisfied by the Interest (see Definition 3.1. If one or more of the PIT entries has a Content Object Hash restriction, the forwarder must verify the cryptographic hash matches the PIT entry, otherwise it does not satisfy those entries. If the forwarder finds one or more satisfied PIT entries, it forwards the Content Object out the interfaces denoted as ingress interfaces of the PIT entries, then removes those satisfied PIT entries.
   (b) A Content Object that does not match any PIT entries should be dropped.
   (c) A forwarder should verify that a Content Object arrives from an expected previous hop. If the previous hop is not in the FIB forwarding path for the name, it is likely the Content Object is an off-path injection attack, and it should be dropped.
   (d) If the node has a Content Store, it may save the Content Object in the Content Store. See Sec. **??** about Content Store cache directives.

3.1 Forwarding Fragmented Messages

If CCNx messages are fragmented, then the above forwarding rules need to include fragment matching mechanisms.

3.2 Routing and Strategy

There have been several proposals for CCN routing. This section describes one approach based on Core Based

Multicast and a second approach based on distance vector unicast/multicast protocol called Distance-based Content Routing (DCR).

One PARC proposal uses Core Based Multicast to setup acyclic graphs over which nodes distribute Interests. A single node authorized for a name prefix is elected the core of a name prefix and it becomes the root of a tree. Any publisher in the prefix sends a JOIN message to the core, which activates routers along the path to become engaged in the namespace forwarding. Interest messages are sent over the tree, so it is acyclic and interests do not loop. Any node along an interest path with content that matches the interest may return that content and stops forwarding the interest. This protocol is different than CBN. In CBN, a single acyclic graph is used as a broadcast distribution graph, over which subscriptions and routing advertisements flow. In our solution, the underlying graph is constructed based on the current participants of a namespace.

The Distance-based Content Routing (DCR) protocol enables routers to maintain multiple loop-free routes to the nearest sites advertising a named data object or name prefix, and establish content delivery trees over which all sites advertising the same named data object or name prefix can be contacted. Prior proposals for name-based content routing in information- centric networks (ICN) require flooding of content requests, or the exchange of information about the locations of content replicas and some routing information regarding the topology of the network. In contrast to all prior routing solutions for ICNs, DCR operates without requiring routers to establish overlays, know the network topology, use complete paths to content replicas, or know about all the sites storing replicas of named content. It is shown that DCR is correct and that is far more scalable than prior name-based routing approaches for ICNs, in terms of the speed with which correct routing tables are obtained and the amount of signaling incurred.

3.3 IP Overlay

CCNx packets can use IP encapsulation in point-to-point UDP or TCP. The forwarder considers each such point-to-point connection an interface. Each Interest and Content Object packet carries unicast IP and unicast MAC addresses.

CCNx can run over an IP multicast overlay. In this case, all Interest and Content Object packets carry the IP multicast destination address. At the MAC layer, the packets carry the corresponding L2 group address, as appropriate for the MAC. In a multicast overlay, an Interest packet will flood the overlay until some node generates a response, which will then flood the overlay corresponding to the Interest's path.

There may be some special cases where a UDP packet goes to a broadcast or multicast address, but the protocol expects a unicast response for the Content Object. These specialized uses are beyond the scope of a general description and depend on specific protocol mechanics to achieve the protocol designer's intentions.

3.4 LAN Ethernet Encapsulation

CCNx packets over direct Ethernet encapsulation use a CCNx frame type and CCNx fragmentation. The source MAC address is always the origin node's address. The destination address may be either a unicast MAC address or the CCNx Ethernet group address, derived from its IPv4 multicast address as per RFC 1112 [3].

A node sending an Interest uses the CCNx group Ethernet address. A node sending a Content Object response uses the unicast destination address for the source of the Interest. This is to avoid flooding all nodes on the network with unsolicited data packets on a switched Ethernet network. If an application specifically expects multicast behavior, it should use its own Ethernet group address.

4. Examples

4.1 CCNx Web Page

This section describes one method to serve web pages using CCNx 1.0. A publisher maintains a single object that describes a manifest for the web page. The entire manifest could be in the single object or it could continue over multiple subordinate objects. Based on the manifest, a client can download the entire contents of the web page.

Fig. 6 illustrates the process. A client requests the web page manifest, such as `/parc/manifest` for the PARC web site. The publisher responds with a Content Object containing the web site manifest. This object would have a relatively short cache lifetime, such as 30 seconds, which would allow some small amount of in-network caching, but short enough to allow liveliness in page updates. The manifest also carries the name of the versioned web page, which is the root of the persistent content objects with very long cache lifetimes. In this example, the current web page is identified as `/parc/page/v12`, being the 12th version. There follows a list

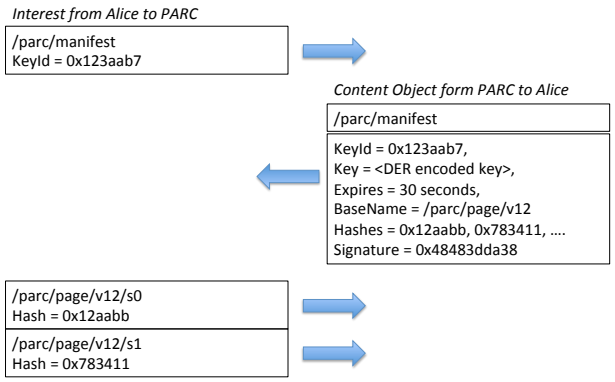

Figure 6. Web page manifest

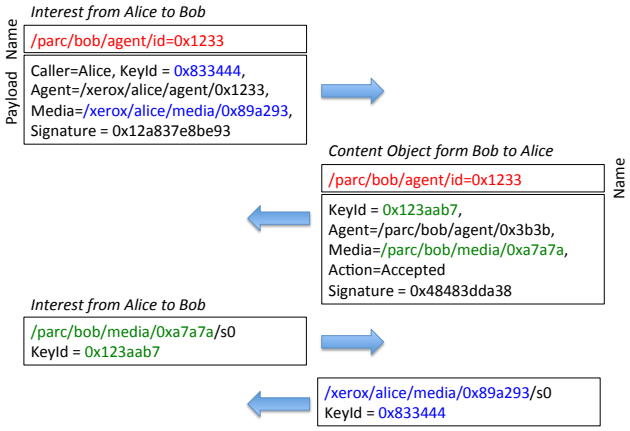

Figure 7. Phone call setup

of Content Object Hashes for each chunk of the web page. Using hash based naming allows for unequivocal download of the web pages. As shown in Fig. 6, after Alice gets back the manifest, she can begin download the actual web pages using the hash names `/parc/page/v12/s0` with hash `0x12aabb`, and so forth.

The security of this solution relies on Interests with KeyId and Interest with Content Object Hash restrictions. The initial request from Alice includes the KeyId of PARC's web site. This means that if a Content Store replies from cache, it should have verified the signature on the cached object to avoid persistently injecting bad content. Otherwise, the Interest would follow routing to an authoritative publisher who could issue and sign a proper manifest. If the signature on the manifest does not validate at Alice, she must retry her Interest, relying on routing to get the Interest to the right place. If PARC is the subject of a man-in-the-middle attack, then retrying will not work, but neither would any other technique. Once Alice accepts the manifest, then she can request the remainder of the content by hash, which is verified by the network even in the fast path.

Note that in Fig. 6, we show that the manifest Content Object includes the signer's public key inside it. This allows a Content Store to mechanically verify that the Key matches the KeyId and that it verifies the Signature. With this information, a Content Store can respond to an Interest with a KeyId restriction with assurance that it is not injecting bad content.

4.2 CCNx Phone Call

This section presents a notional example of how a point-to-point phone call would operate over CCNx. Fig. 7 illustrates the call setup procedure and Fig. 8 shows the call teardown. Each user has a well-known name for their call agent. Bob's agent is named `/parc/bob/agent`, so Alice uses that name plus a session ID. In her Interest she also indicates her agent's name. In this section, we have used short hexadecimal numbers for nonces, signatures, and key ids. A practical protocol would use much longer octet strings.

As shown in Fig. 7, when Alice wants to call Bob, she sends an Interest with payload to Bob. She creates a nonce ID for the call, in this example it is 0x1233. If Bob receives the Interest, he can send an Accept or Reject message back to Alice. In this case, Bob accepts the call, so he sends an Accept message back to Alice that also specifies his media stream. The contents of the Interest payload and Content Object payload are secured by cryptographic signatures so Bob knows whose calling and Alice knows it was Bob who replied.

If Bob does not reply, either because the Interest or Content Object are lost, then Alice can resend the Interest to Bob with the same nonce ID. Bob will repeat back with the same Content Object each time, until the transaction completes. It may be that the network loses the Content Object along the way, in which case the next Interest from Alice could retrieve a cached copy. That is acceptable, as long as Bob is still waiting to hear back on the given media stream name.

After Alice receives Bob's Accept respond, she can begin issuing Interests for Bob's media stream using the name he gave. In this example, the media stream for Bob is named `parc/bob/media/0xa7a7a`, so Alice begins asking for sequential Content Object segments of the stream, beginning with segment `s0`. Likewise, Bob begins requesting Alice's media stream using the name

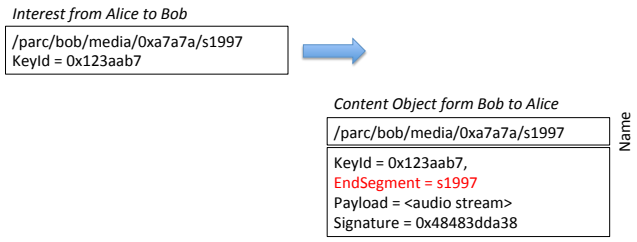

Figure 8. Phone call teardown

she gave in her Interest message.

Fig. 8 illustrates how the call ends. At some point, Bob is done with his media stream. He replies to the next Interest from Alice and indicates in the Content Object that the `EndSegment` is the current segment number, meaning that he will publish no more objects under the given name. He could include a last audio clip in the payload. The protocol could include a final end call confirmation on the call agent names.

4.3 CCNx for Video On Demand

An exemplary Video on Demand (VoD) system based on CCNx would operate as follows:

1. A user logs in to the service by issuing an Interest with their identity. The service responds with a challenge, and the user finishes the login by proving their identity to the challenge using their credentials.

2. The user gives the title of a movie, and the service returns the network name of a transport stream. For example, the user asks for the movie Looper in an Interest, which is of the form `/ccnflicks/session-0x1244/query/Looper`, where the name identifies the user's session and the movie name. The Interest would also carry the publisher's KeyId. The service responds with a Content Object that names the transport stream, such as `/ccnflicks/movies/looper_h264`. It could name several transport streams for the client application to pick the best encoding.

3. The user then requests a transport key, based on their identity and their device's identity. The Interest would be named for the movie transport, such as `/ccnflicks/movies/looper_h264/getkey`, and have encrypted payload (or additional name components) of the user's identity and device id. The service would respond with an encrypted Content Object with their personalized movie key.

4. The user then begins requesting the selected movie encodings manifest and content objects using Interests with Content Object Hash names. The Content Objects with the video stream use encryption that, when combined with the device id and the personalized movie key, allow playing.

5. Conclusion

The CCNx 1.0 protocols deliver an efficient, flexible Information Centric Networking implementation. This paper described the CCNx 1.0 protocols, from the wire format up through the message semantics. It also detailed protocol use cases and many examples.

References

[1] K. Sklower, B. Lloyd, G. McGregor, D. Carr, and T. Coradetti. The PPP Multilink Protocol (MP). RFC 1990 (Draft Standard), August 1996.

[2] Guido Appenzeller, Isaac Keslassy, and Nick McKeown. Sizing router buffers. In *Proceedings of the 2004 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '04, pages 281–292, New York, NY, USA, 2004. ACM.

[3] S.E. Deering. Host extensions for IP multicasting. RFC 1112 (INTERNET STANDARD), August 1989. Updated by RFC 2236.