ICN Research Group
Internet-Draft
Intended status: Experimental
Expires: May 19, 2017

H. Asaeda
X. Shao
NICT
T. Turletti
Inria

November 15, 2016

Contrace: Traceroute Facility for Content-Centric Network
draft-asaeda-icnrg-contrace-01

Abstract

This document describes the traceroute facility for Content-Centric Network (CCN), named "Contrace". Contrace investigates: 1) the forwarding path information per name prefix, device name, and function/application, 2) the Round-Trip Time (RTT) between content forwarder and consumer, and 3) the states of in-network cache per name prefix.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 19, 2017.

Table of Contents

1. Introduction

In Content-Centric Network (CCN) or Named-Data Network (NDN), publishers provide content through the network, and receivers retrieve content by name. In this network architecture, routers forward content requests by means of their Forwarding Information Bases (FIBs), which are populated by name-based routing protocols. CCN/NDN also enables receivers to retrieve content from an in-network cache.

In CCN/NDN, while consumers do not generally need to know which content forwarder is transmitting the content to them, operators and developers may want to identify the content forwarder and observe the forwarding path information per name prefix for troubleshooting or investigating the network conditions.

Traceroute [5] is a useful tool for analyzing the routing conditions in IP networks as it provides intermediate router addresses along the path between source and destination and the Round-Trip Time (RTT) for the path. However, this IP-based network tool cannot trace the name prefix paths used in CCN/NDN. Moreover, given a source-rooted forwarding path per name prefix, specifying a forwarding source (i.e., router or publisher) for any content is difficult, because we do not always know which branch of the source tree the consumer is on. Additionally, it is not feasible to flood the entire source-rooted tree to find the path from a source to a consumer. Furthermore, such IP-based network tool does not allow the states of the in-network cache to be discovered.

This document describes the specification of "Contrace", an active network measurement tool for investigating the path and caching condition in CCN. Contrace is designed based on the work originally published in [4].

Contrace consists of the Contrace user command and the Contrace forwarding function implementation on a content forwarder (e.g., router). The Contrace user (e.g., consumer) invokes the contrace command (described in Appendix A) with the name prefix of the content, the device name, or the function (or application) name. The Contrace command initiates the Contrace "Request" message (described in Section 3.1). The Request message, for example, obtains forwarding path and cache information. When an appropriate adjacent neighbor router receives the Request message, it retrieves cache information. If the router is not the content forwarder for the request, it inserts its "Report" block (described in Section 3.1.2) into the Request message and forwards the Request message to its upstream neighbor router(s) decided by its FIB. These two message types, Contrace Request and Reply messages, are encoded in the CCNx TLV format [1].

In this way, the Contrace Request message is forwarded by routers toward the content publisher, and the Contrace Report record is inserted by each intermediate router. When the Request message reaches the content forwarder (i.e., a router or the publisher who has the specified cache or content), the content forwarder forms the Contrace "Reply" message (described in Section 3.2) and sends it to the downstream neighbor router. The Reply message is forwarded back toward the Contrace user in a hop-by-hop manner. This request-reply message flow, walking up the tree from a consumer toward a publisher, is inspired by the design of the IP multicast traceroute facility [6].

Contrace supports multipath forwarding. The Request messages can be forwarded to multiple neighbor routers. When the Request messages forwarded to multiple routers, the different Reply messages will be forwarded from different routers or publisher. To support this case, PIT entries initiated by Contrace remain until the defined timeout value is expired.

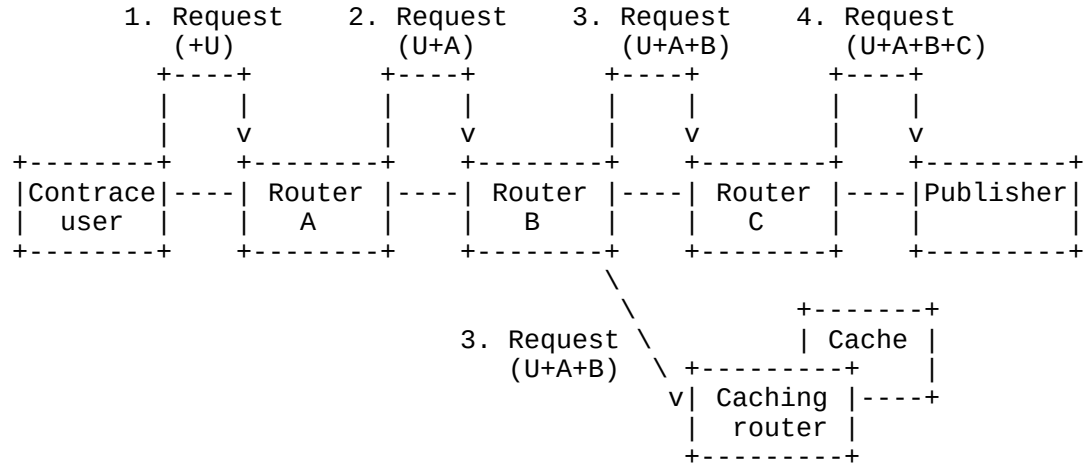


Figure 1: Request messages forwarded by consumer and routers. Contrace user and routers (i.e., Router A,B,C) insert their own Report blocks into the Request message and forward the message toward the content forwarder (i.e., caching router and publisher)

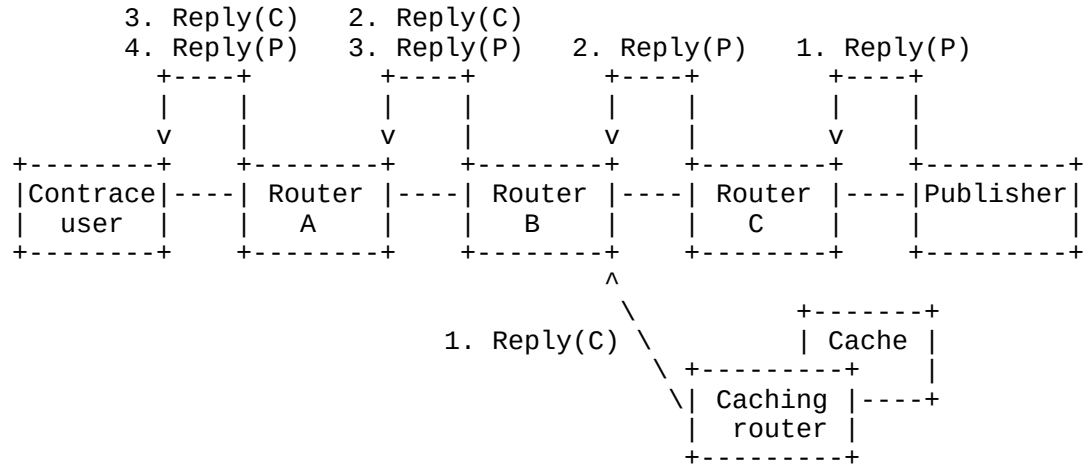


Figure 2: Reply messages forwarded by publisher and routers. Each router forwards the Reply message, and finally the Contrace user receives two Reply messages: one from the publisher and the other from the caching router.

Contrace facilitates the tracing of a routing path and provides: 1) the RTT between content forwarder (i.e., caching router or publisher) and consumer, 2) the states of in-network cache per name prefix, and 3) the forwarding path information per name prefix.

In addition, Contrace identifies the states of the cache, such as the following metrics for Content Store (CS) in the content forwarder: 1) size of the cached content, 2) number of the cached chunks of the content, 3) number of the accesses (i.e., received Interests) per cache or chunk, and 4) lifetime and expiration time per cache or chunk. The number of received Interests per cache or chunk on the routers indicates the popularity of the content.

Furthermore, Contrace implements policy-based information provisioning that enables administrators to "hide" secure or private information, but does not disrupt the forwarding of messages. This policy-based information provisioning reduces the deployment barrier faced by operators in installing and running Contrace on their routers.

2. Terminology

In this document, the key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" are to be interpreted as described in RFC 2119 [2], and indicate requirement levels for compliant Contrace implementations.

2.1. Definitions

Since Contrace requests flow in the opposite direction to the data flow, we refer to "upstream" and "downstream" with respect to data, unless explicitly specified.

Router
: It is a router facilitating name-based content/device/function name or characteristic retrieval in the path between consumer and publisher.

Scheme name
: It indicates a URI and protocol such as "ccnx:/" and "ndn:/". This document considers the protocol for name-based content/device/function name or characteristic retrieval.

Gateway
: It is a router supporting multiple scheme names in the path between consumer and publisher. The router may have multiple (different) FIBs.

Node
: It is a router, gateway, publisher, or consumer.

Content forwarder

It is either a router or a publisher that holds the cache (or content) and forwards it to consumers.

Contrace user
It is a node that invokes the contrace command and initiates the Contrace Request.

Incoming face
The face on which data is expected to arrive from the specified name prefix.

Outgoing face
The face to which data from the publisher or router is expected to transmit for the specified name prefix. It is also the face on which the Contrace Request messages are received.

3. Contrace Message Formats

Contrace uses two message types: Request and Reply. Both messages are encoded in the CCNx TLV format ([1], Figure 3). The Request message consists of a fixed header, Request block TLV Figure 7, and Report block TLV(s) Figure 11. The Reply message consists of a fixed header, Request block TLV, Report block TLV(s), and Reply block/sub-block TLV(s) Figure 14.

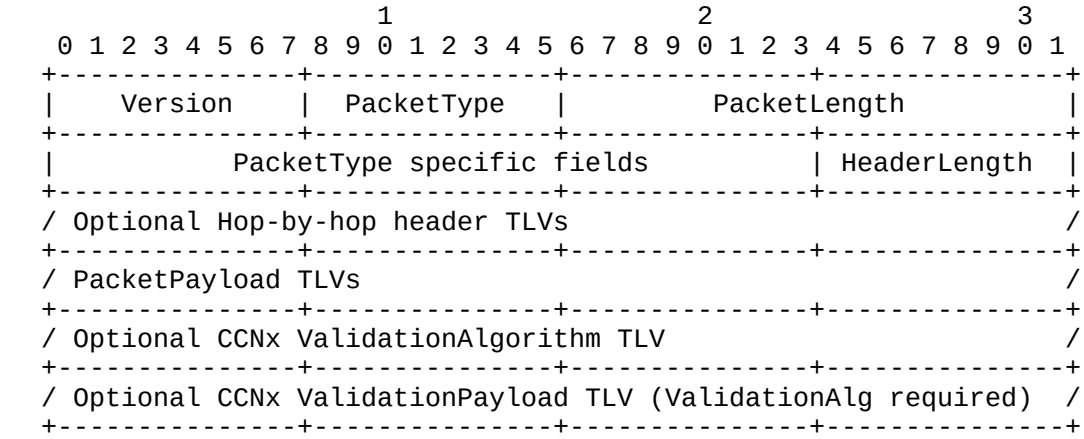


Figure 3: Packet format [1]

The Request and Reply Type values in the fixed header are PT_TRACE_REQ and PT_TRACE_REPLY, respectively (Figure 4). These messages are forwarded in a hop-by-hop manner. When the Request message reaches the content forwarder, the content forwarder turns the Request message into a Reply message by changing the Type field

value in the fixed header from PT_TRACE_REQ to PT_TRACE_REPLY and forwards back to the node that has initiated the Request message.

Code	Type name
1	PT_INTEREST [1]
2	PT_CONTENT [1]
3	PT_RETURN [1]
4	PT_TRACE_REQ
5	PT_TRACE_REPLY

Figure 4: Packet Type Namespace

Each Contrace message MUST begin with a fixed header with either a Request or Reply type value to specify whether it is a Request message or Reply message. Following a fixed header, there can be a sequence of optional hop-by-hop header TLV(s) for a Request message. In the case of a Request message, it is followed by a sequence of Report blocks, each from a router on the path toward the publisher or caching router.

At the beginning of PacketPayload TLVs, one top-level TLV type, T_TRACE (Figure 5), exists at the outermost level of a CCNx protocol message. This TLV indicates that the Name segment TLV(s) and Reply block TLV(s) would follow in the Request or Reply message.

Code	Type name
1	T_INTEREST [1]
2	T_OBJECT [1]
3	T_VALIDATION_ALG [1]
4	T_VALIDATION_PAYLOAD [1]
5	T_TRACE

Figure 5: Top-Level Type Namespace

3.1. Request Message

When a Contrace user initiates a trace request (e.g., by contrace command described in Appendix A), a Contrace Request message is created and forwarded to its upstream router through the Incoming face(s) determined by its FIB.

The packet format of the Contrace Request message is as shown in Figure 6. It consists of a fixed header, Request block TLV (Figure 7), Report block TLV(s) (Figure 11), and Name TLV. The Type value of Top-Level type namespace is T_TRACE (Figure 5). The Type value for the Report message is PT_TRACE_REQ.

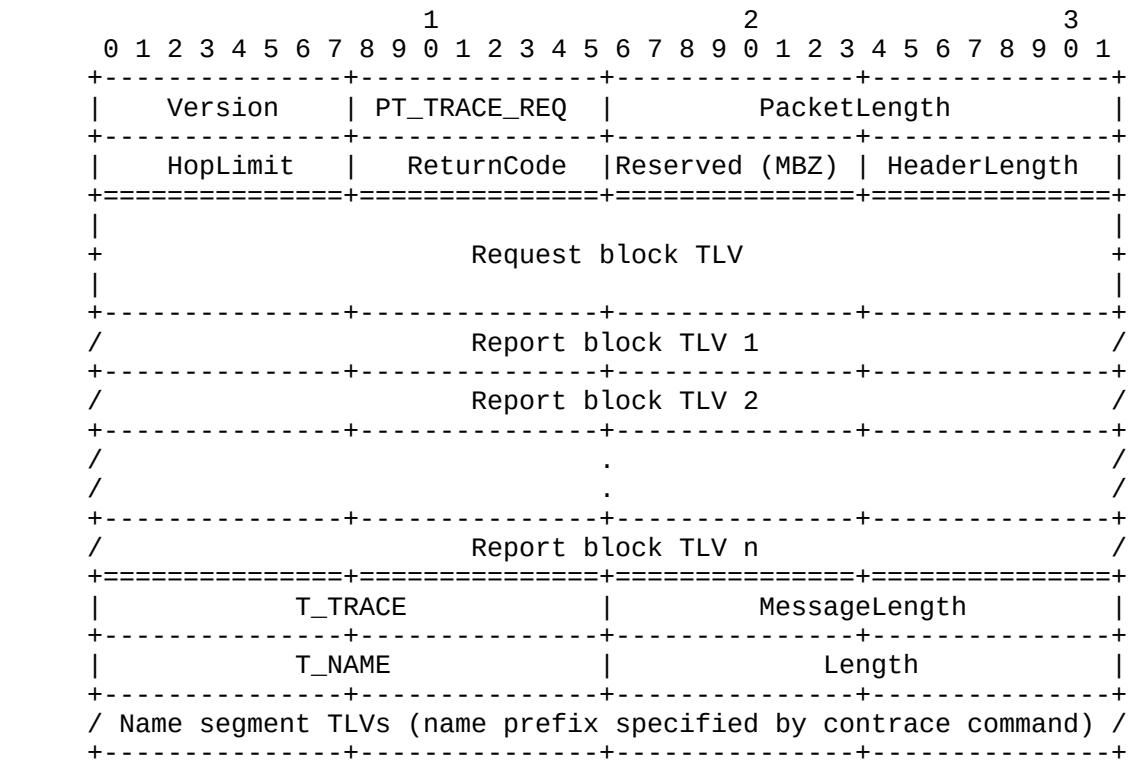


Figure 6: Packet format of the Request message

HopLimit: 8 bits

HopLimit is a counter that is decremented with each hop. It limits the distance a Request may travel on the network.

ReturnCode: 8 bits

ReturnCode is used for the Reply message. This value is replaced by the content forwarder when the Request message is returned as the Reply message (see Section 3.2). Until then, this field MUST be transmitted as zeros and ignored on receipt.

Value	Name	Description
0x00	NO_ERROR	No error
0x01	WRONG_IF	Contrace Request arrived on an interface to which this router would not forward for the specified name/function toward the publisher.
0x02	INVALID_REQUEST	Invalid Contrace Request is received.
0x03	NO_ROUTE	This router has no route for the named prefix and no way to determine a potential route.
0x04	NO_INFO	This router has no cache information for the specified name prefix, device information, or function.
0x05	NO_SPACE	There was not enough room to insert another Report block in the packet.
0x06	INFO_HIDDEN	Information is hidden from this trace because of some policy.
0x0E	ADMIN_PROHIB	Contrace Request is administratively prohibited.
0x0F	UNKNOWN_REQUEST	This router does not support/recognize the Request message.
0x80	FATAL_ERROR	A fatal error is one where the router may know the upstream router but cannot forward the message to it.

Reserved (MBZ): 8 bits

The reserved fields in the Value field MUST be transmitted as zeros and ignored on receipt.

3.1.1. Request Block

When a Contrace user transmits the Request message, it MUST insert the Request block TLV (Figure 7) and the Report block TLV (Figure 11) of its own to the Request message before sending it through the Incoming face(s).

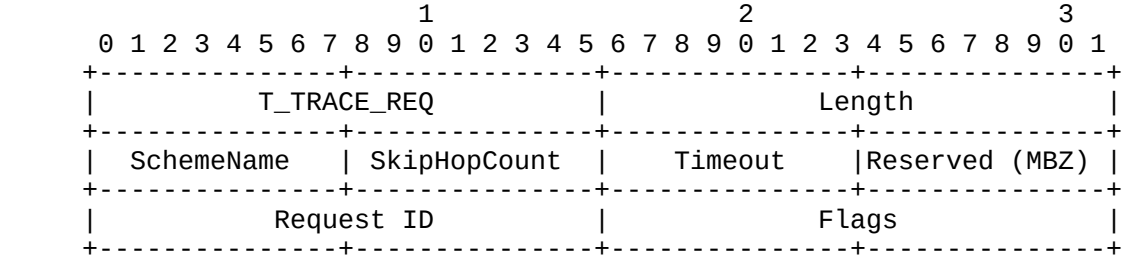


Figure 7: Request block TLV (hop-by-hop header)

Code	Type name
1	T_INTLIFE [1]
2	T_CACHETIME [1]
3	T_MSGHASH [1]
4 - 7	Reserved [1]
8	T_TRACE_REQ
9	T_TRACE_REPORT
%x0FFE	T_PAD [1]
%x0FFF	T_ORG [1]
%x1000-%x1FFF	Reserved [1]

Figure 8: Hop-by-Hop Type Namespace

Type: 16 bits

Format of the Value field. For the single Request block TLV, the type value MUST be T_TRACE_REQ. For all the available types for hop-by-hop type namespace, please see Figure 8.

Length: 16 bits

Length of Value field in octets. For the Request block, it MUST be 4 in the current specification.

SchemeName: 8 bits

Currently, the following scheme names are defined.

Code	Scheme name
0	ccnx:/
1	ndn:/
%x02-%FF	Not assigned

Figure 9: Scheme Names

SkipHopCount: 8 bits

Number of skipped routers. This value MUST be lower than the value of HopLimit at the fixed header.

Timeout: 8 bits

It is a [Contrace Reply Timeout] value later described in Section 8.1. A Contrace user requests routers along the path to keep the PIT entry for the Request until this timeout value expires. Note that, because of some security concern

(Section 10.5), a router along the path may configure the shorter timeout value than this requested timeout value. In that case, the Request may be timed out and the Contrace user may not receive the Reply as expected.

Request ID: 16 bits

This field is used as a unique identifier for this Contrace Request so that duplicate or delayed Reply messages can be detected.

Flags: 16 bits

The trace conditions specified as the contrace command options (described in Appendix A) are transferred in the Flags field. The trace conditions depend on the specified name (i.e., name_prefix, device_name, or function_name) as shown in Figure 10. Note that code %x01 and %x02 are exclusive options; that is, only one of them should be turned on at once.

Code	Type name
%x01	Cache retrieval allowing partial match (name_prefix)
%x02	No cache information required (name_prefix)
%x04	Publisher reachability (name_prefix and device_name)
%x08	Discovery of gateway supporting specified scheme name (name_prefix, device_name, and function_name)
%x16	Function’s or application’s version number retrieval (function_name)
%x32-%x32768	Not assigned

Figure 10: Codes and types specified in Flags field

3.1.2. Report Block

A Contrace user and each upstream router along the path would insert its own Report block TLV without changing the Type field of the fixed header of the Request message until one of these routers is ready to send a Reply. In the Report block TLV (Figure 11), the Request Arrival Time and the Node Identifier MUST be inserted.

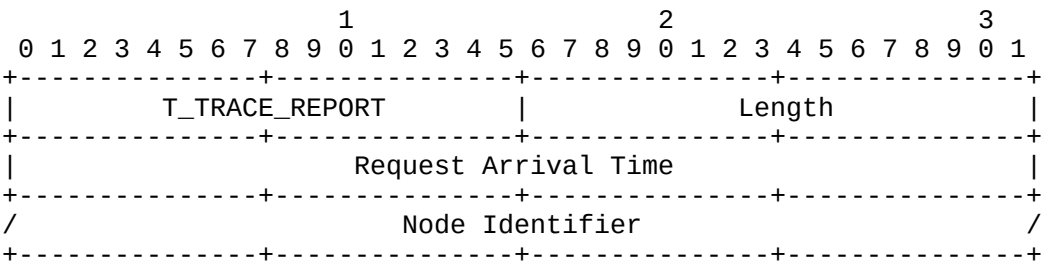


Figure 11: Report block TLV (hop-by-hop header)

Type: 16 bits

Format of the Value field. For the Request block TLV(s), the type value(s) MUST be T_TRACE_REPORT.

Length: 16 bits

Length of Value field in octets.

Request Arrival Time: 32 bits

The Request Arrival Time is a 32-bit NTP timestamp specifying the arrival time of the Contrace Request packet at this router. The 32-bit form of an NTP timestamp consists of the middle 32 bits of the full 64-bit form; that is, the low 16 bits of the integer part and the high 16 bits of the fractional part.

The following formula converts from a UNIX timeval to a 32-bit NTP timestamp:

```
request_arrival_time
= ((tv.tv_sec + 32384) << 16) + ((tv.tv_nsec << 7) / 1953125)
```

The constant 32384 is the number of seconds from Jan 1, 1900 to Jan 1, 1970 truncated to 16 bits. ((tv.tv_nsec << 7) / 1953125) is a reduction of ((tv.tv_nsec / 1000000000) << 16).

Note that all the routers on the path may not have synchronized clocks. In that case, the Contrace user measures the RTT between Contrace user and content forwarder. See Section 9.4.

Node Identifier: variable length

This field specifies the Contrace user or the router identifier (e.g., IPv4 address) of the Incoming face on which packets from the publisher are expected to arrive, or 0 if unknown or

unnumbered. Note that although it would be necessary to define the identifier uniqueness (e.g., by specifying the protocol family) for this field, defining such uniqueness is [TBD] as we may not always rely on the IP addressing architecture. Potentially, this field may be defined as a new TLV. Such discussion is also [TBD].

3.2. Reply Message

When a content forwarder receives a Contrace Request message from the appropriate adjacent neighbor router, it would insert a Reply block TLV and Reply sub-block TLV(s) of its own to the Request message and turn the Request into the Reply by changing the Type field of the fixed header of the Request message from PT_TRACE_REQ to PT_TRACE_REPLY. The Reply message (see Figure 12) would then be forwarded back toward the Contrace user in a hop-by-hop manner.

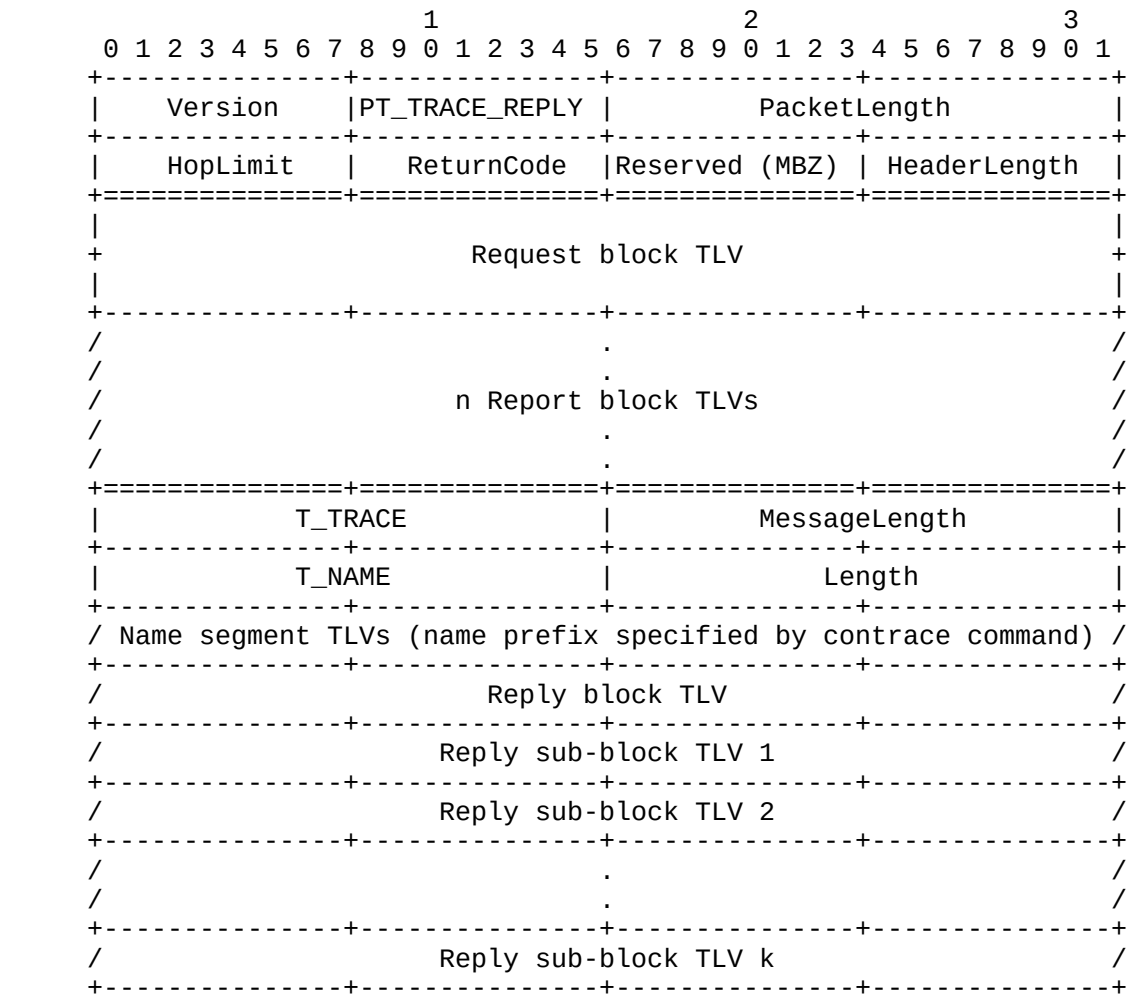


Figure 12: Reply message consists of a fixed header, Request block TLV, Report block TLV(s), Name TLV, and Reply block/sub-block TLV(s)

Code	Type name
0	T_NAME [1]
1	T_PAYLOAD [1]
2	T_KEYIDRESTR [1]
3	T_OBJHASHRESTR [1]
5	T_PAYLDTYPE [1]
6	T_EXPIRY [1]
8	T_TRACE_REPLY
9 - 12	Reserved [1]
%x0FFE	T_PAD [1]
%x0FFF	T_ORG [1]
%x1000-%x1FFF	Reserved [1]

Figure 13: CCNx Message Type Namespace

3.2.1. Reply Block

The Reply block TLV is an envelope for Reply sub-block TLV(s) (explained in Section 3.2.1.1).

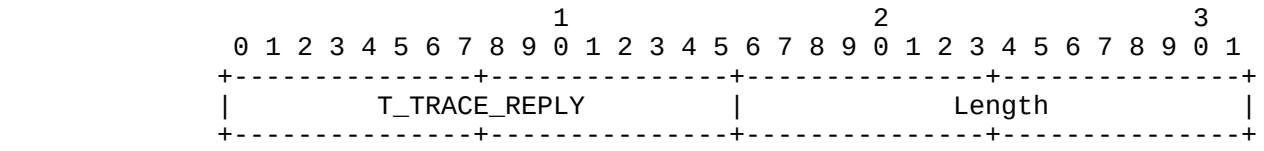


Figure 14: Reply block TLV (packet payload)

Type: 16 bits

Format of the Value field. For the Report block TLV, the type value MUST be T_TRACE_REPLY.

Length: 16 bits

Length of Value field in octets. This length is a total length of Reply sub-block(s).

3.2.1.1. Reply Sub-Block

In addition to the Reply block, a router on the traced path will add one or multiple Reply sub-blocks followed by the Reply block before sending the Reply to its neighbor router.

The Reply sub-block is flexible for various purposes. For instance, operators and developers may want to obtain various characteristics of content such as content's ownership and copyright, or other cache

states and conditions. Various information about device or function (or application) may be also retrieved by the variety of Reply sub-blocks. In this document, Reply sub-block TLVs for T_TRACE_CONTENT and T_TRACE_CONTENT_OWNER (Figure 15) and for T_TRACE_GATEWAY (Figure 16) are defined; other Reply sub-block TLVs are [TBD].

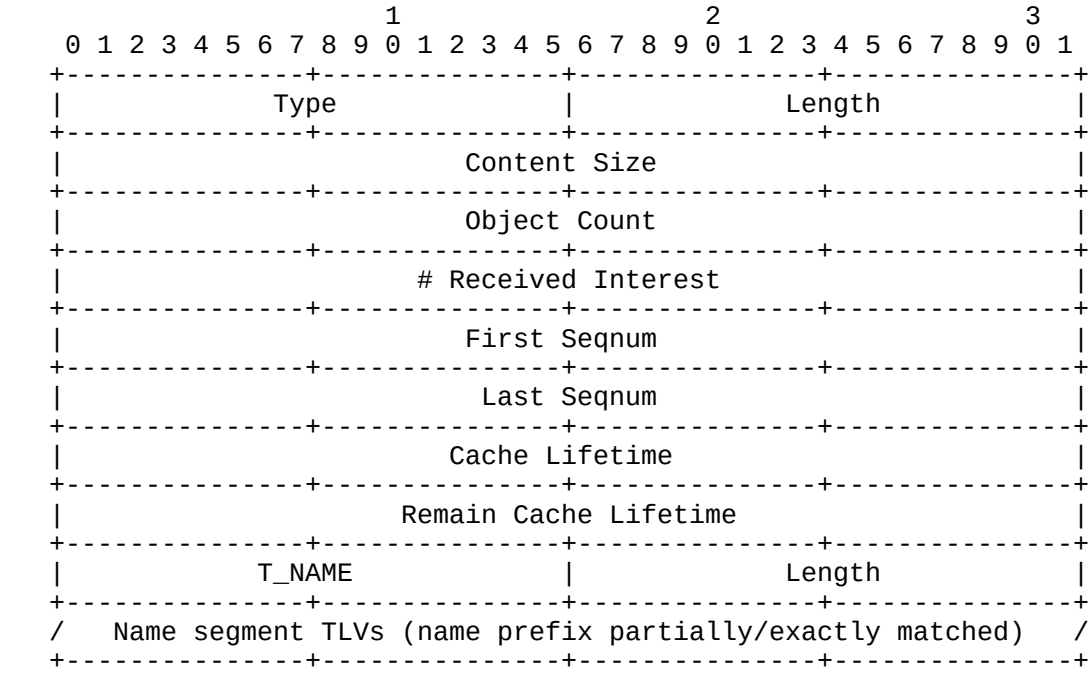


Figure 15: Reply sub-block TLV for T_TRACE_CONTENT and T_TRACE_CONTENT_OWNER (packet payload)

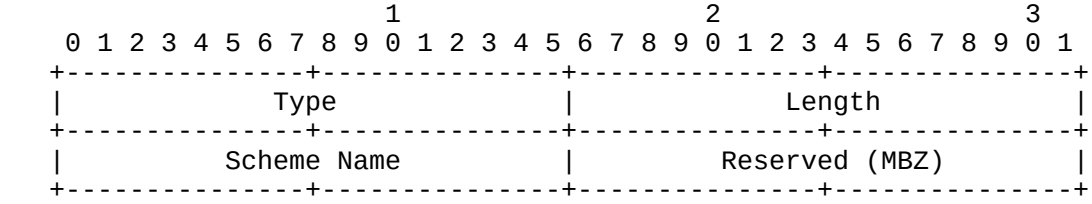


Figure 16: Reply sub-block TLV for T_TRACE_GATEWAY (packet payload)

Code	Type name
0	T_TRACE_CONTENT
1	T_TRACE_CONTENT_OWNER
2	T_TRACE_GATEWAY
3	T_TRACE_DEVICE
4	T_TRACE_FUNCTION
%x0FFF	T_ORG
%x1000-%x1FFF	Reserved (Experimental Use)

Figure 17: Contrace Reply Type Namespace

Type: 16 bits

Format of the Value field. For the Reply sub-block TLV, the type value MUST be one of the type value defined in the Contrace Reply Type Namespace (Figure 17). T_TRACE_CONTENT is specified when the cache information is replied from a caching router. T_TRACE_CONTENT_OWNER is specified when the content information is replied from a publisher. T_TRACE_GATEWAY is used to discover a gateway that has a FIB for the specified scheme name.

Length: 16 bits

Length of Value field in octets.

Scheme Name: 8 bits

The code of the scheme name defined in Figure 9.

Content Size: 32 bits

The total size (MB) of the (cached) content objects. Note that the maximum size expressed by 32 bit field is 65 GB.

Object Count: 32 bits

The number of the (cached) content objects.

Received Interest: 32 bits

The number of the received Interest messages to retrieve the content.

First Seqnum: 32 bits

The first sequential number of the (cached) content objects.

Last Seqnum: 32 bits

The last sequential number of the (cached) content objects. Above First Seqnum and this Last Seqnum do not guarantee the consecutiveness of the cached content objects.

Cache Lifetime: 32 bits

The elapsed time after the oldest content object in the cache is stored. The Cache Lifetime is a 32-bit NTP timestamp, and the formula converts from a UNIX timeval to a 32-bit NTP timestamp is same as that of Section 3.1.2.

Remain Cache Lifetime: 32 bits

The lifetime of a content object, which is removed first among the cached content objects. The Remain Cache Lifetime is a 32-bit NTP timestamp.

4. Contrace User Behavior

4.1. Sending Contrace Request

A Contrace user initiates a Contrace Request by sending the Request message to the adjacent neighbor router(s) of interest. As a typical example, a Contrace user invokes the contrace command (detailed in Appendix A) that forms a Request message and sends it to the user's adjacent neighbor router(s).

When the Contrace user's program initiates a Request message, it MUST insert the necessary values, the "Request ID" (in the Request block) and the "Node Identifier" (in the Report block), in the Request and Report blocks. Contrace user's program MUST also record the Request ID at the corresponding PIT entry. The Request ID is a unique identifier for the Contrace Request.

A Contrace Request with a scheme name (e.g., ccnx:/, ndn:/) is used to discover a router that has the FIB of the specified scheme name. In other words, a Contrace user can discover gateway(s) that support different protocols such as CCN and NDN. (Note that defining the way to interoperate different protocols is [TBD].) A Contrace Request with a scheme name may be forwarded to upstream routers being far from the Contrace user (see Section 5.2); therefore, it is RECOMMENDED that the number of traced routers is limited for the Request (e.g., contrace command with "-r" option with small hops, and repeated by incrementing the hop count if needed). It does not provide other information, e.g., cache information.

After the Contrace user's program sends the Request message, until the Reply times out, the Contrace user's program MUST keep the following information; Request ID and Flags specified in the Request block, Node Identifier and Request Arrival Time specified in the Report block, and HopLimit specified in the fixed header.

4.2. Receiving Contrace Reply

A Contrace user's program will receive one or multiple Contrace Reply messages from the adjacent neighbor router that has previously received and forwarded the Request message(s). When the program receives the Reply, it MUST compare the kept Request ID and the Request ID noted in the Reply. If they do not match, the Reply message SHOULD be silently discarded.

If the number of the Report blocks in the received Reply is more than the initial HopLimit value (which was inserted in the original Request) + 1, the Reply SHOULD be silently ignored.

After the Contrace user has determined that s/he has traced the whole path or as much as s/he can expect to, s/he might collect statistics by waiting a timeout. Useful statistics provided by Contrace can be seen in Section 9.

5. Router Behavior

5.1. Receiving Contrace Request

Upon receiving a Contrace Request message, a router MUST examine whether the message comes from a valid adjacent neighbor node. If it is invalid, the Request SHOULD be silently ignored.

When a router receives a Contrace Request message with a scheme name (e.g., ndn:/), the router checks whether the Request sets both "%x02" and "%x08" bits in the Flag or not. If not, the router returns a Reply with the INVALID_REQUEST return code. The router next checks whether it has the FIB for the specified scheme name. It the router has the corresponding FIB, it sends the Reply message back to the Contrace user. For a Request with a scheme name, the router SHOULD only inform the supported scheme name. See Section 5.3. If the router does not have the FIB for the specified scheme name but has the default neighbor router for another scheme name (e.g., FIB entry for ccnx:/), it forwards the Request to the default neighbor router. See Section 5.2. If the router does not have the FIB for the specified scheme name and does not have the default neighbor router for another scheme name, the router returns the Reply with the NO_ROUTE return code.

When a router receives a Request message requesting cache information, the router retrieves the cache information from its CS. If the router is the caching router that caches the requested content, it sends the Reply message. See Section 5.3. Otherwise, the router forwards the Request message to its upstream router(s). See Section 5.2.

If a router cannot continue the Request, it MUST put an appropriate ReturnCode in the Request message, change the Type field value in the fixed header from PT_TRACE_REQ to PT_TRACE_REPLY, and forward the Reply message back toward the Contrace user. See Section 7.

5.2. Forwarding Contrace Request

When a router decides to forward a Request message to its upstream router(s), it MUST insert a Report block having the Request Arrival Time and Node Identifier to the hop-by-hop TLV header of the Request message. The router then forwards the Request message upstream toward the publisher or caching router based on the FIB entry.

When the router forwards the Request message, it MUST record the Request ID at the corresponding PIT entry. The router can later decide the PIT entry to correctly forward back the Reply message even if it receives multiple Reply messages within the same timeout period. (See below.)

Contrace supports multipath forwarding. The Request messages can be forwarded to multiple neighbor routers. When the Request messages forwarded to multiple routers, the different Reply messages will be forwarded from different routers or publisher. To support this case, PIT entries initiated by Contrace remain until the configured Contrace Reply Timeout (Section 8.1) passes. In other words, unlike the ordinary Interest-Data communications in CCN, the router SHOULD NOT remove the PIT entry created by the Contrace Request before the timeout value expires, even if the router receives the Contrace Reply.

Routers can configure the Contrace Reply Timeout Section 8.1, which is the allowable timeout value to keep the PIT entry. In order to avoid DoS attacks (Section 10.5), routers MAY configure the shorter timeout value than the user-configured Contrace timeout value. If it is shorter, the Request may be timed out and the Contrace user may not receive the Reply as expected.

Contrace Requests SHOULD NOT result in PIT aggregation in routers during the Request message transmission.

5.3. Sending Contrace Reply

When a router decides to send a Reply message to its downstream neighbor router or the Contrace user, it MUST insert a Report block having the Request Arrival Time and Node Identifier to the hop-by-hop TLV header of the Request message. And then the router MUST insert the corresponding Reply block and Reply sub-block(s) to the payload if there is no error. The router does not insert any Reply block/ sub-block if there is an error. The router finally changes the Type field in the fixed header from PT_TRACE_REQ to PT_TRACE_REPLY and forwards the message back as the Reply toward the Contrace user in a hop-by-hop manner.

When a router decides to send the Reply message for the Request for the specified scheme name as seen in Section 5.1, it forms the Reply message including a Reply block and a Reply sub-block with the T_TRACE_GATEWAY type value (Figure 16) and the scheme name (Figure 9). After the router puts the NO_ERROR return code in the fixed header, it sends the Reply back toward the Contrace user.

When a router decides to send the Reply message for the Request for the cache information, it forms the Reply message including a Reply block and a Reply sub-block with the T_TRACE_CONTENT type value (Figure 15) and various cache information. After the router puts the NO_ERROR return code in the fixed header, it sends the Reply back toward the Contrace user.

5.4. Forwarding Contrace Reply

When the router receives a Contrace Reply whose Request ID matches the one in the original Contrace Request block TLV from a valid adjacent neighbor node, it MUST relay the Contrace Reply back to the Contrace user. If the router does not receive the corresponding Reply within the [Contrace Reply Timeout] period, then it removes the corresponding PIT entry and terminates the trace.

Contrace Replies MUST NOT be cached in routers upon the Reply message transmission.

6. Publisher Behavior

Upon receiving a Contrace Request message, a publisher MUST examine whether the message comes from a valid adjacent neighbor node. If it is invalid, the Request SHOULD be silently ignored.

If a publisher cannot accept the Request, it MUST note an appropriate ReturnCode in the Request message, change the Type field value in the fixed header from PT_TRACE_REQ to PT_TRACE_REPLY, and forward the

message as the Reply back to the Contrace user. See Section 7 for details.

If a publisher accepts the Request forwarded by a valid adjacent neighbor node, it retrieves the local content information. The Reply message having a Reply block and Reply sub-block(s) is transmitted back to the neighbor node that had forwarded the Request message.

7. Contrace Termination

When performing an expanding hop-by-hop trace, it is necessary to determine when to stop expanding. There are several cases an intermediate router might return a Reply before a Request reaches the caching router or the publisher.

7.1. Arriving at Publisher or Gateway

A Contrace Request can be determined to have arrived at the publisher or gateway.

7.2. Arriving at Router Having Cache

A Contrace Request can be determined to have arrived at the router having the specified content cache within the specified HopLimit.

7.3. No Route

If the router cannot determine the forwarding paths or neighbor routers for the specified named prefix, device name, or function, the router MUST note a ReturnCode of NO_ROUTE in the fixed header of the message, and forwards the message as the Reply back to the Contrace user.

7.4. No Information

If the router does not have any information about the specified named prefix, device name, or function, the router MUST note a ReturnCode of NO_INFO in the fixed header of the message, and forwards the message as the Reply back to the Contrace user.

7.5. No Space

If appending the Report block would make the Contrace Request packet longer than the MTU of the Incoming face, or longer than 1280 bytes (especially in the situation supporting IPv6 as the payload [3]), the router MUST note a ReturnCode of NO_SPACE in the fixed header of the message, and forwards the message as the Reply back to the Contrace user.

7.6. Fatal Error

A Contrace Request has encountered a fatal error if the last ReturnCode in the trace has the 0x80 bit set (see Section 3.1).

7.7. Contrace Reply Timeout

If a Contrace user or a router encounters the Request or Reply message whose expires its own [Contrace Reply Timeout] value (Section 8.1), which is used to time out a Contrace Reply such as the case of Section 7.8.

7.8. Non-Supported Node

Cases will arise in which a router or a publisher along the path does not support Contrace. In such cases, a Contrace user and routers that forward the Contrace Request will time out the Contrace request.

7.9. Administratively Prohibited

If Contrace is administratively prohibited, a router or a publisher rejects the Request message and its downstream router will reply the Contrace Reply with the ReturnCode of ADMIN_PROHIB.

8. Configurations

8.1. Contrace Reply Timeout

The [Contrace Reply Timeout] value is used to time out a Contrace Reply. Both Contrace users and routers can configure their own Contrace Reply Timeout values. Contrace users, for example, can configure the timeout value by the contrace command. The default [Contrace Reply Timeout] value is 4 (seconds). Routers may want to configure the short timeout values because of some security concern, e.g., Section 10.5. However, the [Contrace Reply Timeout] value SHOULD NOT be larger than 6 (seconds) and SHOULD NOT be lower than 3 (seconds).

8.2. HopLimit in Fixed Header

If a Contrace user does not specify the HopLimit value in a fixed header for a Request message as the HopLimit, the HopLimit is set to 32. Note that a Contrace user specifies 0 as the HopLimit, it is an invalid Request and discarded.

8.3. Access Control

A router MAY configure the valid or invalid networks to enable an access control. The access control can be defined per named prefix, such as "who can retrieve which named prefix". See Section 10.2.

9. Diagnosis and Analysis

9.1. Number of Hops

A Contrace Request message is forwarded in a hop-by-hop manner and each forwarding router appended its own Report block. We can then verify the number of hops to reach the content forwarder or the publisher.

9.2. Caching Router and Gateway Identification

It is possible to identify the caching routers or a gateway in the path from the Contrace user to the content forwarder, while some routers may hide their identifier (with all-zeros) in the Report blocks (Section 10.1).

9.3. TTL or Hop Limit

By taking the HopLimit from the content forwarder and forwarding TTL threshold over all hops, it is possible to discover the TTL or hop limit required for the content forwarder to reach the Contrace user.

9.4. Time Delay

If the routers have synchronized clocks, it is possible to estimate the RTT between Contrace user and successive hops. If all the routers on the path do not have synchronized clocks, the Contrace user can estimate the RTT between Contrace user and content forwarder. Note that this delay includes control processing overhead, so is not necessarily indicative of the delay that data traffic would experience.

9.5. Path Stretch

By getting the path stretch "d / P", where "d" is the hop count of the data and "P" is the hop count from the consumer to the publisher, we can measure the improvement in path stretch in various cases, such as different caching and routing algorithms. We can then facilitate investigation of the performance of the protocol.

9.6. Cache Hit Probability

Contrace can show the number of received interests per cache or chunk on a router. By this, Contrace measures the content popularity (i.e., the number of accesses for each content/cache), and you can investigate the routing/caching strategy in networks.

10. Security Considerations

This section addresses some of the security considerations.

10.1. Policy-Based Information Provisioning for Request

Although Contrace gives excellent troubleshooting cues, some network administrators or operators may not want to disclose everything about their network to the public, or may wish to securely transmit private information to specific members of their networks. Contrace provides policy-based information provisioning allowing network administrators to specify their response policy for each router.

The access policy regarding "who is allowed to retrieve what kind of information" can be defined for each router. The permission, whether (1) All (all cache information is disclosed), (2) Partial (cache information with the particular name prefix can (or cannot) be disclosed), or (3) Deny (no cache information is disclosed), is defined at routers.

On the other hand, we entail that each router does not disrupt forwarding Contrace Request and Reply messages. When a Request message is received, the router SHOULD insert Report block. Here, according to the policy configuration, the Node Identifier field in the Report block MAY be null (i.e., all-zeros), but the Request Arrival Time field SHOULD NOT be null. At last, the router SHOULD forward the Request message to the upstream router toward the content forwarder if no fatal error occurs.

10.2. Filtering of Contrace Users Located in Invalid Networks

A router MAY support an access control mechanism to filter out Requests from invalid Contrace users. For it, invalid networks (or domains) could, for example, be configured via a list of allowed/ disallowed networks (as seen in Section 8.3). If a Request is received from the disallowed network (according to the Node Identifier in the Request block), the Request SHOULD NOT be processed and the Reply with the ReturnCode of INFO_HIDDEN may be used to note that. The router MAY, however, perform rate limited logging of such events.

10.3. Topology Discovery

Contrace can be used to discover actively-used topologies. If a network topology is a secret, Contrace Requests may be restricted at the border of the domain, using the ADMIN_PROHIB return code.

10.4. Characteristics of Content

Contrace can be used to discover what publishers are sending to what kinds of contents. If this information is a secret, Contrace Requests may be restricted at the border of the domain, using the ADMIN_PROHIB return code.

10.5. Shortening Contrace Reply Timeout

Routers can configure the Contrace Reply Timeout Section 8.1, which is the allowable timeout value to keep the PIT entry. In order to avoid DoS attacks, routers MAY configure the shorter timeout value than the user-configured Contrace timeout value. If it is shorter, the Request may be timed out and the Contrace user may not receive the Reply as expected.

10.6. Limiting Request Rates

A router may limit Contrace Requests by ignoring some of the consecutive messages. The router MAY randomly ignore the received messages to minimize the processing overhead, i.e., to keep fairness in processing requests, or prevent traffic amplification. No error is returned. The rate limit is left to the router’s implementation.

10.7. Limiting Reply Rates

Contrace supporting multipath forwarding may result in one Request returning multiple Reply messages. In order to prevent abuse, the routers in the traced path MAY need to rate-limit the Replies. No error is returned. The rate limit function is left to the router’s implementation.

10.8. Adjacency Verification

Contrace Request and Reply messages MUST be forwarded by adjacent neighbor nodes or routers. Forwarding Contrace messages given from non-adjacent neighbor nodes/routers MUST be prohibited. Such invalid messages SHOULD be silently discarded. Note that defining the secure way to verify the adjacency cannot rely on the way specified in CCNx message format or semantics, and hence an new TLV for adjacency verification using hop-by-hop TLV header will be [TBD].

11. References

11.1. Normative References

[1] Mosko, M., Solis, I., and C. Wood, "CCNx Messages in TLV Format", draft-irtf-icnrg-ccnxmessages-03 (work in progress), June 2016.

[2] Bradner, S., "Key words for use in RFCs to indicate requirement levels", RFC 2119, March 1997.

[3] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998.

11.2. Informative References

[4] Asaeda, H., Matsuzono, K., and T. Turletti, "Contrace: A Tool for Measuring and Tracing Content-Centric Networks", IEEE Communications Magazine, Vol.53, No.3, pp.182-188, March 2015.

[5] Malkin, G., "Traceroute Using an IP Option", RFC 1393, January 1993.

[6] Asaeda, H., Mayer, K., and W. Lee, "Mtrace Version 2: Traceroute Facility for IP Multicast", draft-ietf-mboned-mtrace-v2-16 (work in progress), October 2016.

[7] Gill, V., Heasley, J., Meyer, D., Savola, P., and C. Pignataro, "The Generalized TTL Security Mechanism (GTSM)", RFC 5082, October 2007.

Appendix A. Contrace Command and Options

The contrace command enables the Contrace user to investigate the forwarding path based on the name prefix of the content (e.g., ccnx:/news/today), device name, and function (or application) name. The name prefix, device name, and function name (or application name) are mandatory but exclusive options; that is, only one of them should be used with the contrace command at once.

The usage of contrace command is as follows:

Usage: contrace [-p] [-g] [-n] [-o] [-r hop_count] [-s hop_count] [-w wait_time] name_prefix; or,

Usage: contrace [-r hop_count] [-s hop_count] [-w wait_time] device_name | function_name (or application_name)

name_prefix
Name prefix of the content (e.g., ccnx:/news/today) the Contrace user wants to trace. If the Contrace user specifies only a scheme name, e.g., "ccnx:/", s/he must specify "-g" option (i.e., contrace -g ccnx:/). In that case, the Contrace user discovers the router having the FIB of the specified scheme name and the RTT between Contrace user and the router. The -p option allows a partial match for the name prefix; otherwise, an exact match is required.

device_name
Device name (e.g., ccnx:/%device/server-A, ccnx:/%device/sensor-123) the Contrace user wants to trace. Here, we assume the contrace command with the "%device" prefix indicates the trace request for specified device/server/node, but defining the syntax of device name specification is [TBD].

function_name (or application_name)
Function name (e.g., ccnx:/%function/firewall, ccnx:/%function/transcoding/mpeg2-h.264) or application name (e.g., ccnx:/%application/mplayer) the Contrace user wants to trace. Here, we assume the contrace command with the "%function" or "%application" prefix indicates the trace request for specified function or application, but defining the syntax of function or application name specification is [TBD].

g option
This option can be specified if a Contrace user wants to discover a gateway that supports specified scheme name and may have multiple FIBs. When a Contrace user specifies only a scheme name, e.g., "ccnx:/", this option must be specified and other content name prefix is ignored.

n option
This option can be specified if a Contrace user only needs the routing path information to the specified content/cache and RTT between Contrace user and content forwarder (i.e., cache information is not given).

o option
This option can be specified if a Contrace user needs to trace the path to the content publisher. If this option is specified, each router along the path to the publisher only forwards the Request message; it does not insert each Report block and does not send Reply even if it caches the specified content. The publisher (who has the complete set of content and is not a caching router) replies the Reply message. Specifying only a scheme name is not allowed with this option.

r option
: Number of traced routers. If the Contrace user specifies this option, only the specified number of hops from the Contrace user trace the Request; each router inserts its own Report block and forwards the Request message to the upstream router(s), and the last router stops the trace and sends the Reply message back to the Contrace user. This value is set in the "HopLimit" field located in the fixed header of the Request. For example, when the Contrace user invokes the Contrace command with this option such as "-r 3", the two upstream routers along the path append their Report blocks in the Request message, and the next (and last) router sends back the Reply message. If there is a caching router within the hop count along the path, the caching router sends back the Reply message and terminates the trace request. If the last router does not have the corresponding cache, it replies the Reply message with NO_INFO return code (described in Section 3.1) with no Reply block TLV inserted. The Request messages are terminated at publishers; therefore, although the maximum value for this option a Contrace user can specify is 255, the Request messages should be in general reached at the publisher within significantly lower than 255 hops.

s option
: Number of skipped routers. If the Contrace user specifies this option, the number of hops from the Contrace user simply forward the Contrace Request messages without adding its own Report block and without replying the Request, and the next upstream router starts the trace. This value is set in the "SkipHopCount" field located in the Request block TLV. For example, when the Contrace user invokes the Contrace command with this option such as "-s 3", the three upstream routers along the path only forwards the Request message, but does not append their Report blocks in the hop-by-hop headers and does not send the Reply messages even though they have the corresponding cache. The Request messages are terminated at publishers; therefore, although the maximum value for this option a Contrace user can specify is 255, if the Request messages reaches the publisher, the publisher silently discards the Request message and the request will be timed out.

w option
: This option defines the Contrace timeout value (in seconds) that the Contrace user will wait for the Reply. After the timeout, the Contrace user terminates the Request and silently discards the Reply message even if s/he receives the Reply. Note that routers along the path can configure the Contrace Reply Timeout Section 8.1, which is the allowable timeout value to keep the PIT entry. In order to avoid DoS attacks Section 10, routers MAY configure the shorter timeout value than the user-configured

Contrace timeout value. If it is shorter, the Request may be timed out and the Contrace user may not receive the Reply as expected.

Authors' Addresses

Hitoshi Asaeda
National Institute of Information and Communications Technology
4-2-1 Nukui-Kitamachi
Koganei, Tokyo 184-8795
Japan

Email: asaeda@nict.go.jp

Xun Shao
National Institute of Information and Communications Technology
4-2-1 Nukui-Kitamachi
Koganei, Tokyo 184-8795
Japan

Email: x-shao@nict.go.jp

Thierry Turletti
Inria
2004 Route des Lucioles
Sophia Antipolis 06902
France

Email: thierry.turletti@inria.fr

ICN Research Group
Internet-Draft
Intended status: Informational
Expires: May 4, 2017

A. Azgin
R. Ravindran
Huawei Technologies

October 31, 2016

Enabling Network Identifier (NI) in Information Centric Networks to Support Optimized Forwarding
draft-azgin-icnrg-ni-00

Abstract

The objective of this proposal is to introduce the notion of network identifier (NI) in the ICN architecture. This is in addition to the existing names (i.e., content identifiers, CIs, or application identifiers, AIs, in general) that are currently used for both naming and routing/forwarding purposes. Network identifiers are needed considering the requirements on future networking architectures such as: (i) to support persistent names (or persistently named objects) and large-scale and high-speed mobility of any network entity (i.e, devices, services, and content), (ii) to accommodate different types of Internet of Things (IoT) services, many of which require low-latency performance, and enabling edge computing to support service virtualization, which will require support for large scale migration and replication of named resources, and (iii) to scale the ICN architecture to future Internet scale considering the exponentially increasing named entities. These considerations also require enabling a network based name resolution service for efficient and scalable routing.

In the current draft, we begin by highlighting the issues associated with ICN networking when utilizing only the AIs, which include persistently named content, services, and devices. Next we discuss the function NI serves, and provide a discussion on the two current NI-based proposals, along with their scope and functionalities. This is with the objective of having a single NI construct for ICN that is flexible enough to adapt to different networking contexts.

Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

Information centric networking (ICN) is proposed as a future Internet architecture to evolve the current host-centric design of Internet towards a content-oriented one, where the named object becomes the principle entity in networking. In doing so, contents, services, and devices become disentangled from location allowing for efficient use of the distributed in-network caches and compute resources with more flexible and dynamic packet forwarding techniques. ICN is expected to offer a scalable and secure networking solution to address many challenges of the current IP architecture. Towards this, we propose to formalize the notion of network identifier (NI) in ICN protocol, that is separate from content name or application identifier (CI/AI, or simply AI) used to both name resources and route user requests.

2. Application Identifier (AI) vs. Network Identifier (NI) in ICN

AI represent the names of service, content, or devices assigned by the application providers or device manufacturers, and which can be validated through appropriate security mechanisms. ICN should provide flexibility in accommodating a broad set of identifiers, within which the two well-known classes include hierarchical and flat identifiers. While a hierarchical identifier provides contextual richness for the names, a flat identifier offers a fixed predictable overhead and variable security properties within a given context. Today, this identifier set is already in the order of billions (with hundreds of millions top-level domain names [VRSGN], and billions of second-level domain names). As tens of billions of devices are expected to join the network, this identifier set will be further augmented with the corresponding data objects significantly expanding its size. To decouple applications from the underlying network dynamics, identifiers are expected to be persistent within the scope of the application and its deployment.

NI provides a binding for the AI to the network, at a location and in a topology relevant manner. NI is managed by the network provider to name the routers, point of attachments, servers and end devices. In addition to ICN names, in an overlay deployment, NI could assume names of the underlay network as well, such as IP or Ethernet addresses. The growth of the NI space is proportional to the rate of growth of domain topology, the total number of AS, and the end points (if they are managed by the network), hence, being much slower than the rate of growth of the named resources in the AI space. Hence if the objective is to limit the size of the forwarding table and scale control plane, it is desirable to route requests on NIs, with the mapping between AI and NI is achieved in a scalable manner using a network based name resolution system.

Content-centric design used by ICN allows end hosts to make requests using any type of name supported by the applications, including hierarchical (human-readable or hash-based) identifiers (as considered by CCN, NDN[CCN] for both the client application use and the network use-for routing-), or fixed flat identifiers (as considered by MobilityFirst[MFRST] in the network for routing). We refer to an ICN architecture that supports any application naming format (i.e., human-readable or flat) within the network for routing as a non-restricted ICN architecture (as in CCN/NDN), whereas an ICN architecture with a fixed naming format for routing within the network as a restricted ICN architecture (as in MobilityFirst).

As packet forwarding in ICN utilizes names or identifiers (associated with contents, hosts, or services) which are typically managed by applications, thereby of persistent nature, using such names in packet forwarding introduces the following list challenges in regards to routing scalability and forwarding efficiency [NAMES].

- Using AI for Routing/Forwarding: Overloading an identifier as a locator can lead to unstable routing control and forwarding plane operations, particularly when replication and mobility of content or end points are taken into consideration. Applications typically construct names and replicate contents or services to optimize their delivery without any consideration towards network scalability or efficiency. Hence name aggregation does not help with scaling the routing and forwarding as originally imagined, and the cost of this would be quite significant in real world scenarios, as discussed in [NCMP]. Furthermore, it is also observed in [QCMP] that, in certain scenarios (such as content mobility), name-based forwarding approaches can operate more efficiently, if used in conjunction with address-assisted schemes such as DNS or anchor point based approaches like Mobile IP [RFC3220]. Additionally, when names are used for network reachability, more practical problems such as name-suffix hole may arise, as the content requests are forwarded towards non-existent caches [MDHT].

- Routing/Forwarding Scalability: Routing scalability is typically achieved by designing NIs with aggregate-able property, which is the case for the current IP architecture. However, having such feature in a non-restricted ICN architecture would lead to relinquishing the persistency of the names, along with its security binding such as trust, as the names would involve a topological component for scalability, which can also suggest resources to be renamed depending on, for instance, network or business specs or characteristics. When content names or application identifiers use a hierarchical identifier format, we observe scalability problems in control and data plane operations

[SFWD]. Such problems are caused by various factors. For instance, the explosive growth observed in namespaces can lead to a similar growth in routing/forwarding information base or table sizes [AFWD][SPIT][WPIT], even when namespace aggregation is enabled, to significantly limit the forwarding efficiency and forwarding capacity. If ICN routing with hierarchical naming is the accepted form of naming, name-aggregation is highly unlikely to achieve any practical scalability. This is because, naming ontology and assignment typically consider application objectives of contextualizing names, service and content placement and replication to better suit the consumers' needs without considering any network objectives on control and data plane efficiency and scalability.

- o Handling Mobility, Migration, and Replication: The impact of namespace expansion on routing/forwarding performance is typically exacerbated with content mobility, or the use of multi-homing and resource replication due to diminished aggregate-ability [NCMP]. The authors in [QCMP] concludes that, as more than 20% of end hosts make more than 10 network address transitions every day, thereby suggesting that mobility should be considered as the norm rather than the exception. Furthermore, to achieve location independent routing based on AIs, each mobility event associated with a device or a popular content may trigger updates on up to 14% of Internet routers.

For the above reasons, restructuring the identifier to directly or indirectly contain a globally routable component becomes an important requirement, especially, to handle mobility at the network layer for architectures that do not restrict names or identifiers to any specific format. We can refer to such operation as the Application and Network identifier split (where the NI represents the globally routable component, and the AI represents the persistent name/ identifier) which enables splitting of the namespace to support routable, persistent, and human-friendly names or identifiers. In such a framework, names would be divided accordingly, i.e., based on application binding (offering persistent names) vs. advertised network entities (in routing plane) to provide a more scalable routing architecture. For instance, a persistent name or identifier /Provider/Type/Name, which would be used to create secure content objects, can be published by multiple content distributors, where it would be mapped to different NIs, such as /Distributor/Region/Zone/ Storage, to resolve content names or identifiers to specific infrastructure entities. The fundamental requirements with this form of splitting is no different than that of MobilityFirst [MFRST] or LISP [RFC6830], which is the requirement of a network based name resolution system to map the two namespaces.

So far, various approaches have been proposed to support the use of NI in ICN-based networking architectures, depending on how this information is structured and where it is placed within the Interest (which may also determine the structuring of Data packets). Next, we discuss these solutions by specifically focusing on label-based ICN forwarding [FWLDR][FWLRP][MAAS] and ICN-based Map-and-Encap [MPNCP][SNAMP] to provide a general guidance on the use of NI in information centric networks.

3. NI based ICN Forwarding

AI based routing is a feasible solution within certain contexts such as: (i) when resources are static and routing is limited to local area networks or local domains, such as access networks within the scalability considerations of the control and forwarding plane; (ii) in ad hoc situations where AI can be combined with suitable suffix filters to seek content of interest for the applications.

On the other hand, the use of NI becomes important in the following situations: (i) when the Interest packet goes outside the local domain, where routing on AI is optionally supported (i.e., routing scalability and efficiency seeks precedence); (ii) when the Interest enters a local domain, and the domain has specific knowledge of an NI associated with the resource inside its domain.

With the above considerations, with respect to end-to-end networking, NI is not a mandatory feature, but an optional one. However, as significant amount of user traffic fetches resources outside the requesting host’s local domain, it becomes crucial to provide architectural support for NI in an ICN protocol. So far, two solutions for NI in ICN, overall with the same objectives but serving different purposes, have been proposed. These include the forwarding-label proposal [FWLDR] and the Link Object described in [SNAMP]. We next summarize these proposals and discuss their differences.

3.1. Label based ICN forwarding

Label-based ICN forwarding provides NI capability by encoding a network address along with (optional) security binding attributes within an Interest packet to guide it towards a content source (which can be the Producer, a content repository or a cache). We refer to this label as the forwarding label [FWLDR], which can be offered as part of an ICN network service (such as a name resolution service with ICN APIs to register and resolve names). For the forwarding label, we have the following important considerations: (i) forwarding label, if present in the Interest packet, takes precedence (over AI) for routing, (ii) forwarding label is mutable in the sense that it

can be swapped or removed by intermediate network elements in the network based on routing considerations within its domain. Here, forwarding labels are not limited to only the ICN names, but, in an overlay mode, they can also represent names from other transport layers as well, for instance, an IP address or a MAC address.

Forwarding label consists of multiple components, with the NI representing the locator information. Forwarding label is embedded within the Interest message at the edge router or the end point within certain trust considerations, if the namespace supports the use of an NI to reach a specific destination. For security reasons, edge routers can validate the label based on the trust context or ignore any label inserted by an ICN forwarder at the end hosts, by removing the inserted label if the forwarding on labels is not supported, or by swapping it with a new one depending on the feedback from the name resolution system. Such an approach requires no trust relationship among different domains, as each domain is capable of resolving content namespace to a target domain, and swapping the received label with one to which its resolves.

Forwarding label support for a namespace can be offered at a global scale (i.e., supported by all the domains) or a local scale (supported by a subset of the existing domains). For instance, some autonomous systems can prioritize forwarding solely based on the content names (or offer limited support for label-based forwarding on specific namespaces). In such case, forwarding labels can include additional service tag (or information on the associated service, for which the use of forwarding label might be supported in certain domains, such as towards mobility service) for routing packets on the supported domains. In doing so, we can strategically forward requests over domains that support such service to provide more deterministic service guarantees.

If forwarding label use is supported (or permitted) within a domain, by default, forwarding label is given preference over content identifiers for packet forwarding. In such case, to maximize the forwarding efficiency, additional mapping tables can be implemented at the edge or border ICN routers for quick longest-prefix matching (LPM) lookup on content names to determine a (or the) matching forwarding label(s), which can then be used by the router to perform LPM lookup on the FIB. As forwarding label typically represents a target domain or router, a single LPM lookup on the FIB may suffice to find the outgoing interface for the received Interest. This state can also be software-defined based on application requirements using an SDN based control plane.

3.2. Link-object based ICN forwarding

ICN-based Map-and-Encap utilizes link objects, which include information on how to retrieve content objects. For instance, link objects can represent domains that host the content object, or direction towards which the requests need to be forwarded to find a matching content object. Link objects consist of two optional headers: (i) a link header, which includes the potential directives that can be used for forwarding and is signed by the Producer to validate its authenticity during forwarding, and (ii) a delegation header, which is used to represent the link choice utilized by the previous forwarder. Since delegations may change at consecutive hops depending on the view of forwarders' network state and forwarding strategy, delegation header represents a variable component that can be altered during packet forwarding.

The role of link objects is mainly for guidance, to provide global routing support on locally defined or routable content identifiers. Hence, if link objects are implemented, they are consulted by the ICN enabled routers only when forwarding lookup on content identifiers returns no match on the forwarding information base.

3.3. Link Object vs. Forwarding Label

Next we list the major differences between a link object and a forwarding label.

- Link objects are set by the end host's forwarding daemon with certain level of trust associated to it, restricting the link component to be immutable during forwarding. Forwarding labels are set by the ICN edge routers or the end-host applications, with the ability of network based management during Interest forwarding, allowing each domain to perform packet forwarding according to its administrative and service policies.

- Forwarding label allows the use of trust association to bind AI to the NI depending on the context associated with its use, whereas for the link objects, trust relationship is established by default.

- Another difference is related to the processing of forwarding label and link objects at the ICN routers. Link object is processed only if the router cannot find a matching FIB entry for the content identifier. On the other hand, forwarding label is processed before a content identifier, if its use is enabled.

- Forwarding label can be enabled as part of a service, limiting its use to the supported namespaces and requiring its use whenever

supported. Link object is more of an application driven component and network service agnostic, allowing the network to decide on its use.

- Forwarding label can be considered as an enabler for faster packet processing at the ICN routers and optimized routing to a content source, whereas link object can be considered as a hint towards where to find the content. Since it is processed after FIB lookup on the content identifier fails, it typically leads to lower computational and bandwidth efficiency.

- As a link object can encode multiple routing hints, it can direct a request towards multiple identifier locations, giving an ICN router the option to choose any one of them based on the router's forwarding strategy. This selection is shared between consecutive hosts, but not enforced, which may lead to non-optimal forwarding paths. Forwarding label, on the other hand, is enforced consistently at consecutive hops within a domain whenever its use is supported.

4. Name Resolution System Considerations

To manage the AI to NI mapping, we need a name resolution system (NRS). In addition to exposing APIs to application to register its name to the NRS, it should also scale and work efficiently considering the scale of named resources that need to be published, resolved, removed, and updated at high frequency, for instance, corresponding to high-speed mobility scenarios.

The following are the design choices for the NRS:

- Hierarchical System: Here, AI to NI mapping is managed by the application providers, but similar to DNS, the service has to sync its name reachability information with high level name resolvers. NDNS is an example of such a system [NDNS]. This design is typically suitable in cases when resources are static, rather than for highly dynamic systems such as ICN, where replication and mobility will be the norm. Also, such system has to scale to resolve information objects in contrast to host resolution, which represents the current use.

- Integrated/Flat System: Here, resolution service is integrated within the ICN infrastructure, where the router contributes a part of its compute and storage resources to enable this service. This integration allows multiple ways of designing a generic name resolution service, similar to the designs for Global Name Resolution Service (GNRS) in MobilityFirst [GNS] [ASPC] [GNRS].

- Distributed System: Compared to the flat system, this type of architecture preserves the contextual nature of DNS, by using the context in the name to identify a home controller, where respective AI to NI mapping can be resolved. At the same time, such a system removes the need for home controllers to sync up with high level resolvers. For instance, /company/content-id would be mapped with a resolver named /company/resolver-id.

5. Differences with respect to Existing IP-based Proposals

To address persistent identity, routing scalability, multihoming, and mobility limitations of the current IP, various incremental solutions have been proposed, among which identifier/locator split emerged as the key solution to address these challenges [RFC4984]. Here, we specifically focus on three of these solutions: (i) Host Identity Protocol (HIP) [HIP], (ii) Identifier-Locator Network Protocol (ILNP) [ILNP], and Locator/Identifier Seperation Protocol (LISP) [RFC6830]. HIP and ILNP achieve ID/locator separation and binding at the host level whereas LISP achieves that at the network level (i.e., at the network edge using service routers).

In HIP, public cryptographic keys are used as host identifiers, which provide the binding to higher layer protocols instead of IP addresses [RFC7401]. ILNP divides IP namespace into two distinct namespaces of identifiers and locators, each of which carrying distinct semantics with identifier representing the non-topological name for the host and locator representing the topologically bound name for the network [RFC6740]. LISP is a map-and-encap type protocol, which achieves id/locator separation by defining (i) endpoint identifiers, which are used for routing at the access network and which represent the IP address for the host, and (ii) routing locators, which are used for routing at the core and which represent the IP address for the egress routers.

These protocols fundamentally differ from ICN’s objective to define a new network layer, where name based routing, location independent caching, mobility, multihoming, and multi-path routing are the integral features. More specifically, this draft proposes to enable AI/NI binding as a network service to allow efficient routing of user requests depending on the application context.

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

6.2. Informative References

[AFWD] Yi, C., Afanasyev, A., Wang, L., Zhang, B., and L. Zhang, "Adaptive Forwarding in Named Data Networking", ACM CCR, Jul 2012.

[ASPC] Sharma, A., Tie, X., Uppal, H., Venkataramani, A., Westbrook, D., and A. Yadav, "A Global Name Service for a Highly Mobile Internetwork", ACM SIGGCOM, 2014.

[CCN] Jacobson, V., Smetters, D., Thornton, J., Plass, M., Briggs, N., and R. Braynard, "Networking Named Content", ACM CoNEXT, 2009.

[FWLDR] Ravindran, R., Chakraborti, A., and A. Azgin, "Forwarding Label Support in CCN Protocol", draft-ravi-ccn-forwarding-label-02, March, 2016.

[FWLRP] Azgin, A., Ravindran, R., and G. Wang, "A Scalable Mobility-Centric Architecture for Named Data Networking", IEEE ICCCN Scene Workshop, 2014.

[GNRS] Hu, Y., Yates, R., and D. Raychaudhuri, "A Hierarchically Aggregated In-Network Global Name Resolution Service for the Mobile Internet".

[GNS] Venkataramani, A., Sharma, A., Tie, X., Uppal, H., Westbrook, D., Kurose, J., and D. Raychaudhuri, "Design Requirements for a Global Name Service for a Mobility-Centric, Trustworthy Internetwork", IEEE COMSNETS, 2013.

[HIP] Nikander, P., Gurtov, A., and T. Henderson, "Host identity protocol (HIP): Connectivity, mobility, multi-homing, security, and privacy over IPv4 and IPv6 networks", IEEE Communications Surveys and Tutorials, pp: 186-204, 2010.

[ILNP] Atkinson, R., "An Overview of the Identifier-Locator Network Protocol (ILNP)", Technical Report, University College London, 2005.

[MAAS] Azgin, A., Ravindran, R., Chakraborti, A., and G. Wang, "Seamless Producer Mobility as a Service in Information Centric Networks", ACM ICN IC5G Workshop, 2016.

[MDHT] Liu, H., Foy, X., and D. Zhang, "A Multi-level DHT routing Framework with Aggregation", ACM SIGCOMM ICN Workshop, 2012.

[MFRST] Venkataramani, A., Kurose, J., Raychaudhuri, D., Nagaraja, K., Mao, M., and S. Banerjee, "MobilityFirst: A Mobility-centric and Trustworthy Internet Architecture", ACM SIGCOMM CCR, 2014.

[MPNCP] Afanasyev, A., Yi, C., Wang, L., Zhang, B., and L. Zhang, "Map-and-Encap for Scaling NDN Routing", NDN Technical Report, ndn-004-02, 2015.

[NAMES] Baid, A., Vu, T., and D. Raychaudhuri, "Comparing Alternative Approaches for Networking of Named Objects in the Future Internet", IEEE INFOCOM NOMEN Workshop, 2012.

[NCMP] Adhatarao, S., Chen, J., Arumaithurai, M., Fu, X., and K. Ramakrishnan, "Comparison of Naming Schema in ICN", IEEE LANMAN, 2016.

[NDNS] Afanasyev, A., "Addressing Operational Challenges in Named Data Networking Through NDNS Distributed Database", 2013.

[QCMP] Gao, Z., Venkataramani, A., Kurose, J., and S. Heimlicher, "Towards a Quantitative Comparison of Location-Independent Network Architectures", ACM SIGCOMM, 2014.

[RFC2629] Rose, M., "Writing I-Ds and RFCs using XML", RFC 2629, DOI 10.17487/RFC2629, June 1999, <http://www.rfc-editor.org/info/rfc2629>.

[RFC3220] Perkins, C., "IP Mobility Support for IPv4", RFC 3220, 2002.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <http://www.rfc-editor.org/info/rfc3552>.

[RFC4984] Meyer, D., Zhang, L., and K. Fall, "Report from the IAB Workshop on Routing and Addressing", RFC 4984, 2007.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC6740] Atkinson, R. and S. Bhatti, "Identifier-Locator Network Protocol (ILNP) Architectural Description", RFC 6740, 2012.

[RFC6830] Farinacci, D., Fuller, V., Meyer, D., and D. Lewis, "The Locator/ID Separation Protocol (LISP)", RFC 6830, 2013.

[RFC7401] Moskowitz, R., Heer, T., Jokela, P., and T. Henderson, "Host Identity Protocol Version 2 (HIPv2)", RFC 7401, 2015.

[SFWD] Yuan, H., Song, T., and P. Crowley, "Scalable NDN Forwarding: Concepts, Issues and Principles", IEEE ICCCN, 2012.

[SNAMP] Afanasyev, A., Yi, C., Wang, L., Zhang, B., and L. Zhang, "SNAMP: Secure Namespace Mapping to Scale NDN Forwarding", IEEE Global Internet Symposium, 2015.

[SPIT] Yuan, H. and P. Crowley, "Scalable Pending Interest Table Design: From Principles to Practice", IEEE INFOCOM, 2014.

[VRSGN] "Verisign Domain Name Industry Brief", July 2016.

[WPIT] Varvello, M., Perino, D., and L. Linguaglossa, "On the Design and Implementation of a Wire-speed Pending Interest Table", IEEE INFOCOM NOMEN Workshop, 2013.

Appendix A. Additional Stuff

This becomes an Appendix.

Authors' Addresses

Aytac Azgin
Huawei Technologies
Santa Clara, CA 95050
USA

Email: aytac.azgin@huawei.com

Ravishankar Ravindran
Huawei Technologies
Santa Clara, CA 95050
USA

Email: ravi.ravindran@huawei.com

ICNRG
Internet-Draft
Intended status: Experimental
Expires: July 28, 2019

M. Mosko
PARC, Inc.
I. Solis
LinkedIn
C. Wood
University of California Irvine

January 24, 2019

CCNx Messages in TLV Format
draft-irtf-icnrg-ccnxmessages-09

Abstract

This document specifies the encoding of CCNx messages in a TLV packet format, including the TLV types used by each message element and the encoding of each value. The semantics of CCNx messages follow the encoding-independent CCNx Semantics specification.

This document is a product of the Information Centric Networking research group (ICNRG).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 28, 2019.

Copyright Notice

Table of Contents

1. Introduction

This document specifies a Type-Length-Value (TLV) packet format and the TLV type and value encodings for CCNx messages. A full description of the CCNx network protocol, providing an encoding-free description of CCNx messages and message elements, may be found in [CCNSemantics]. CCNx is a network protocol that uses a hierarchical name to forward requests and to match responses to requests. It does not use endpoint addresses, such as Internet Protocol. Restrictions in a request can limit the response by the public key of the response's signer or the cryptographic hash of the response. Every CCNx forwarder along the path does the name matching and restriction checking. The CCNx protocol fits within the broader framework of Information Centric Networking (ICN) protocols [RFC7927].

This document describes a TLV scheme using a fixed 2-byte T and a fixed 2-byte L field. The rational for this choice is described in Section 5. Briefly, this choice avoids multiple encodings of the same value (aliases) and reduces the work of a validator to ensure compliance. Unlike some uses of TLV in networking, the each network hop must evaluate the encoding, so even small validation latencies at each hop could add up to a large overall forwarding delay. For very small packets or low throughput links, where the extra bytes may become a concern, one may use a TLV compression protocol, for example [compress] and [CCNxz].

This document specifies:

- The TLV packet format.
- The overall packet format for CCNx messages.
- The TLV types used by CCNx messages.
- The encoding of values for each type.
- Top level types that exist at the outermost containment.

- Interest TLVs that exist within Interest containment.

- Content Object TLVs that exist within Content Object containment.

This document is supplemented by this document:

- Message semantics: see [CCNSemantics] for the protocol operation regarding Interest and Content Object, including the Interest Return protocol.

- URI notation: see [CCNxURI] for the CCNx URI notation.

The type values in Section 4 represent the values in common usage today. These values may change pending IANA assignments. All type values are relative to their parent containers. For example, each level of a nested TLV structure might define a "type = 1" with a completely different meaning. In the following, we use the symbolic names defined in that section.

Packets are represented as 32-bit wide words using ASCII art. Due to the nested levels of TLV encoding and the presence of optional fields and variable sizes, there is no concise way to represent all possibilities. We use the convention that ASCII art fields enclosed by vertical bars "|" represent exact bit widths. Fields with a forward slash "/" are variable bit widths, which we typically pad out to word alignment for picture readability.

The document represents the consensus of the ICN RG. It is the first ICN protocol from the RG, created from the early CCNx protocol [nnc] with significant revision and input from the ICN community and RG members. The draft has received critical reading by several members of the ICN community and the RG. The authors and RG chairs approve of the contents. The document is sponsored under the IRTF and is not issued by the IETF and is not an IETF standard. This is an experimental protocol and may not be suitable for any specific application and the specification may change in the future.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Definitions

- Name: A hierarchically structured variable length identifier. It is an ordered list of path segments, which are variable length octet strings. In human-readable form, it is represented in URI

format as ccnx:/path/part. There is no host or query string. See [CCNxURI] for complete details.

- Interest: A message requesting a Content Object with a matching Name and other optional selectors to choose from multiple objects with the same Name. Any Content Object with a Name and attributes that matches the Name and optional selectors of the Interest is said to satisfy the Interest.

- Content Object: A data object sent in response to an Interest request. It has an optional Name and a content payload that are bound together via cryptographic means.

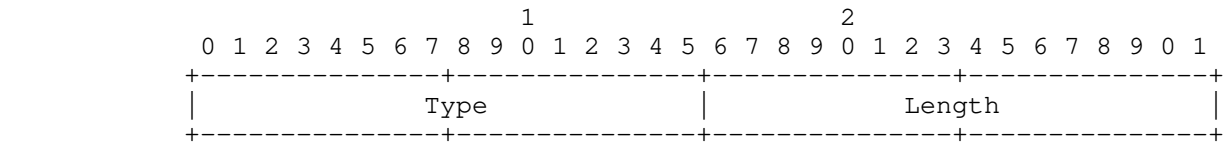
3. Type-Length-Value (TLV) Packets

We use 16-bit Type and 16-bit Length fields to encode TLV based packets. This provides 64K different possible types and value field lengths of up to 64KiB. With 64K possible types at each level of TLV encoding, there should be sufficient space for basic protocol types, while also allowing ample room for experimentation, application use, vendor extensions, and growth. This encoding does not allow for jumbo packets beyond 64 KiB total length. If used on a media that allows for jumbo frames, we suggest defining a media adapation envelope that allows for multiple smaller frames.

There are several global TLV definitions that we reserve at all hierarchical contexts. The TLV types in the range 0x1000 - 0x1FFF are reserved for experimental use. The TLV type T_ORG is also reserved for vendor extensions (see Section 3.3.2). The TLV type T_PAD is used to optionally pad a field out to some desired alignment.

Abbrev	Name	Description
T_ORG	Vendor Specific Information (Section 3.3.2)	Information specific to a vendor implementation (see below).
T_PAD	Padding (Section 3.3.1)	Adds padding to a field (see below).
n/a	Experimental	Experimental use.

Table 1: Reserved TLV Types



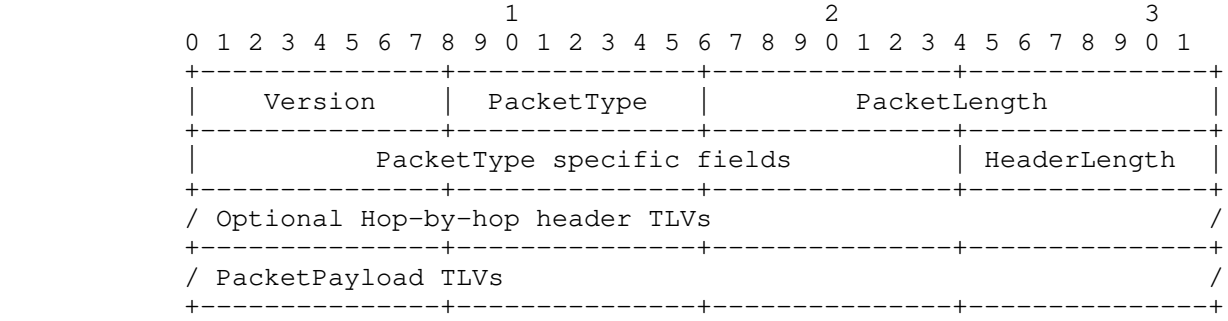
The Length field contains the length of the Value field in octets. It does not include the length of the Type and Length fields. The length MAY be zero.

TLV structures are nestable, allowing the Value field of one TLV structure to contain additional TLV structures. The enclosing TLV structure is called the container of the enclosed TLV.

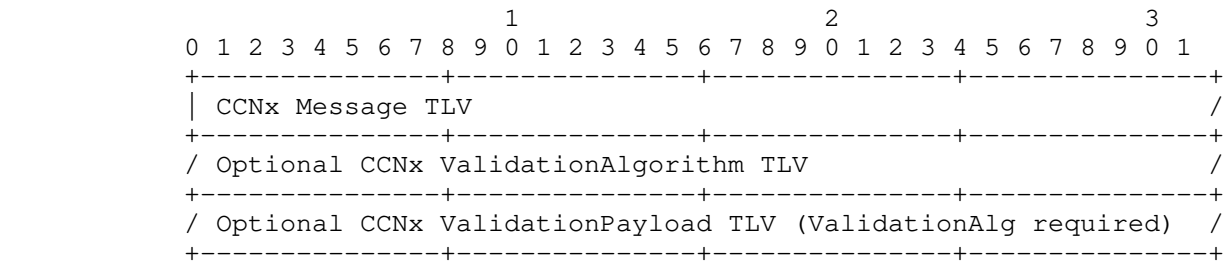
Type values are context-dependent. Within a TLV container, one may re-use previous type values for new context-dependent purposes.

3.1. Overall packet format

Each packet includes the 8 byte fixed header, described below, followed by a set of TLV fields. These fields are optional hop-by-hop headers and the Packet Payload.



The packet payload is a TLV encoding of the CCNx message, followed by optional Validation TLVs.



This document describes the Version "1" TLV encoding.

After discarding the fixed and hop-by-hop headers the remaining PacketPayload should be a valid protocol message. Therefore, the PacketPayload always begins with 4 bytes of type-length that specifies the protocol message (whether it is an Interest, Content Object, or other message type) and its total length. The embedding of a self-sufficient protocol data unit inside the fixed and hop-by-hop headers allows a network stack to discard the headers and operate only on the embedded message. It also de-couples the PacketType field -- which specifies how to forward the packet -- from the PacketPayload.

The range of bytes protected by the Validation includes the CCNx Message and the ValidationAlgorithm.

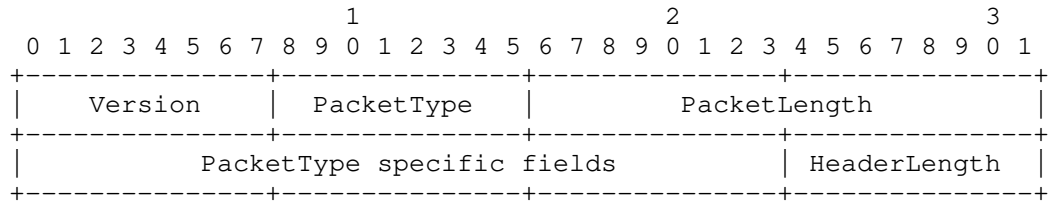
The ContentObjectHash begins with the CCNx Message and ends at the tail of the packet.

3.2. Fixed Headers

CCNx messages begin with an 8 byte fixed header (non-TLV format). The HeaderLength field represents the combined length of the Fixed and Hop-by-hop headers. The PacketLength field represents the entire Packet length from the first byte of Version to the last byte of the packet.

A specific PacketType may assign meaning to the "PacketType specific fields," which are otherwise reserved. For the three defined PacketTypes (Interest, ContentObject, and InterestReturn), we define those values in this document.

The PacketPayload of a CCNx packet is the protocol message itself. The Content Object Hash is computed over the PacketPayload only, excluding the fixed and hop-by-hop headers as those might change from hop to hop. Signed information or Similarity Hashes should not include any of the fixed or hop-by-hop headers. The PacketPayload should be self-sufficient in the event that the fixed and hop-by-hop headers are removed.



- Version: defines the version of the packet.
- HeaderLength: The length of the fixed header (8 bytes) and hop-by-hop headers. The minimum value MUST be "8".
- PacketType: describes forwarder actions to take on the packet.
- PacketLength: Total octets of packet including all headers (fixed header plus hop-by-hop headers) and protocol message.
- PacketType Specific Fields: specific PacketTypes define the use of these bits.

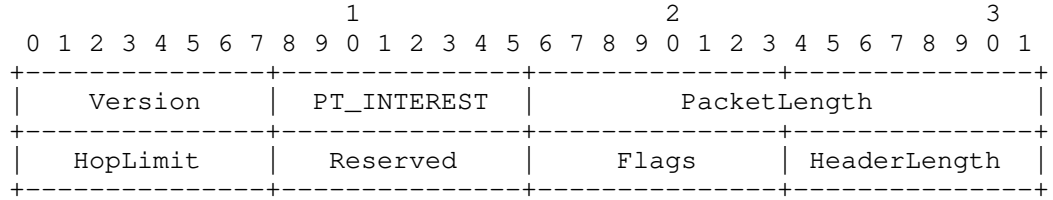
The PacketType field indicates how the forwarder should process the packet. A Request Packet (Interest) has PacketType PT_INTEREST, a Response (Content Object) has PacketType PT_CONTENT, and an InterestReturn has PacketType PT_RETURN.

HeaderLength is the number of octets from the start of the packet (Version) to the end of the hop-by-hop headers. PacketLength is the number of octets from the start of the packet to the end of the packet. Both lengths have a minimum value of 8 (the fixed header itself).

The PacketType specific fields are reserved bits whose use depends on the PacketType. They are used for network-level signaling.

3.2.1. Interest Fixed Header

If the PacketType is PT_INTEREST, it indicates that the PacketPayload should be processed as an Interest message. For this type of packet, the Fixed Header includes a field for a HopLimit as well as Reserved and Flags fields. The Reserved field MUST be set to 0 in an Interest - this field will be set to a return code in the case of an Interest Return. There are currently no Flags defined, so this field MUST be set to 0.



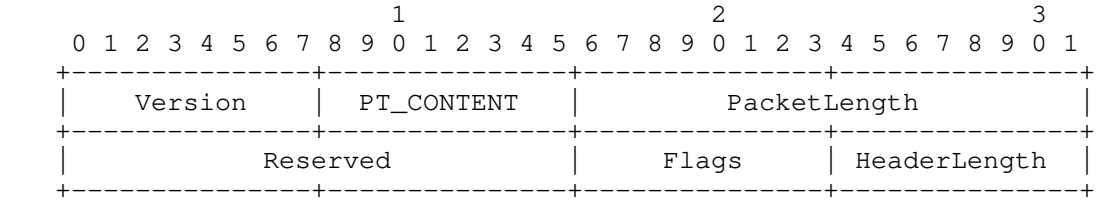
3.2.1.1. Interest HopLimit

For an Interest message, the HopLimit is a counter that is decremented with each hop. It limits the distance an Interest may travel on the network. The node originating the Interest MAY put in any value - up to the maximum of 255. Each node that receives an Interest with a HopLimit decrements the value upon reception. If the value is 0 after the decrement, the Interest MUST NOT be forwarded off the node.

It is an error to receive an Interest with a 0 hop-limit from a remote node.

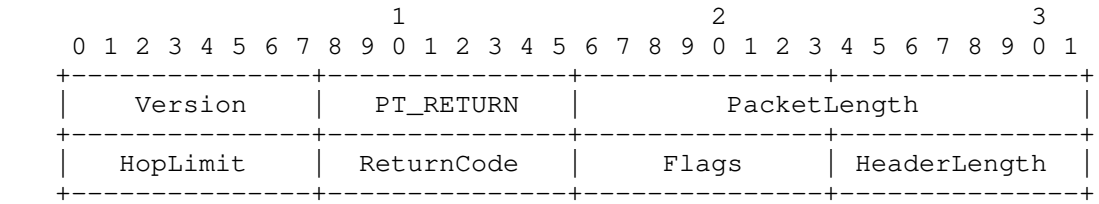
3.2.2. Content Object Fixed Header

If the PacketType is PT_CONTENT, it indicates that the PacketPayload should be processed as a Content Object message. A Content Object defines a Flags field, however there are currently no flags defined, so the Flags field must be set to 0.



3.2.3. InterestReturn Fixed Header

If the PacketType is PT_RETURN, it indicates that the PacketPayload should be processed as a returned Interest message. The only difference between this InterestReturn message and the original Interest is that the PacketType is changed to PT_RETURN and a ReturnCode is is put into the ReturnCode field. All other fields are unchanged from the Interest packet. The purpose of this encoding is to prevent packet length changes so no additional bytes are needed to return an Interest to the previous hop. See [CCNSemantics] for a protocol description of this packet type.



3.2.3.1. InterestReturn HopLimit

This is the original Interest's HopLimit, as received. It is the value before being decremented at the current node (i.e. the received value).

3.2.3.2. InterestReturn Flags

These are the original Flags as set in the Interest.

3.2.3.3. Return Code

The numeric value assigned to the return types is defined below. This value is set by the node creating the Interest Return.

A return code of "0" MUST NOT be used, as it indicates that the returning system did not modify the Return Code field.

Type	Return Type
T_RETURN_NO_ROUTE	No Route
T_RETURN_LIMIT_EXCEEDED	Hop Limit Exceeded
T_RETURN_NO_RESOURCES	No Resources
T_RETURN_PATH_ERROR	Path Error
T_RETURN_PROHIBITED	Prohibited
T_RETURN_CONGESTED	Congested
T_RETURN_MTU_TOO_LARGE	MTU too large
T_RETURN_UNSUPPORTED_HASH_RESTRICTION	Unsupported ContentObjectHashRestriction
T_RETURN_MALFORMED_INTEREST	Malformed Interest

Table 2: Return Codes

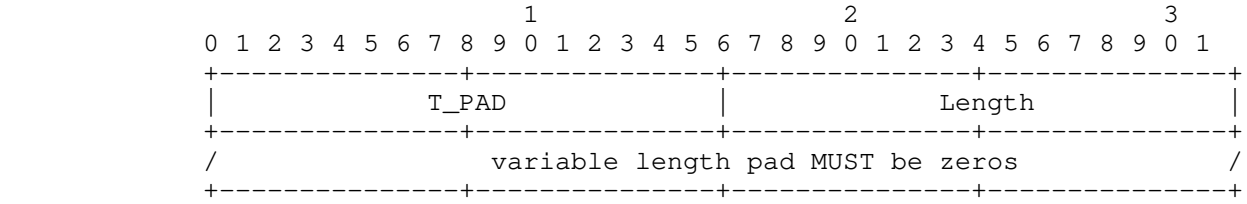
3.3. Global Formats

This section defines global formats that may be nested within other TLVs.

3.3.1. Pad

The pad type may be used by protocols that prefer word-aligned data. The size of the word may be defined by the protocol. Padding 4-byte words, for example, would use a 1-byte, 2-byte, and 3-byte Length. Padding 8-byte words would use a (0, 1, 2, 3, 5, 6, 7)-byte Length.

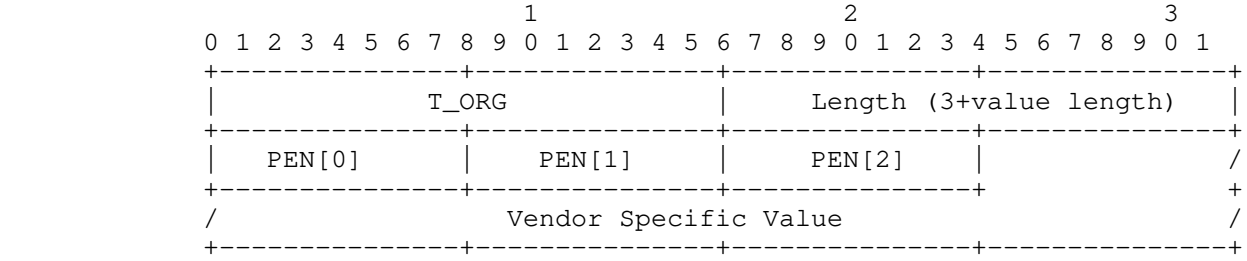
One MUST NOT pad inside a Name. Apart from that, a pad MAY be inserted after any other TLV in the CCNx Message or in the Validation Dependent Data In the remainder of this document, we will not show optional pad TLVs.



3.3.2. Organization Specific TLVs

Organization specific TLVs (also known as Vendor TLVs) MUST use the T_ORG type. The Length field is the length of the organization specific information plus 3. The Value begins with the 3 byte organization number derived from the last three digits of the IANA Private Enterprise Numbers [EpriseNumbers], followed by the organization specific information.

A T_ORG MAY be used as a path segment in a Name, in which case it is a regular path segment and is part of the regular name matching.



3.3.3. Hash Format

Hash values are used in several fields throughout a packet. This TLV encoding is commonly embedded inside those fields to specify the specific hash function used and it's value. Note that the reserved

TLV types are also reserved here for user-defined experimental functions.

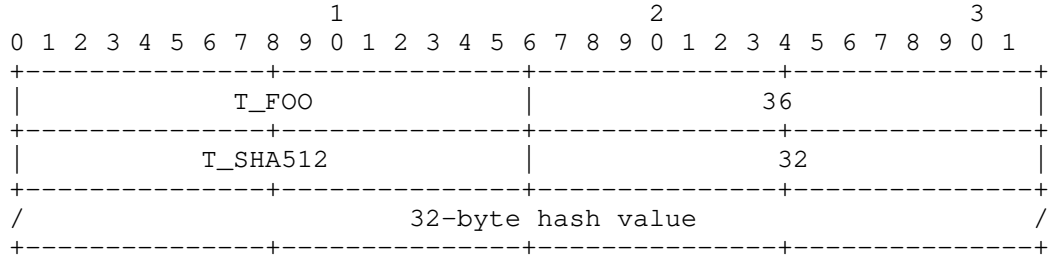
The LENGTH field of the hash value MUST be less than or equal to the hash function length. If the LENGTH is less than the full length, it is taken as the left LENGTH bytes of the hash function output. Only specified truncations are allowed, not arbitrary truncations.

This nested format is used because it allows binary comparison of hash values for certain fields without a router needing to understand a new hash function. For example, the KeyIdRestriction is bit-wise compared between an Interest's KeyIdRestriction field and a ContentObject's KeyId field. This format means the outer field values do not change with differing hash functions so a router can still identify those fields and do a binary comparison of the hash TLV without need to understand the specific hash used. An alternative approach, such as using T_KEYID_SHA512-256, would require each router keep an up-to-date parser and supporting user-defined hash functions here would explode the parsing state-space.

A CCNx entity MUST support the hash type T_SHA-256. An entity MAY support the remaining hash types.

Abbrev	Lengths (octets)
T_SHA-256	32
T_SHA-512	64, 32
n/a	Experimental TLV types

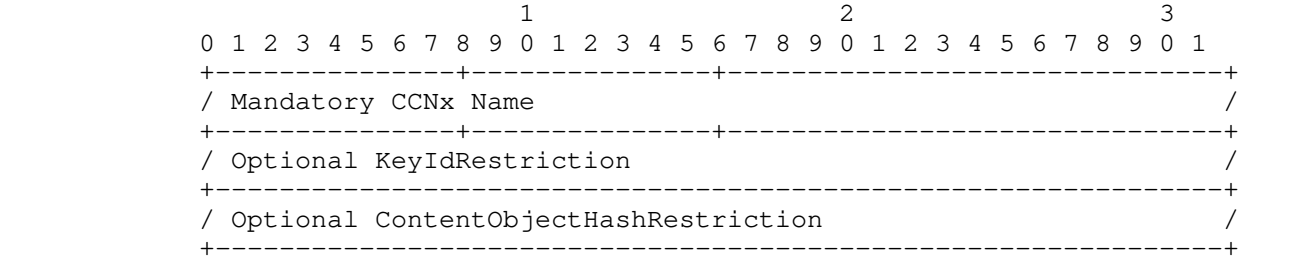
Table 3: CCNx Hash Functions



Example nesting inside type T_FOO

3.3.4. Link

A Link is the tuple: {Name, [KeyIdRestr], [ContentObjectHashRestr]}. It is a general encoding that is used in both the payload of a Content Object with PayloadType = "Link" and in the KeyLink field in a KeyLocator. A Link is essentially the body of an Interest.



3.4. Hop-by-hop TLV headers

Hop-by-hop TLV headers are unordered and meaning MUST NOT be attached to their ordering. Three hop-by-hop headers are described in this document:

Abbrev	Name	Description
T_INTLIFE	Interest Lifetime (Section 3.4.1)	The time an Interest should stay pending at an intermediate node.
T_CACHETIME	Recommended Cache Time (Section 3.4.2)	The Recommended Cache Time for Content Objects.
T_MSGHASH	Message Hash (Section 3.4.3)	The hash of the CCNx Message to end of packet using Section 3.3.3 format.

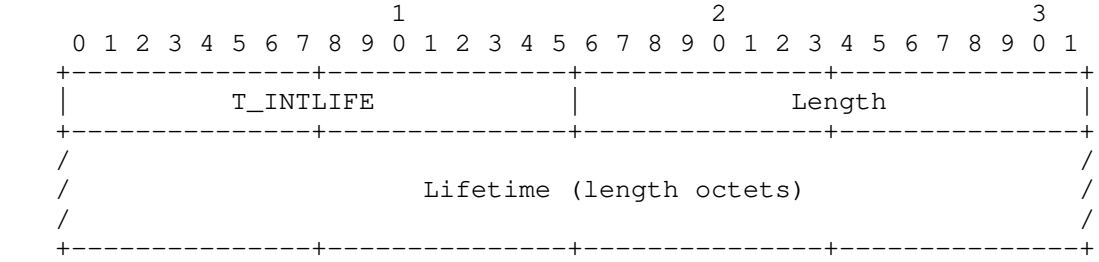
Table 4: Hop-by-hop Header Types

Additional hop-by-hop headers are defined in higher level specifications such as the fragmentation specification.

3.4.1. Interest Lifetime

The Interest Lifetime is the time that an Interest should stay pending at an intermediate node. It is expressed in milliseconds as an unsigned, network byte order integer.

A value of 0 (encoded as 1 byte %x00) indicates the Interest does not elicit a Content Object response. It should still be forwarded, but no reply is expected and a forwarder could skip creating a PIT entry.

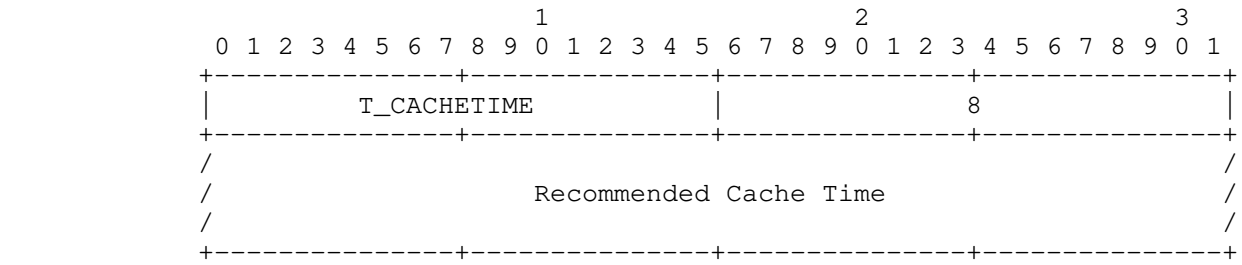


3.4.2. Recommended Cache Time

The Recommended Cache Time (RCT) is a measure of the useful lifetime of a Content Object as assigned by a content producer or upstream node. It serves as a guideline to the Content Store cache in determining how long to keep the Content Object. It is a recommendation only and may be ignored by the cache. This is in contrast to the ExpiryTime (described in Section 3.6.2.2.2) which takes precedence over the RCT and must be obeyed.

Because the Recommended Cache Time is an optional hop-by-hop header and not a part of the signed message, a content producer may re-issue a previously signed Content Object with an updated RCT without needing to re-sign the message. There is little ill effect from an attacker changing the RCT as the RCT serves as a guideline only.

The Recommended Cache Time (a millisecond timestamp) is a network byte ordered unsigned integer of the number of milliseconds since the epoch in UTC of when the payload expires. It is a 64-bit field.



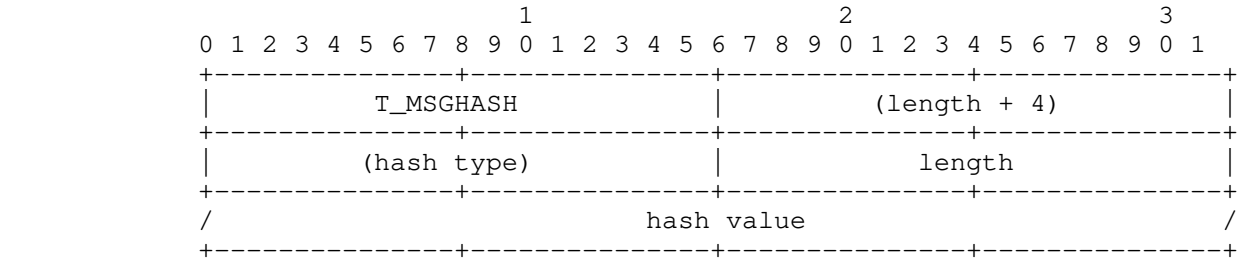
3.4.3. Message Hash

Within a trusted domain, an operator may calculate the message hash at a border device and insert that value into the hop-by-hop headers of a message. An egress device should remove the value. This permits intermediate devices within that trusted domain to match against a ContentObjectHashRestriction without calculating it at every hop.

The message hash is a cryptographic hash from the start of the CCNx Message to the end of the packet. It is used to match against the ContentObjectHashRestriction (Section 3.6.2.1.2). The Message Hash may be of longer length than an Interest's restriction, in which case the device should use the left bytes of the Message Hash to check against the Interest's value.

The Message Hash may only carry one hash type and there may only be one Message Hash header.

The Message Hash header is unprotected, so this header is only of practical use within a trusted domain, such as an operator's autonomous system.



Message Hash Header

3.5. Top-Level Types

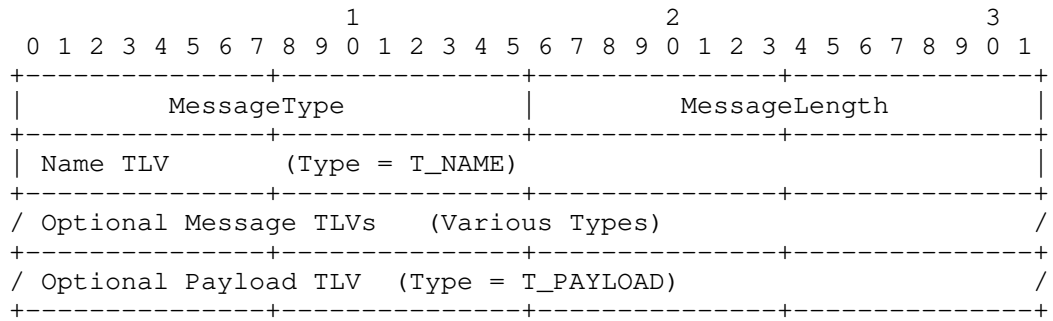
The top-level TLV types listed below exist at the outermost level of a CCNx protocol message.

Abbrev	Name	Description
T_INTEREST	Interest (Section 3.6)	An Interest MessageType.
T_OBJECT	Content Object (Section 3.6)	A Content Object MessageType
T_VALIDATION_ALG	Validation Algorithm (Section 3.6.4.1)	The method of message verification such as Message Integrity Check (MIC), a Message Authentication Code (MAC), or a cryptographic signature.
T_VALIDATION_PAYLOAD	Validation Payload (Section 3.6.4.2)	The validation output, such as the CRC32C code or the RSA signature.

Table 5: CCNx Top Level Types

3.6. CCNx Message

This is the format for the CCNx protocol message itself. The CCNx message is the portion of the packet between the hop-by-hop headers and the Validation TLVs. The figure below is an expansion of the "CCNx Message TLV" depicted in the beginning of Section 3. The CCNx message begins with MessageType and runs through the optional Payload. The same general format is used for both Interest and Content Object messages which are differentiated by the MessageType field. The first enclosed TLV of a CCNx Message is always the Name TLV. This is followed by an optional Message TLVs and an optional Payload TLV.



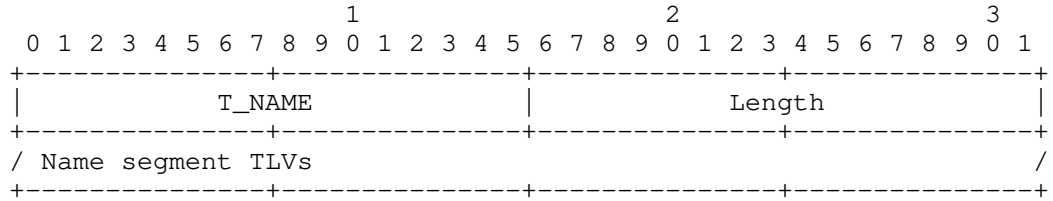
Abbrev	Name	Description
T_NAME	Name (Section 3.6.1)	The CCNx Name requested in an Interest or published in a Content Object.
T_PAYLOAD	Payload (Section 3.6.3)	The message payload.

Table 6: CCNx Message Types

3.6.1. Name

A Name is a TLV encoded sequence of segments. The table below lists the type values appropriate for these Name segments. A Name MUST NOT include PAD TLVs.

As described in CCNx Semantics [CCNSemantics], using the CCNx URI [CCNxURI] notation, a T_NAME with 0 length corresponds to ccnx:/ (the default route) and is distinct from a name with one zero length segment, such as ccnx:/NAME=. In the TLV encoding, ccnx:/ corresponds to T_NAME with 0 length, while ccnx:/NAME= corresponds to T_NAME with 4 length and T_NAMESEGMENT with 0 length.



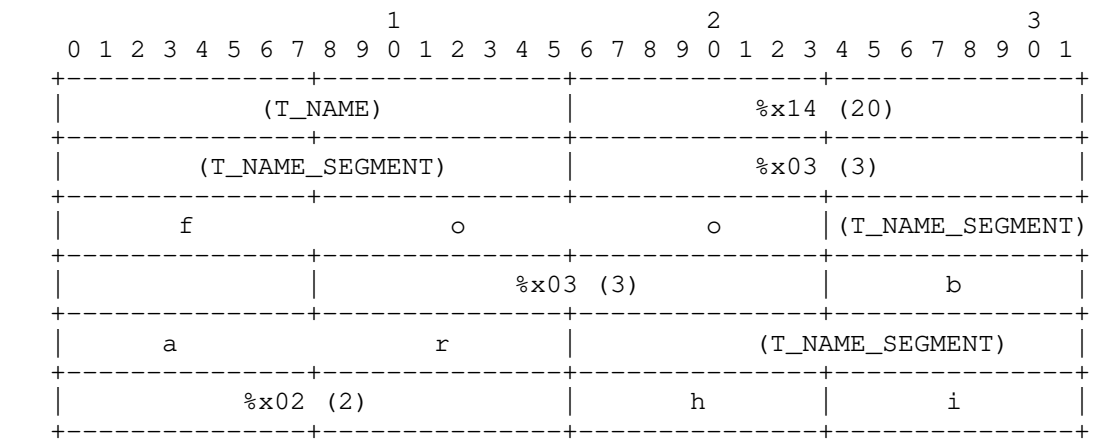
Symbolic Name	Name	Description
T_NAMESEGMENT	Name segment (Section 3.6.1.1)	A generic name Segment.
T_IPID	Interest Payload ID (Section 3.6.1.2)	An identifier that represents the Interest Payload field. As an example, the Payload ID might be a hash of the Interest Payload. This provides a way to differentiate between Interests based on their payloads without having to parse all the bytes of the payload itself; instead using only this Payload ID Name segment.
T_APP:00 - T_APP:4096	Application Components (Section 3.6.1.1)	Application-specific payload in a name segment. An application may apply its own semantics to the 4096 reserved types.

Table 7: CCNx Name Types

3.6.1.1. Name Segments

4096 special application payload name segments are allocated. These have application semantics applied to them. A good convention is to put the application's identity in the name prior to using these name segments.

For example, a name like "ccnx:/foo/bar/hi" would be encoded as:



3.6.1.2. Interest Payload ID

The InterestPayloadID is a name segment created by the origin of an Interest to represent the Interest Payload. This allows the proper multiplexing of Interests based on their name if they have different payloads. A common representation is to use a hash of the Interest Payload as the InterestPayloadID.

As part of the TLV 'value', the InterestPayloadID contains a one identifier of method used to create the InterestPayloadID followed by a variable length octet string. An implementation is not required to implement any of the methods to receive an Interest; the InterestPayloadID may be treated only as an opaque octet string for purposes of multiplexing Interests with different payloads. Only a device creating an InterestPayloadID name segment or a device verifying such a segment need to implement the algorithms.

It uses the Section 3.3.3 encoding of hash values.

In normal operations, we recommend displaying the InterestPayloadID as an opaque octet string in a CCNx URI, as this is the common denominator for implementation parsing.

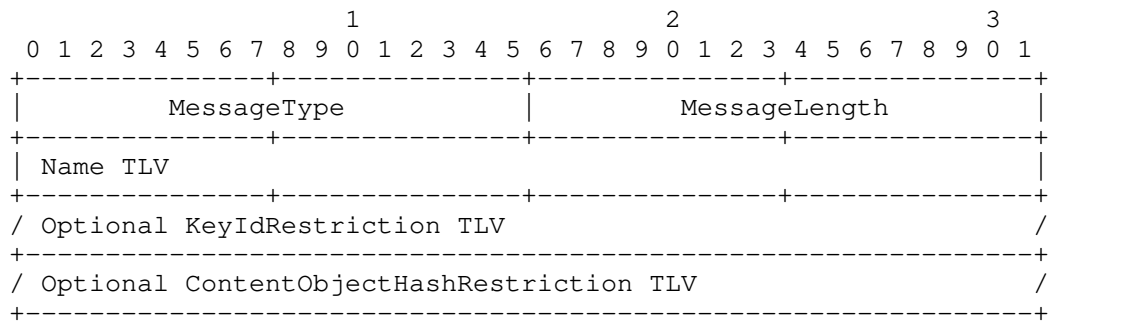
The InterestPayloadID, even if it is a hash, should not convey any security context. If a system requires confirmation that a specific entity created the InterestPayload, it should use a cryptographic signature on the Interest via the ValidationAlgorithm and ValidationPayload or use its own methods inside the Interest Payload.

3.6.2. Message TLVs

Each message type (Interest or Content Object) is associated with a set of optional Message TLVs. Additional specification documents may extend the types associated with each.

3.6.2.1. Interest Message TLVs

There are two Message TLVs currently associated with an Interest message: the KeyIdRestriction selector and the ContentObjectHashRestr selector are used to narrow the universe of acceptable Content Objects that would satisfy the Interest.



Abbrev	Name	Description
T_KEYIDRESTR	KeyIdRestriction (Section 3.6.2.1.1)	A Section 3.3.3 representation of the KeyId
T_OBJHASHRESTR	ContentObjectHashRestriction (Section 3.6.2.1.2)	A Section 3.3.3 representation of the hash of the specific Content Object that would satisfy the Interest.

Table 8: CCNx Interest Message TLV Types

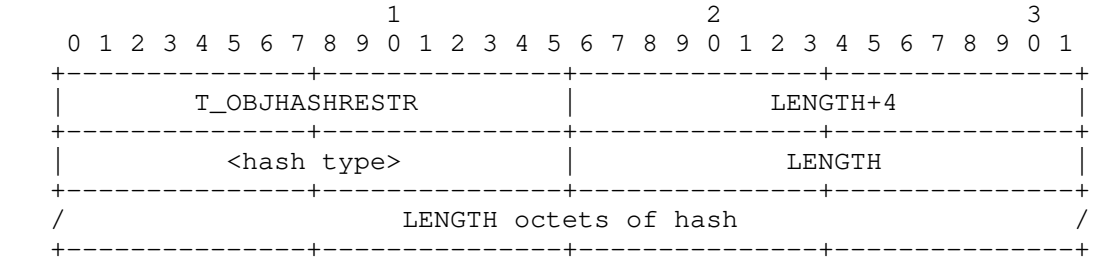
3.6.2.1.1. KeyIdRestriction

An Interest MAY include a KeyIdRestriction selector. This ensures that only Content Objects with matching KeyIds will satisfy the Interest. See Section 3.6.4.1.4.1 for the format of a KeyId.

3.6.2.1.2. ContentObjectHashRestriction

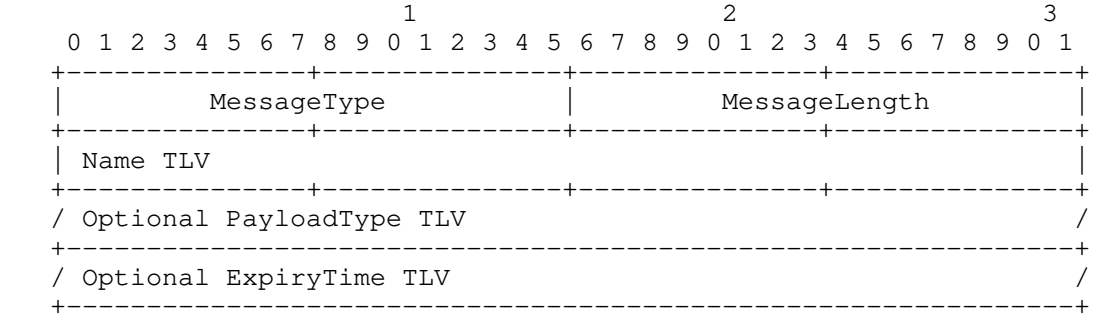
An Interest MAY contain a ContentObjectHashRestriction selector. This is the hash of the Content Object - the self-certifying name restriction that must be verified in the network, if an Interest carried this restriction. It is calculated from the beginning of the CCNx Message to the end of the packet. The LENGTH MUST be from one of the allowed values for that hash (see Section 3.3.3).

The ContentObjectHashRestriction SHOULD be of type T_SHA-256 and of length 32 bytes.



3.6.2.2. Content Object Message TLVs

The following message TLVs are currently defined for Content Objects: PayloadType (optional) and ExpiryTime (optional).



Abbrev	Name	Description
T_PAYLDTYPE	PayloadType (Section 3.6.2.2.1)	Indicates the type of Payload contents.
T_EXPIRY	ExpiryTime (Section 3.6.2.2.2)	The time at which the Payload expires, as expressed in the number of milliseconds since the epoch in UTC. If missing, Content Object may be used as long as desired.

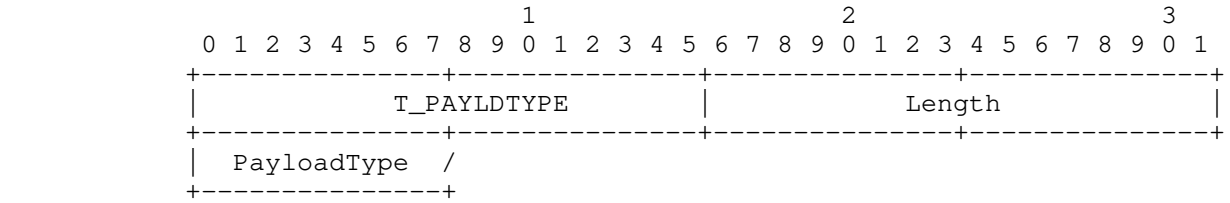
Table 9: CCNx Content Object Message TLV Types

3.6.2.2.1. PayloadType

The PayloadType is a network byte order integer representing the general type of the Payload TLV.

- T_PAYLOADTYPE_DATA: Data (possibly encrypted)
- T_PAYLOADTYPE_KEY: Key
- T_PAYLOADTYPE_LINK: Link

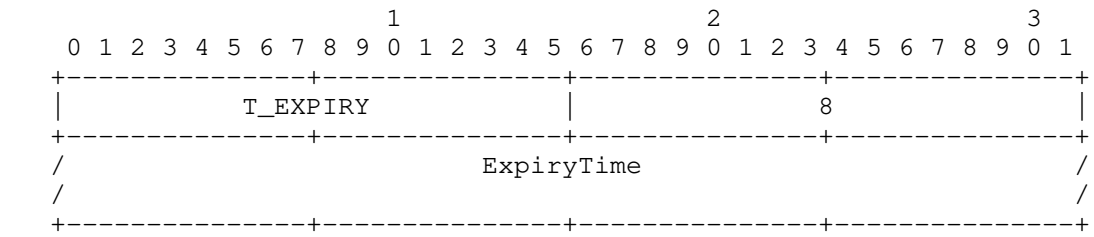
The Data type indicate that the Payload of the ContentObject is opaque application bytes. The Key type indicates that the Payload is a DER encoded public key. The Link type indicates that the Payload is one or more Link (Section 3.3.4). If this field is missing, a "Data" type is assumed.



3.6.2.2.2. ExpiryTime

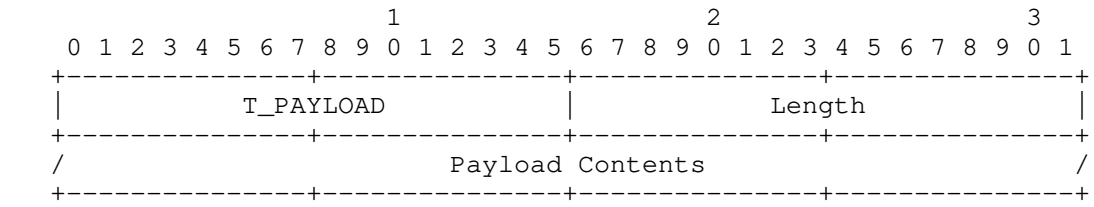
The ExpiryTime is the time at which the Payload expires, as expressed by a timestamp containing the number of milliseconds since the epoch in UTC. It is a network byte order unsigned integer in a 64-bit field. A cache or end system should not respond with a Content Object past its ExpiryTime. Routers forwarding a Content Object do

not need to check the ExpiryTime. If the ExpiryTime field is missing, the Content Object has no expressed expiration and a cache or end system may use the Content Object for as long as desired.



3.6.3. Payload

The Payload TLV contains the content of the packet. It MAY be of zero length. If a packet does not have any payload, this field MAY be omitted, rather than carrying a zero length.



3.6.4. Validation

Both Interests and Content Objects have the option to include information about how to validate the CCNx message. This information is contained in two TLVs: the ValidationAlgorithm TLV and the ValidationPayload TLV. The ValidationAlgorithm TLV specifies the mechanism to be used to verify the CCNx message. Examples include verification with a Message Integrity Check (MIC), a Message Authentication Code (MAC), or a cryptographic signature. The ValidationPayload TLV contains the validation output, such as the CRC32C code or the RSA signature.

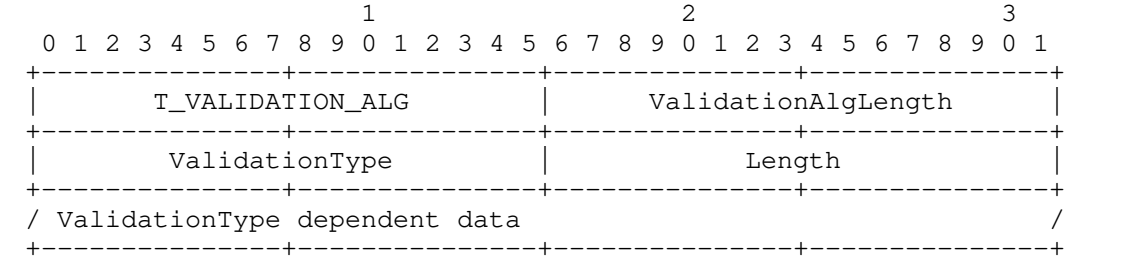
An Interest would most likely only use a MIC type of validation - a crc, checksum, or digest.

3.6.4.1. Validation Algorithm

The ValidationAlgorithm is a set of nested TLVs containing all of the information needed to verify the message. The outermost container has type = T_VALIDATION_ALG. The first nested TLV defines the specific type of validation to be performed on the message. The type

is identified with the "ValidationType" as shown in the figure below and elaborated in the table below. Nested within that container are the TLVs for any ValidationType dependent data, for example a Key Id, Key Locator etc.

Complete examples of several types may be found in Section 3.6.4.1.5



Abbrev	Name	Description
T_CRC32C	CRC32C (Section 3.6.4.1.1)	Castagnoli CRC32 (iSCSI, ext4, etc.), with normal form polynomial 0x1EDC6F41.
T_HMAC-SHA256	HMAC-SHA256 (Section 3.6.4.1.2)	HMAC (RFC 2104) using SHA256 hash.
T_RSA-SHA256	RSA-SHA256 (Section 3.6.4.1.3)	RSA public key signature using SHA256 digest.
EC-SECP-256K1	SECP-256K1 (Section 3.6.4.1.3)	Elliptic Curve signature with SECP-256K1 parameters (see [ECC]).
EC-SECP-384R1	SECP-384R1 (Section 3.6.4.1.3)	Elliptic Curve signature with SECP-384R1 parameters (see [ECC]).

Table 10: CCNx Validation Types

3.6.4.1.1. Message Integrity Checks

MICs do not require additional data in order to perform the verification. An example is CRC32C that has a "0" length value.

3.6.4.1.2. Message Authentication Checks

MACs are useful for communication between two trusting parties who have already shared private keys. Examples include an RSA signature of a SHA256 digest or others. They rely on a KeyId. Some MACs might use more than a KeyId, but those would be defined in the future.

3.6.4.1.3. Signature

Signature type Validators specify a digest mechanism and a signing algorithm to verify the message. Examples include RSA signature og a SHA256 digest, an Elliptic Curve signature with SECP-256K1 parameters, etc. These Validators require a KeyId and a mechanism for locating the publishers public key (a KeyLocator) - optionally a PublicKey or Certificate or KeyLink.

3.6.4.1.4. Validation Dependent Data

Different Validation Algorithms require access to different pieces of data contained in the ValidationAlgorithm TLV. As described above, Key Ids, Key Locators, Public Keys, Certificates, Links and Key Names all play a role in different Validation Algorithms. Any number of Validation Dependent Data containers can be present in a Validation Algorithm TLV.

Following is a table of CCNx ValidationType dependent data types:

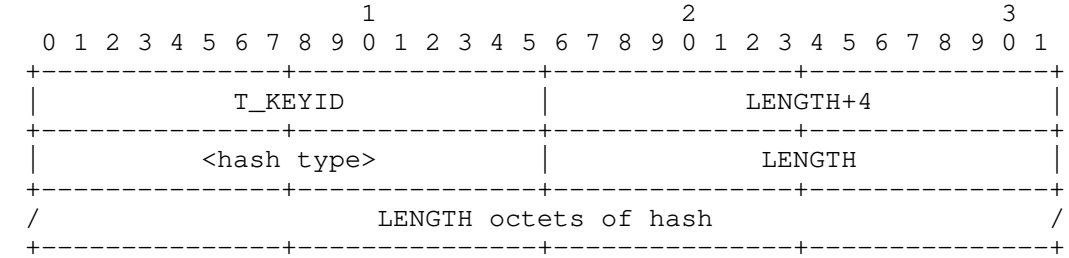
Abbrev	Name	Description
T_KEYID	SignerKeyId (Section 3.6.4.1.4.1)	An identifier of the shared secret or public key associated with a MAC or Signature.
T_PUBLICKEY	Public Key (Section 3.6.4.1.4.2)	DER encoded public key.
T_CERT	Certificate (Section 3.6.4.1.4.3)	DER encoded X509 certificate.
T_KEYLINK	KeyLink (Section 3.6.4.1.4.4)	A CCNx Link object.
T_SIGTIME	SignatureTime (Section 3.6.4.1.4.5)	A millsecond timestamp indicating the time when the signature was created.

Table 11: CCNx Validation Dependent Data Types

3.6.4.1.4.1. KeyId

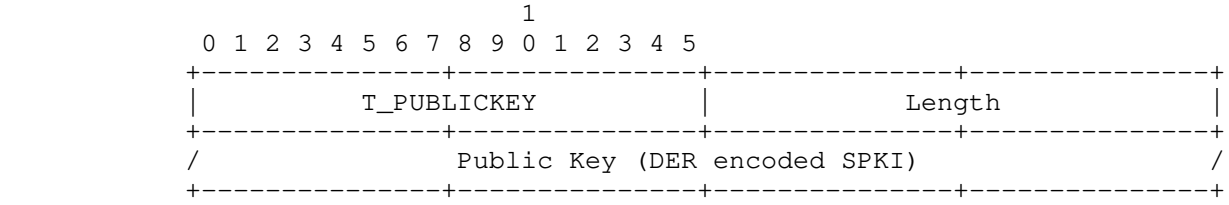
The KeyId is the publisher key identifier. It is similar to a Subject Key Identifier from X509 [RFC 5280, Section 4.2.1.2]. It should be derived from the key used to sign, such as from the SHA-256 hash of the key. It applies to both public/private key systems and to symmetric key systems.

The KeyId is represented using the Section 3.3.3. If a protocol uses a non-hash identifier, it should use one of the reserved values.

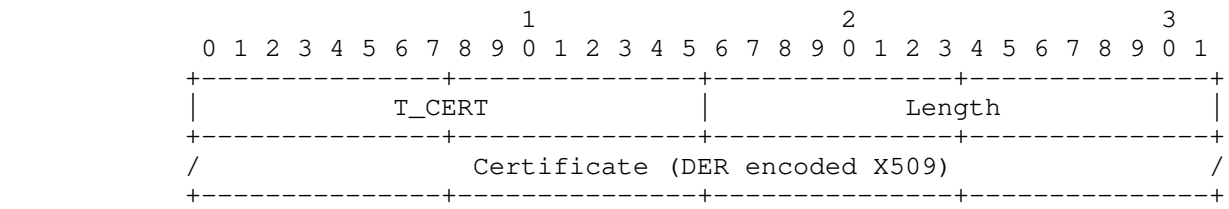


3.6.4.1.4.2. Public Key

A Public Key is a DER encoded Subject Public Key Info block, as in an X509 certificate.



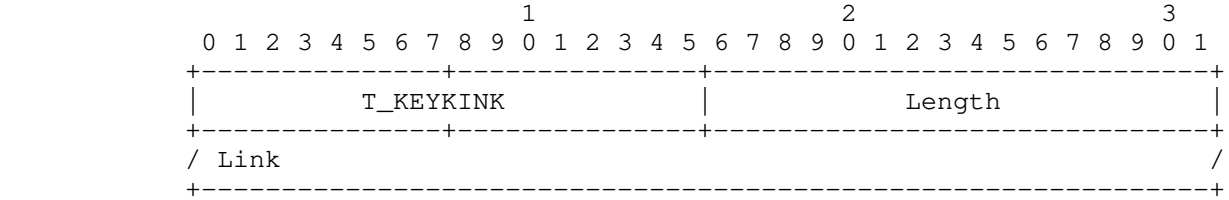
3.6.4.1.4.3. Certificate



3.6.4.1.4.4. KeyLink

A KeyLink type KeyLocator is a Link.

The KeyLink ContentObjectHashRestr, if included, is the digest of the Content Object identified by KeyLink, not the digest of the public key. Likewise, the KeyIdRestr of the KeyLink is the KeyId of the ContentObject, not necessarily of the wrapped key.

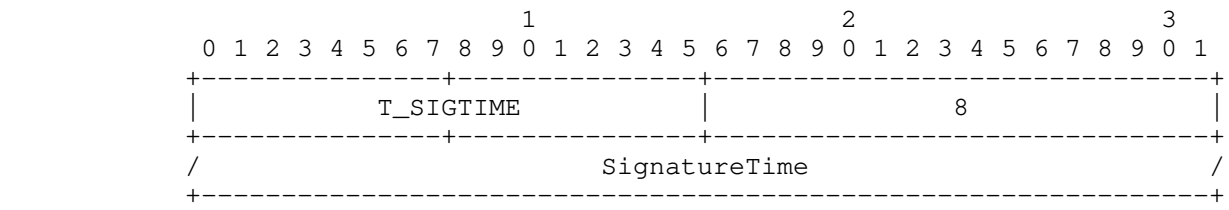


3.6.4.1.4.5. SignatureTime

The SignatureTime is a millisecond timestamp indicating the time at which a signature was created. The signer sets this field to the current time when creating a signature. A verifier may use this time to determine whether or not the signature was created during the validity period of a key, or if it occurred in a reasonable sequence with other associated signatures. The SignatureTime is unrelated to

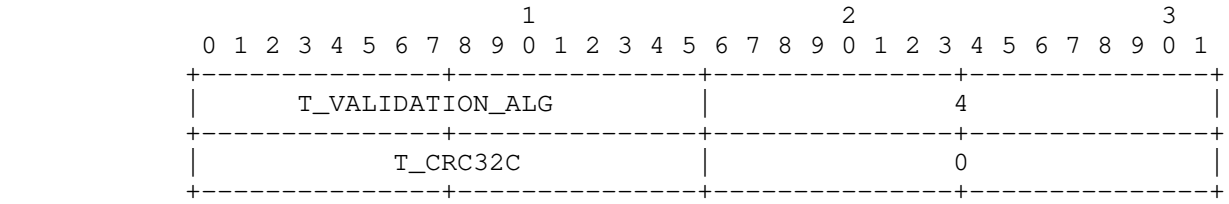
any time associated with the actual CCNx Message, which could have been created long before the signature. The default behavior is to always include a SignatureTime when creating an authenticated message (e.g. HMAC or RSA).

SignatureTime is a network byte ordered unsigned integer of the number of milliseconds since the epoch in UTC of when the signature was created. It is a fixed 64-bit field.

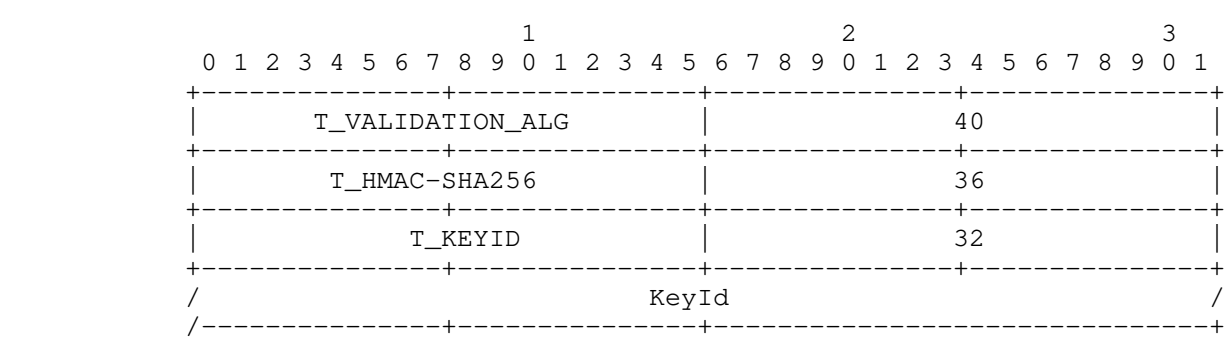


3.6.4.1.5. Validation Examples

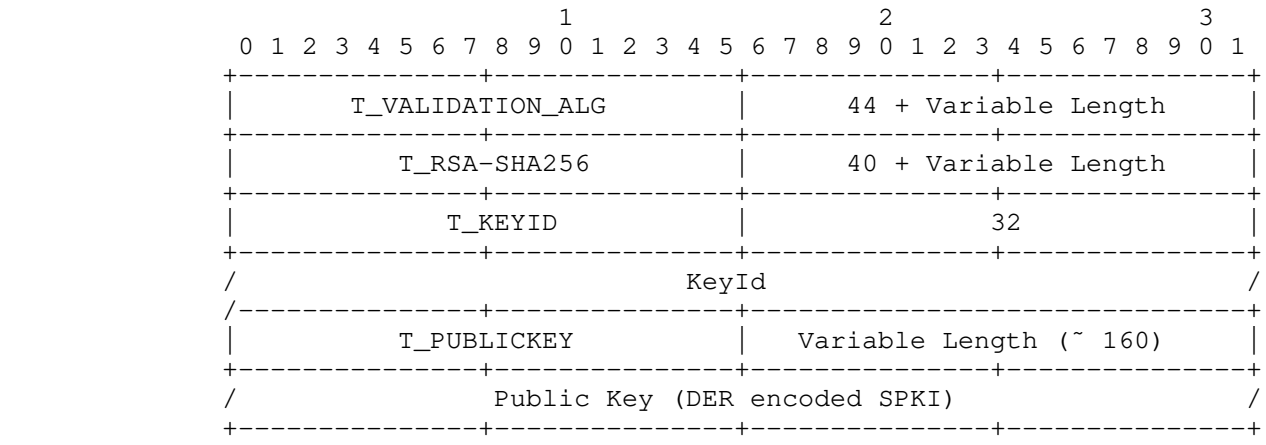
As an example of a MIC type validation, the encoding for CRC32C validation would be:



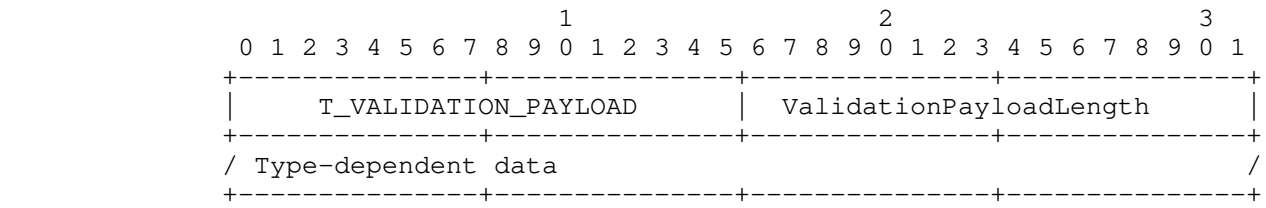
As an example of a MAC type validation, the encoding for an HMAC using a SHA256 hash would be:



As an example of a Signature type validation, the encoding for an RSA public key signing using a SHA256 digest and Public Key would be:



3.6.4.2. Validation Payload



The ValidationPayload contains the validation output, such as the CRC32C code or the RSA signature.

4. IANA Considerations

This section details each kind of protocol value that can be registered. Each type registry can be updated by incrementally expanding the type space, i.e., by allocating and reserving new types. As per [RFC5226] this section details the creation of the "CCNx Registry" and several sub-registries.

Property	Value
Name	CCNx Registry
Abbrev	CCNx

Registry Creation

4.1. Packet Type Registry

The following packet types should be allocated. A PacketType MUST be 1 byte. New packet types are allocated via "RFC Required" action.

Property	Value
Name	Packet Type Registry
Parent	CCNx Registry
Review process	RFC Required
Syntax	1 octet

Registry Creation

Type	Name	Reference
%x00	PT_INTEREST	Fixed Header Types (Section 3.2)
%x01	PT_CONTENT	Fixed Header Types (Section 3.2)
%x02	PT_RETURN	Fixed Header Types (Section 3.2)

Packet Type Namespace

4.2. Interest Return Code Registry

The following InterestReturn code types should be allocated.

Property	Value
Name	Interest Return Code
Parent	CCNx Registry
Review process	Specification Required
Syntax	1 octet

Registry Creation

Type	Name	Reference
%x00	Reserved	
%x01	T_RETURN_NO_ROUTE	Fixed Header Types (Section 3.2.3.3)
%x02	T_RETURN_LIMIT_EXCEEDED	Fixed Header Types (Section 3.2.3.3)
%x03	T_RETURN_NO_RESOURCES	Fixed Header Types (Section 3.2.3.3)
%x04	T_RETURN_PATH_ERROR	Fixed Header Types (Section 3.2.3.3)
%x05	T_RETURN_PROHIBITED	Fixed Header Types (Section 3.2.3.3)
%x06	T_RETURN_CONGESTED	Fixed Header Types (Section 3.2.3.3)
%x07	T_RETURN_MTU_TOO_LARGE	Fixed Header Types (Section 3.2.3.3)
%x08	T_RETURN_UNSUPPORTED_HASH_RESTRICTION	Fixed Header Types (Section 3.2.3.3)
%x09	T_RETURN_MALFORMED_INTEREST	Fixed Header Types (Section 3.2.3.3)

Interest Return Type Namespace

4.3. Hop-by-Hop Type Registry

The following hop-by-hop types should be allocated.

Property	Value
Name	Hop-by-Hop Type Registry
Parent	CCNx Registry
Review process	RFC Required
Syntax	2 octet TLV type

Registry Creation

Type	Name	Reference
%x0000	Reserved	
%x0001	T_INTLIFE	Hop-by-hop TLV headers (Section 3.4)
%x0002	T_CACHETIME	Hop-by-hop TLV headers (Section 3.4)
%x0003	T_MSGHASH	Hop-by-hop TLV headers (Section 3.4)
%x0004 - %x0007	Reserved	
%x0FFE	T_PAD	Pad (Section 3.3.1)
%x0FFF	T_ORG	Organization-Specific TLVs (Section 3.3.2)
%x1000-%x1FFF	Reserved	Experimental Use (Section 3)

Hop-by-Hop Type Namespace

4.4. Top-Level Type Registry

The following top-level types should be allocated.

Property	Value
Name	Top-Level Type Registry
Parent	CCNx Registry
Review process	RFC Required
Syntax	2 octet TLV type

Registry Creation

Type	Name	Reference
%x0000	Reserved	
%x0001	T_INTEREST	Top-Level Types (Section 3.5)
%x0002	T_OBJECT	Top-Level Types (Section 3.5)
%x0003	T_VALIDATION_ALG	Top-Level Types (Section 3.5)
%x0004	T_VALIDATION_PAYLOAD	Top-Level Types (Section 3.5)

Top-Level Type Namespace

4.5. Name Segment Type Registry

The following name segment types should be allocated.

Property	Value
Name	Name Segment Type Registry
Parent	CCNx Registry
Review process	Specification Required
Syntax	2 octet TLV type

Registry Creation

Type	Name	Reference
%x0000	Reserved	
%x0001	T_NAMESEGMENT	Name (Section 3.6.1)
%x0002	T_IPID	Name (Section 3.6.1)
%x0010 - %x0013	Reserved	Used in other drafts
%x0FFF	T_ORG	Organization-Specific TLVs (Section 3.3.2)
%x1000 - %x1FFF	T_APP:00 - T_APP:4096	Application Components (Section 3.6.1)

Name Segment Type Namespace

4.6. Message Type Registry

The following CCNx message segment types should be allocated.

Property	Value
Name	Message Type Registry
Parent	CCNx Registry
Review process	RFC Required
Syntax	2 octet TLV type

Registry Creation

Type	Name	Reference
%x0000	T_NAME	Message Types (Section 3.6)
%x0001	T_PAYLOAD	Message Types (Section 3.6)
%x0002	T_KEYIDRESTR	Message Types (Section 3.6)
%x0003	T_OBJHASHRESTR	Message Types (Section 3.6)
%x0005	T_PAYLDTYPE	Content Object Message Types (Section 3.6.2.2)
%x0006	T_EXPIRY	Content Object Message Types (Section 3.6.2.2)
%x0007 - %x000C	Reserved	Used in other RFC drafts
%x0FFE	T_PAD	Pad (Section 3.3.1)
%x0FFF	T_ORG	Organization-Specific TLVs (Section 3.3.2)
%x1000-%x1FFF	Reserved	Experimental Use (Section 3)

CCNx Message Type Namespace

4.7. Payload Type Registry

The following payload types should be allocated.

Property	Value
Name	PayloadType Registry
Parent	CCNx Registry
Review process	Specification Required
Syntax	Variable length unsigned integer

Registry Creation

Type	Name	Reference
%x00	T_PAYLOADTYPE_DATA	Payload Types (Section 3.6.2.2.1)
%x01	T_PAYLOADTYPE_KEY	Payload Types (Section 3.6.2.2.1)
%x02	T_PAYLOADTYPE_LINK	Payload Types (Section 3.6.2.2.1)

Payload Type Namespace

4.8. Validation Algorithm Type Registry

The following validation algorithm types should be allocated. Note: registration requires public specification of the algorithm.

Property	Value
Name	Validation Algorithm Type Registry
Parent	CCNx Registry
Review process	Specification Required
Syntax	2 octet TLV type

Registry Creation

Type	Name	Reference
%x0000	Reserved	
%x0001	Unassigned	
%x0002	T_CRC32C	Validation Algorithm (Section 3.6.4.1)
%x0003	Unassigned	
%x0004	T_HMAC-SHA256	Validation Algorithm (Section 3.6.4.1)
%x0005	T_RSA-SHA256	Validation Algorithm (Section 3.6.4.1)
%x0006	EC-SECP-256K1	Validation Algorithm (Section 3.6.4.1)
%x0007	EC-SECP-384R1	Validation Algorithm (Section 3.6.4.1)
%x0FFE	T_PAD	Pad (Section 3.3.1)
%x0FFF	T_ORG	Organization-Specific TLVs (Section 3.3.2)
%x1000-%x1FFF	Reserved	Experimental Use (Section 3)

Validation Algorithm Type Namespace

4.9. Validation Dependent Data Type Registry

The following validation dependent data types should be allocated.

Property	Value
Name	Validation Dependent Data Type Registry
Parent	CCNx Registry
Review process	RFC Required
Syntax	2 octet TLV type

Registry Creation

Type	Name	Reference
%x0000	Reserved	
%x0001 - %x0008	Unassigned	
%x0009	T_KEYID	Validation Dependent Data (Section 3.6.4.1.4)
%x000A	T_PUBLICKEYLOC	Validation Dependent Data (Section 3.6.4.1.4)
%x000B	T_PUBLICKEY	Validation Dependent Data (Section 3.6.4.1.4)
%x000C	T_CERT	Validation Dependent Data (Section 3.6.4.1.4)
%x000D	T_LINK	Validation Dependent Data (Section 3.6.4.1.4)
%x000E	T_KEYLINK	Validation Dependent Data (Section 3.6.4.1.4)
%x000F	T_SIGTIME	Validation Dependent Data (Section 3.6.4.1.4)
%x0FFF	T_ORG	Organization-Specific TLVs (Section 3.3.2)
%x1000-%x1FFF	Reserved	Experimental Use (Section 3)

Validation Dependent Data Type Namespace

4.10. Hash Function Type Registry

The following CCNx hash function types should be allocated. Note: registration requires public specification of the algorithm.

Property	Value
Name	Hash Function Type Registry
Parent	CCNx Registry
Review process	Specification Required
Syntax	2 octet TLV type

Registry Creation

Type	Name	Reference
%x0000	Reserved	
%x0001	T_SHA-256	Hash Format (Section 3.3.3)
%x0002	T_SHA-512	Hash Format (Section 3.3.3)
%x0FFF	T_ORG	Organization-Specific TLVs (Section 3.3.2)
%x1000-%x1FFF	Reserved	Experimental Use (Section 3)

CCNx Hash Function Type Namespace

5. Security Considerations

The CCNx protocol is a layer 3 network protocol, which may also operate as an overlay using other transports, such as UDP or other tunnels. It includes intrinsic support for message authentication via a signature (e.g. RSA or elliptic curve) or message authentication code (e.g. HMAC). In lieu of an authenticator, it may instead use a message integrity check (e.g. SHA or CRC). CCNx does not specify an encryption envelope, that function is left to a high-layer protocol (e.g. [esic]).

The CCNx message format includes the ability to attach MICs (e.g. SHA-256 or CRC), MACs (e.g. HMAC), and Signatures (e.g. RSA or ECDSA) to all packet types. This does not mean that it is a good idea to use an arbitrary ValidationAlgorithm, nor to include computationally expensive algorithms in Interest packets, as that could lead to computational DoS attacks. Applications should use an

explicit protocol to guide their use of packet signatures. As a general guideline, an application might use a MIC on an Interest to detect unintentionally corrupted packets. If one wishes to secure an Interest, one should consider using an encrypted wrapper and a protocol that prevents replay attacks, especially if the Interest is being used as an actuator. Simply using an authentication code or signature does not make an Interests secure. There are several examples in the literature on how to secure ICN-style messaging [mobile] [ace].

As a layer 3 protocol, this document does not describe how one arrives at keys or how one trusts keys. The CCNx content object may include a public key embedded in the object or may use the PublicKeyLocator field to point to a public key (or public key certificate) that authenticates the message. One key exchange specification is CCNxKE [ccnxke] [mobile], which is similar to the TLS 1.3 key exchange except it is over the CCNx layer 3 messages. Trust is beyond the scope of a layer-3 protocol protocol and left to applications or application frameworks.

The combination of an ephemeral key exchange (e.g. CCNxKE [ccnxke]) and an encapsulating encryption (e.g. [esic]) provides the equivalent of a TLS tunnel. Intermediate nodes may forward the Interests and Content Objects, but have no visibility inside. It also completely hides the internal names in those used by the encryption layer. This type of tunneling encryption is useful for content that has little or no cache-ability as it can only be used by someone with the ephemeral key. Short term caching may help with lossy links or mobility, but long term caching is usually not of interest.

Broadcast encryption or proxy re-encryption may be useful for content with multiple uses over time or many consumers. There is currently no recommendation for this form of encryption.

The specific encoding of messages will have security implications. This document uses a type-length-value (TLV) encoding. We chose to compromise between extensibility and unambiguous encodings of types and lengths. Some TLVs use variable length T and variable length L fields to accomodate a wide gamut of values while trying to be byte-efficient. Our TLV encoding uses a fixed length 2-byte T and 2-byte L. Using a fixed-length T and L field solves two problems. The first is aliases. If one is able to encode the same value, such as 0x2 and 0x02, in different byte lengths then one must decide if they mean the same thing, if they are different, or if one is illegal. If they are different, then one must always compare on the buffers not the integer equivalents. If one is illegal, then one must validate the TLV encoding -- every field of every packet at every hop. If they are the same, then one has the second problem: how to specify

packet filters. For example, if a name has 6 name components, then there are 7 T's and 7 L's, each of which might have up to 4 representations of the same value. That would be 14 fields with 4 encodings each, or 1001 combinations. It also means that one cannot compare, for example, a name via a memory function as one needs to consider that any embedded T or L might have a different format.

The Interest Return message has no authenticator from the previous hop. Therefore, the payload of the Interest Return should only be used locally to match an Interest. A node should never forward that Interest payload as an Interest. It should also verify that it sent the Interest in the Interest Return to that node and not allow anyone to negate Interest messages.

Caching nodes must take caution when processing content objects. It is essential that the Content Store obey the rules outlined in [CCNSemantics] to avoid certain types of attacks. Unlike NDN, CCNx 1.0 has no mechanism to work around an undesired result from the network (there are no "excludes"), so if a cache becomes poisoned with bad content it might cause problems retrieving content. There are three types of access to content from a content store: unrestricted, signature restricted, and hash restricted. If an Interest has no restrictions, then the requester is not particular about what they get back, so any matching cached object is OK. In the hash restricted case, the requester is very specific about what they want and the content store (and every forward hop) can easily verify that the content matches the request. In the signature verified case (often used for initial manifest discovery), the requester only knows the KeyId that signed the content. It is this case that requires the closest attention in the content store to avoid amplifying bad data. The content store must only respond with a content object if it can verify the signature -- this means either the content object carries the public key inside it or the Interest carries the public key in addition to the KeyId. If that is not the case, then the content store should treat the Interest as a cache miss and let an endpoint respond.

A user-level cache could perform full signature verification by fetching a public key according to the PublicKeyLocator. That is not, however, a burden we wish to impose on the forwarder. A user-level cache could also rely on out-of-band attestation, such as the cache operator only inserting content that it knows has the correct signature.

The CCNx grammar allows for hash algorithm agility via the HashType. It specifies a short list of acceptable hash algorithms that should be implemented at each forwarder. Some hash values only apply to end systems, so updating the hash algorithm does not affect forwarders --

they would simply match the buffer that includes the type-length-hash buffer. Some fields, such as the ConObjHash, must be verified at each hop, so a forwarder (or related system) must know the hash algorithm and it could cause backward compatibility problems if the hash type is updated.

A CCNx name uses binary matching whereas a URI uses a case insensitive hostname. Some systems may also use case insensitive matching of the URI path to a resource. An implication of this is that human-entered CCNx names will likely have case or non-ASCII symbol mismatches unless one uses a consistent URI normalization to the CCNx name. It also means that an entity that registers a CCNx routable prefix, say ccnx:/example.com, would need separate registrations for simple variations like ccnx:/Example.com. Unless this is addressed in URI normalization and routing protocol conventions, there could be phishing attacks.

For a more general introduction to ICN-related security concerns and approaches, see [RFC7927] and [RFC7945]

6. References

6.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

6.2. Informative References

[ace] Shang, W., Yu, Y., Liang, T., Zhang, B., and L. Zhang, "NDN-ACE: Access control for constrained environments over named data networking", NDN Technical Report NDN-0036, 2015, <http://new.named-data.net/wp-content/uploads/2015/12/ndn-0036-1-ndn-ace.pdf>.

[CCNSemantics] Mosko, M., Solis, I., and C. Wood, "CCNx Semantics (Internet draft)", 2018, <https://www.ietf.org/id/draft-irtf-icnrg-ccnxsemantics-09.txt>.

[ccnxke] Mosko, M., Uzun, E., and C. Wood, "CCNx Key Exchange Protocol Version 1.0", 2017, <https://www.ietf.org/archive/id/draft-wood-icnrg-ccnxkeyexchange-02.txt>.

[CCNxURI] Mosko, M. and C. Wood, "The CCNx URI Scheme (Internet draft)", 2017, <http://tools.ietf.org/html/draft-mosko-icnrg-ccnxuri-02>.

[CCNxz] Mosko, M., "CCNxz TLV Header Compression Experimental Code", 2016-2018, <https://github.com/PARC/CCNxz>.

[compress] Mosko, M., "Header Compression for TLV-based Packets", 2016, <https://datatracker.ietf.org/meeting/interim-2016-icnrg-02/materials/slides-interim-2016-icnrg-2-7>.

[ECC] Certicom Research, "SEC 2: Recommended Elliptic Curve Domain Parameters", 2010, <http://www.secg.org/sec2-v2.pdf>.

[EpriseNumbers] IANA, "IANA Private Enterprise Numbers", 2015, <http://www.iana.org/assignments/enterprise-numbers/enterprise-numbers>.

[esic] Mosko, M. and C. Wood, "Encrypted Sessions In CCNx (ESIC)", 2017, <https://www.ietf.org/id/draft-wood-icnrg-esic-01.txt>.

[mobile] Mosko, M., Uzun, E., and C. Wood, "Mobile Sessions in Content-Centric Networks", IFIP Networking, 2017, <http://dl.ifip.org/db/conf/networking/networking2017/1570334964.pdf>.

[nnc] Jacobson, V., Smetters, D., Thornton, J., Plass, M., Briggs, N., and R. Braynard, "Networking Named Content", 2009, <http://dx.doi.org/10.1145/1658939.1658941>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 5226, DOI 10.17487/RFC5226, May 2008, <https://www.rfc-editor.org/info/rfc5226>.

[RFC7927] Kutscher, D., Eum, S., Pentikousis, K., Psaras, I., Corujo, D., Saucez, D., Schmidt, T., and M. Waehlisch, "Information-Centric Networking (ICN) Research Challenges", 2016, <https://trac.tools.ietf.org/html/rfc7927>.

[RFC7945] Pentikousis, K., Ohlman, B., Davies, E., Spirou, S., and G. Boggia, "Information-Centric Networking: Evaluation and Security Considerations", 2016, <https://trac.tools.ietf.org/html/rfc7945>.

Authors' Addresses

Marc Mosko
PARC, Inc.
Palo Alto, California 94304
USA

Phone: +01 650-812-4405
Email: marc.mosko@parc.com

Ignacio Solis
LinkedIn
Mountain View, California 94043
USA

Email: nsolis@linkedin.com

Christopher A. Wood
University of California Irvine
Irvine, California 92697
USA

Phone: +01 315-806-5939
Email: woodc1@uci.edu

ICNRG
Internet-Draft
Intended status: Experimental
Expires: July 28, 2019

M. Mosko
PARC, Inc.
I. Solis
LinkedIn
C. Wood
University of California Irvine

January 24, 2019

CCNx Semantics
draft-irtf-icnrg-ccnxsemantics-10

Abstract

This document describes the core concepts of the Content Centric Networking (CCNx) architecture and presents a network protocol based on two messages: Interests and Content Objects. It specifies the set of mandatory and optional fields within those messages and describes their behavior and interpretation. This architecture and protocol specification is independent of a specific wire encoding.

The protocol also uses a Control message called an InterestReturn, whereby one system can return an Interest message to the previous hop due to an error condition. This indicates to the previous hop that the current system will not respond to the Interest.

This document is a product of the Information Centric Networking research group (ICNRG).

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on July 28, 2019.

Table of Contents

1. Introduction

This document describes the principles of the CCNx architecture. It describes a network protocol that uses a hierarchical name to forward requests and to match responses to requests. It does not use endpoint addresses, such as Internet Protocol. Restrictions in a request can limit the response by the public key of the response's signer or the cryptographic hash of the response. Every CCNx forwarder along the path does the name matching and restriction checking. The CCNx protocol fits within the broader framework of Information Centric Networking (ICN) protocols [RFC7927]. This document concerns the semantics of the protocol and is not dependent on a specific wire format encoding. The CCNx Messages [CCNMessages] document describes a type-length-value (TLV) wire protocol encoding. This section introduces the main concepts of CCNx, which are further elaborated in the remainder of the document.

The CCNx protocol derives from the early ICN work by Jacobson et al. [nnc]. Jacobson's version of CCNx is known as the 0.x version ("CCNx 0.x") and the present work is known as the 1.0 version ("CCNx 1.0"). There are two active implementations of CCNx 1.0. The most complete implementation is Community ICN (CINC) [cicn], a Linux Foundation project hosted at fd.io. Another active implementation is CCN-lite [ccnlite], with support for IoT systems and the RIOT operating system. CCNx 0.x formed the basis of the Named Data Networking [ndn] (NDN) university project.

The current CCNx 1.0 specification diverges from CCNx 0.x in a few significant areas. The most pronounced behavioral difference between CCNx 0.x and CCNx 1.0 is that CCNx 1.0 has a simpler response processing behavior. In both versions, a forwarder uses a hierarchical longest prefix match of a request name against the forwarding information base (FIB) to send the request through the network to a system that can issue a response. A forwarder must then match a response's name to a request's name to determine the reverse

path and deliver the response to the requester. In CCNx 0.x, the Interest name may be a hierarchical prefix of the response name, which allows a form of layer 3 content discovery. In CCNx 1.0, a response's name must exactly equal a request's name. Content discovery is performed by a higher-layer protocol.

CCNx Selectors [selectors] is an example of using a higher-layer protocol on top of the CCNx 1.0 layer-3 to perform content discovery. The selector protocol uses a method similar to the original CCNx 0.x techniques without requiring partial name matching of a response to a request in the forwarder.

The document represents the consensus of the ICN RG. It is the first ICN protocol from the RG, created from the early CCNx protocol [nnc] with significant revision and input from the ICN community and RG members. The draft has received critical reading by several members of the ICN community and the RG. The authors and RG chairs approve of the contents. The document is sponsored under the IRTF and is not issued by the IETF and is not an IETF standard. This is an experimental protocol and may not be suitable for any specific application and the specification may change in the future.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

1.2. Architecture

We describe the architecture of the network in which CCNx operates and introduce certain terminology from [terminology]. The detailed behavior of each component and message grammars are in Section 2.

A producer (also called a publisher) is an endpoint that encapsualtes content in Content Objects for transport in the CCNx network. A producer has a public/private keypair and signs (directly or indirectly) the content objects. Usually, the producer's keyid (hash of the public key) is well-known or may be derived from the producer's namespace via standard means.

A producer operates within one or more namespaces. A namespace is a name prefix that is represented in the forwarding information base (FIB). This allows a request to reach the producer and fetch a response (if one exists).

The forwarding information base (FIB) is a table that tells a forwarder where to send a request. It may point to a local

application, a local cache or content store, or to a remote system. If there is no matching entry in the FIB, a forwarder cannot process a request. The detailed rules on name matching to the FIB are given in Section 2.4.4. An endpoint has a FIB, though it may be a simple default route. An intermediate system (i.e. a router) typically has a much larger FIB. A core CCNx forwarder, for example, would know all the global routes.

A consumer is an endpoint that requests a name. It is beyond the scope of this document to describe how a consumer learns of a name or publisher keyid -- higher layer protocols build on top of CCNx handle those tasks, such as search engines or lookup services or well known names. The consumer constructs a request, called an Interest, and forwards it via the endpoint's FIB. The consumer should get back either a response, called a Content Object, that matches the Interest or a control message, called an InterestReturn, that indicates the network cannot handle the request.

There are three ways to detect errors in Interest handling. An InterestReturn is a network control message that indicates a low-level error like no route or out of resources. If an Interest arrives at a producer, but the producer does not have the requested content, the producer should send an application-specific error message (e.g. a not found message). Finally, a consumer may not receive anything, in which case it should timeout and, depending on the application, retry the request or return an error to the application.

1.3. Protocol Overview

The goal of CCNx is to name content and retrieve the content from the network without binding it to a specific network endpoint. A routing system (specified separately) populates the forwarding information base (FIB) tables at each CCNx router with hierarchical name prefixes that point towards the content producers under that prefix. A request finds matching content along those paths, in which case a response carries the data, or if no match is found a control message indicates the failure. A request may further refine acceptable responses with a restriction on the response's signer and the cryptographic hash of the response. The details of these restrictions are described below.

The CCNx name is a hierarchical series of path segments. Each path segment has a type and zero or more bytes. Matching two names is done as a binary comparison of the type and value, segment by segment. The human-readable form is defined under a URI scheme "ccnx:" [CCNxURI], though the canonical encoding of a name is a series of (type, octet string) pairs. There is no requirement that

any path segment be human readable or UTF-8. The first few segments in a name will matched against the FIB and a routing protocol may put its own restrictions on the routable name components (e.g. a maximum length or character encoding rules). In principle, path segments and names have unbounded length, though in practice they are limited by the wire format encoding and practical considerations imposed by a routing protocol. Note that in CCNx path segments use binary comparison whereas in a URI the authority uses case-insensitive hostname (due to DNS).

The CCNx name, as used by the forwarder, is purposefully left as a general octet-encoded type and value without any requirements on human readability and character encoding. The reason for this is that we are concerned with how a forwarder processes names. We expect that applications, routing protocols, or other higher layers will apply their own conventions and restrictions on the allowed path segment types and path segment values.

CCNx is a request and response protocol to fetch chunks of data using a name. The integrity of each chunk may be directly asserted through a digital signature or Message Authentication Code (MAC), or, alternatively, indirectly via hash chains. Chunks may also carry weaker message integrity checks (MICs) or no integrity protection mechanism at all. Because provenance information is carried with each chunk (or larger indirectly protected block), we no longer need to rely on host identities, such as those derived from TLS certificates, to ascertain the chunk legitimacy. Data integrity is therefore a core feature of CCNx; it does not rely on the data transmission channel. There are several options for data confidentiality, discussed later.

This document only defines the general properties of CCNx names. In some isolated environments, CCNx users may be able to use any name they choose and either inject that name (or prefix) into a routing protocol or use other information foraging techniques. In the Internet environment, there will be policies around the formats of names and assignments of names to publishers, though those are not specified here.

The key concept of CCNx is that a subjective name is cryptographically bound to a fixed payload. These publisher-generated bindings can therefore be cryptographically verified. A named payload is thus the tuple {{Name, ExtraFields, Payload, ValidationAlgorithm}, ValidationPayload}, where all fields in the inner tuple are covered by the validation payload (e.g. signature). Consumers of this data can check the binding integrity by re-computing the same cryptographic hash and verifying the digital signature in ValidationPayload.

In addition to digital signatures (e.g. RSA), CCNx also supports message authentication codes (e.g. HMAC) and message integrity codes (e.g. SHA-256 or CRC). To maintain the cryptographic binding, there should be at least one object with a signature or authentication code, but not all objects require it. For example, a first object with a signature could refer to other objects via a hash chain, a Merkle tree, or a signed manifest. The later objects may not have any validation and rely purely on the references. The use of an integrity code (e.g. CRC) is intended for detecting accidental corruption in an Interest.

CCNx specifies a network protocol around Interests (request messages) and Content Objects (response messages) to move named payloads. An Interest includes the Name -- which identifies the desired response -- and optional matching restrictions. Restrictions limit the possible matching Content Objects. Two restrictions exist: KeyIdRestr and ContentObjectHashRestr. The first restriction on the KeyId limits responses to those signed with a ValidationAlgorithm KeyId field equal to the restriction. The second is the Content ObjectHash restriction, which limits the response to one where the cryptographic hash of the entire named payload is equal to the restriction.

The hierarchy of a CCNx Name is used for routing via the longest matching prefix in a Forwarder. The longest matching prefix is computed name segment by name segment in the hierarchical path name, where each name segment must be exactly equal to match. There is no requirement that the prefix be globally routable. Within a deployment any local routing may be used, even one that only uses a single flat (non-hierarchical) name segment.

Another concept of CCNx is that there should be flow balance between Interest messages and Content Object messages. At the network level, an Interest traveling along a single path should elicit no more than one Content Object response. If some node sends the Interest along more than one path, that node should consolidate the responses such that only one Content Object flows back towards the requester. If an Interest is sent broadcast or multicast on a multiple-access media, the sender should be prepared for multiple responses unless some other media-dependent mechanism like gossip suppression or leader election is used.

As an Interest travels the forward path following the Forwarding Information Base (FIB), it establishes state at each forwarder such that a Content Object response can trace its way back to the original requester(s) without the requester needing to include a routable return address. We use the notional Pending Interest Table (PIT) as

a method to store state that facilitates the return of a Content Object.

The notional PIT table stores the last hop of an Interest plus its Name and optional restrictions. This is the data required to match a Content Object to an Interest (see Section 9). When a Content Object arrives, it must be matched against the PIT to determine which entries it satisfies. For each such entry, at most one copy of the Content Object is sent to each listed last hop in the PIT entries.

An actual PIT table is not mandated by the specification. An implementation may use any technique that gives the same external behavior. There are, for example, research papers that use techniques like label switching in some parts of the network to reduce the per-node state incurred by the PIT table [dart]. Some implementations store the PIT state in the FIB, so there is not a second table.

If multiple Interests with the same {Name, KeyIdRestr, ContentObjectHashRestr} tuple arrive at a node before a Content Object matching the first Interest comes back, they are grouped in the same PIT entry and their last hops aggregated (see Section 2.4.2). Thus, one Content Object might satisfy multiple pending Interests in a PIT.

In CCNx, higher-layer protocols are often called "name-based protocols" because they operate on the CCNx Name. For example, a versioning protocol might append additional name segments to convey state about the version of payload. A content discovery protocol might append certain protocol-specific name segments to a prefix to discover content under that prefix. Many such protocols may exist and apply their own rules to Names. They may be layered with each protocol encapsulating (to the left) a higher layer's Name prefix.

This document also describes a control message called an InterestReturn. A network element may return an Interest message to a previous hop if there is an error processing the Interest. The returned Interest may be further processed at the previous hop or returned towards the Interest origin. When a node returns an Interest it indicates that the previous hop should not expect a response from that node for the Interest, i.e., there is no PIT entry left at the returning node for a Content Object to follow.

There are multiple ways to describe larger objects in CCNx. Aggregating layer-3 content objects in to larger objects is beyond the scope of this document. One proposed method, FLIC [flic], uses a manifest to enumerate the pieces of a larger object. Manifests are, themselves, Content Objects. Another option is to use a convention

in the Content Object name, as in the CCNx Chunking [chunking] protocol where a large object is broken in to small chunks and each chunk receives a special name component indicating its serial order.

At the semantic level, described in this document, we do not address fragmentation. One experimental fragmentation protocol, BeginEnd Fragments [befrags] uses a multipoint-PPP style technique for use over layer-2 interfaces with the CCNx Messages [CCNMessages] TLV wire forman specification.

With these concepts, the remainder of the document specifies the behavior of a forwarder in processing Interest, Content Object, and InterestReturn messages.

2. Protocol

CCNx is a request and response protocol. A request is called an Interest and a response is called a Content Object. CCNx also uses a 1-hop control message called InterestReturn. These are, as a group, called CCNx Messages.

2.1. Message Grammar

The CCNx message ABNF [RFC5234] grammar is shown in Figure 1. The grammar does not include any encoding delimiters, such as TLVs. Specific wire encodings are given in a separate document. If a Validation section exists, the Validation Algorithm covers from the Body (BodyName or BodyOptName) through the end of the ValidationAlg section. The InterestLifetime, CacheTime, and Return Code fields exist outside of the validation envelope and may be modified.

The various fields -- in alphabetical order -- are defined as:

- AbsTime: Absolute times are conveyed as the 64-bit UTC time in milliseconds since the epoch (standard POSIX time).

- CacheTime: The absolute time after which the publisher believes there is low value in caching the content object. This is a recommendation to caches (see Section 4).

- ConObjField: These are optional fields that may appear in a Content Object.

- ConObjHash: The value of the Content Object Hash, which is the SHA256-32 over the message from the beginning of the body to the end of the message. Note that this coverage area is different from the ValidationAlg. This value SHOULD NOT be trusted across domains (see Section 5).

- ExpiryTime: An absolute time after which the content object should be considered expired (see Section 4).

- Hash: Hash values carried in a Message carry a HashType to identify the algorithm used to generate the hash followed by the hash value. This form is to allow hash agility. Some fields may mandate a specific HashType.

- HopLimit: Interest messages may loop if there are loops in the forwarding plane. To eventually terminate loops, each Interest carries a HopLimit that is decremented after each hop and no longer forwarded when it reaches zero. See Section 2.4.

- InterestField: These are optional fields that may appear in an Interest message.

- KeyIdRestr: The KeyId Restriction. A Content Object must have a KeyId with the same value as the restriction.

- ContentObjectHashRestr: The Content Object Hash Restriction. A content object must hash to the same value as the restriction using the same HashType. The ContentObjectHashRestr MUST use SHA256-32.

- KeyId: An identifier for the key used in the ValidationAlg. For public key systems, this should be the SHA-256 hash of the public key. For symmetric key systems, it should be an identifier agreed upon by the parties.

- KeyLink: A Link (see Section 6) that names how to retrieve the key used to verify the ValidationPayload. A message SHOULD NOT have both a KeyLink and a PublicKey.

- Lifetime: The approximate time during which a requester is willing to wait for a response, usually measured in seconds. It is not strongly related to the network round trip time, though it must necessarily be larger.

- Name: A name is made up of a non-empty first segment followed by zero or more additional segments, which may be of 0 length. Path segments are opaque octet strings, and are thus case-sensitive if encoding UTF-8. An Interest MUST have a Name. A Content Object MAY have a Name (see Section 9). The segments of a name are said to be complete if its segments uniquely identify a single Content Object. A name is exact if its segments are complete. An Interest carrying a full name is one which specifies an exact name and the ContentObjectHashRestr of the corresponding Content Object.

- Payload: The message's data, as defined by PayloadType.

- PayloadType: The format of the Payload. If missing, assume DataType. DataType means the payload is opaque application bytes. KeyType means the payload is a DER-encoded public key. LinkType means it is one or more Links (see Section 6).

- PublicKey: Some applications may wish to embed the public key used to verify the signature within the message itself. The PublickKey is DER encoded. A message SHOULD NOT have both a KeyLink and a PublicKey.

- RelTime: A relative time, measured in milli-seconds.

- ReturnCode: States the reason an Interest message is being returned to the previous hop (see Section 10.2).

- SigTime: The absolute time (UTC milliseconds) when the signature was generated.

- Vendor: Vendor-specific opaque data. The Vendor data includes the IANA Private Enterprise Numbers [EpriseNumbers], followed by vendor-specific information. CCNx allows vendor-specific data in most locations of the grammar.

```
Message       := Interest / ContentObject / InterestReturn
Interest      := IntHdr BodyName [Validation]
IntHdr        := HopLimit [Lifetime] *Vendor
ContentObject := ConObjHdr BodyOptName [Validation]
ConObjHdr     := [CacheTime / ConObjHash] *Vendor
InterestReturn:= ReturnCode Interest
BodyName      := Name Common
BodyOptName   := [Name] Common
Common        := *Field [Payload]
Validation    := ValidationAlg ValidatonPayload

Name          := FirstSegment *Segment
FirstSegment  := 1* OCTET / Vendor
Segment       := 0* OCTET / Vendor

ValidationAlg := (RSA-SHA256 / HMAC-SHA256 / CRC32C) *Vendor
ValidatonPayload := 1* OCTET
RSA-SHA256    := KeyId [PublicKey] [SigTime] [KeyLink]
HMAC-SHA256   := KeyId [SigTime] [KeyLink]
CRC32C        := [SigTime]

AbsTime       := 8 OCTET ; 64-bit UTC msec since epoch
CacheTime     := AbsTime
```

```
ConObjField   := ExpiryTime / PayloadType
ConObjHash    := Hash ; The Content Object Hash
DataType      := "1"
ExpiryTime    := AbsTime
Field         := InterestField / ConObjField / Vendor
Hash          := HashType 1* OCTET
HashType      := SHA256-32 / SHA512-64 / SHA512-32
HopLimit      := OCTET
InterestField := KeyIdRestr / ContentObjectHashRestr
KeyId         := 1* OCTET ; key identifier
KeyIdRestr    := 1* OCTET
KeyLink       := Link
KeyType       := "2"
Lifetime      := RelTime
Link          := Name [KeyIdResr] [ContentObjectHashRestr]
LinkType      := "3"
ContentObjectHashRestr  := Hash
Payload       := *OCTET
PayloadType   := DataType / KeyType / LinkType
PublicKey     := ; DER-encoded public key
RelTime       := 1* OCTET ; msec
ReturnCode    := ; see Section 10.2
SigTime       := AbsTime
Vendor        := PEN 0* OCTET
PEN           := ; IANA Private Enterprise Number
```

Figure 1

2.2. Consumer Behavior

To request a piece of content for a given {Name, [KeyIdRest], [ContentObjectHashRestr]} tuple, a consumer creates an Interest message with those values. It MAY add a validation section, typically only a CRC32C. A consumer MAY put a Payload field in an Interest to send additional data to the producer beyond what is in the Name. The Name is used for routing and may be remembered at each hop in the notional PIT table to facilitate returning a content object; Storing large amounts of state in the Name could lead to high memory requirements. Because the Payload is not considered when forwarding an Interest or matching a Content Object to an Interest, a consumer SHOULD put an Interest Payload ID (see Section 3.2) as part of the name to allow a forwarder to match Interests to content objects and avoid aggregating Interests with different payloads. Similarly, if a consumer uses a MAC or a signature, it SHOULD also include a unique segment as part of the name to prevent the Interest from being aggregated with other Interests or satisfied by a Content Object that has no relation to the validation.

The consumer SHOULD specify an InterestLifetime, which is the length of time the consumer is willing to wait for a response. The InterestLifetime is an application-scale time, not a network round trip time (see Section 2.4.2). If not present, the InterestLifetime will use a default value (2 seconds).

The consumer SHOULD set the Interest HopLimit to a reasonable value or use the default 255. If the consumer knows the distances to the producer via routing, it SHOULD use that value.

A consumer hands off the Interest to its first forwarder, which will then forward the Interest over the network to a publisher (or replica) that may satisfy it based on the name (see Section 2.4).

Interest messages are unreliable. A consumer SHOULD run a transport protocol that will retry the Interest if it goes unanswered, up to the InterestLifetime. No transport protocol is specified in this document.

The network MAY send to the consumer an InterestReturn message that indicates the network cannot fulfill the Interest. The ReturnCode specifies the reason for the failure, such as no route or congestion. Depending on the ReturnCode, the consumer MAY retry the Interest or MAY return an error to the requesting application.

If the content was found and returned by the first forwarder, the consumer will receive a Content Object. The consumer SHOULD:

- Ensure the content object is properly formatted.
- Verify that the returned Name matches a pending request. If the request also had KeyIdRestr or ObjHashRest, it MUST also validate those properties.
- If the content object is signed, it SHOULD cryptographically verify the signature. If it does not have the corresponding key, it SHOULD fetch the key, such as from a key resolution service or via the KeyLink.
- If the signature has a SigTime, the consumer MAY use that in considering if the signature is valid. For example, if the consumer is asking for dynamically generated content, it should expect the SigTime to not be before the time the Interest was generated.
- If the content object is signed, it should assert the trustworthiness of the signing key to the namespace. Such an assertion is beyond the scope of this document, though one may use

traditional PKI methods, a trusted key resolution service, or methods like [trust].

- It MAY cache the content object for future use, up to the ExpiryTime if present.

- A consumer MAY accept a content object off the wire that is expired. It may happen that a packet expires while in flight, and there is no requirement that forwarders drop expired packets in flight. The only requirement is that content stores, caches, or producers MUST NOT respond with an expired content object.

2.3. Publisher Behavior

This document does not specify the method by which names populate a Forwarding Information Base (FIB) table at forwarders (see Section 2.4). A publisher is either configured with one or more name prefixes under which it may create content, or it chooses its name prefixes and informs the routing layer to advertise those prefixes.

When a publisher receives an Interest, it SHOULD:

- Verify that the Interest is part of the publishers namespace(s).

- If the Interest has a Validation section, verify the ValidationPayload. Usually an Interest will only have a CRC32C unless the publisher application specifically accommodates other validations. The publisher MAY choose to drop Interests that carry a Validation section if the publisher application does not expect those signatures as this could be a form of computational denial of service. If the signature requires a key that the publisher does not have, it is NOT RECOMMENDED that the publisher fetch the key over the network, unless it is part of the application's expected behavior.

- Retrieve or generate the requested content object and return it to the Interest's previous hop. If the requested content cannot be returned, the publisher SHOULD reply with an InterestReturn or a content object with application payload that says the content is not available; this content object should have a short ExpiryTime in the future or not be cacheable (i.e. an expiry time of 0).

2.4. Forwarder Behavior

A forwarder routes Interest messages based on a Forwarding Information Base (FIB), returns Content Objects that match Interests to the Interest's previous hop, and processes InterestReturn control messages. It may also keep a cache of Content Objects in the

notional Content Store table. This document does not specify the internal behavior of a forwarder -- only these and other external behaviors.

In this document, we will use two processing pipelines, one for Interests and one for Content Objects. Interest processing is made up of checking for duplicate Interests in the PIT (see Section 2.4.2), checking for a cached Content Object in the Content Store (see Section 2.4.3), and forwarding an Interest via the FIB. Content Store processing is made up of checking for matching Interests in the PIT and forwarding to those previous hops.

2.4.1. Interest HopLimit

Interest looping is not prevented in CCNx. An Interest traversing loops is eventually discarded using the hop-limit field of the Interest, which is decremented at each hop traversed by the Interest.

A loop may also terminate because the Interest is aggregated with it's previous PIT entry along the loop. In this case, the Content will be sent back along the loop and eventually return to a node that already forwarded the content, so it will likely not have a PIT entry any more. When the content reaches a node without a PIT entry, it will be discarded. It may be that a new Interest or another looped Interest will return to that same node, in which case the node will either return a cached response to make a new PIT entry, as below.

The HopLimit is the last resort method to stop Interest loops where a Content Object chases an Interest around a loop and where the intermediate nodes, for whatever reason, no longer have a PIT entry and do not cache the Content Object.

Every Interest MUST carry a HopLimit. An Interest received from a local application MAY have a 0 HopLimit, which restricts the Interest to other local sources.

When an Interest is received from another forwarder, the HopLimit MUST be positive, otherwise the forwarder will discard the Interest. A forwarder MUST decrement the HopLimit of an Interest by at least 1 before it is forwarded.

If the decremented HopLimit equals 0, the Interest MUST NOT be forwarded to another forwarder; it MAY be sent to a local publisher application or serviced from a local Content Store.

A RECOMMENDED HopLimit processing pipeline is below:

- If Interest received from a remote system:

    * If received HopLimit is 0, optionally send InterestReturn (HopLimit Exceeded), and discard Interest.

    * Otherwise, decrement the HopLimit by 1.

- Process as per Content Store and Aggregation rules.

- If the Interest will be forwarded:

    * If the (potentailly decremented) HopLimit is 0, restrict forwarding to the local system.

    * Otherwise, forward as desired to local or remote systems.

2.4.2. Interest Aggregation

Interest aggregation is when a forwarder receives an Interest message that could be satisfied by the response to another Interest message already forwarded by the node, so the forwarder suppresses forwarding the new Interest; it only records the additional previous hop so a Content Object sent in response to the first Interest will satisfy both Interests.

CCNx uses an Interest aggregation rule that assumes the InterestLifetime is akin to a subscription time and is not a network round trip time. Some previous aggregation rules assumed the lifetime was a round trip time, but this leads to problems of expiring an Interest before a response comes if the RTT is estimated too short or interfering with an ARQ scheme that wants to re-transmit an Interest but a prior interest over-estimated the RTT.

A forwarder MAY implement an Interest aggregation scheme. If it does not, then it will forward all Interest messages. This does not imply that multiple, possibly identical, Content Objects will come back. A forwarder MUST still satisfy all pending Interests, so one Content Object could satisfy multiple similar interests, even if the forwarded did not suppress duplicate Interest messages.

A RECOMMENDED Interest aggregation scheme is:

- Two Interests are considered 'similar' if they have the same Name, KeyIdRestr, and ContentObjectHashRestr.

- Let the notional value InterestExpiry (a local value at the forwarder) be equal to the receive time plus the InterestLifetime (or a platform-dependent default value if not present).

- An Interest record (PIT entry) is considered invalid if its InterestExpiry time is in the past.

- The first reception of an Interest MUST be forwarded.

- A second or later reception of an Interest similar to a valid pending Interest from the same previous hop MUST be forwarded. We consider these a retransmission requests.

- A second or later reception of an Interest similar to a valid pending Interest from a new previous hop MAY be aggregated (not forwarded). If this Interest has a larger HopLimit than the pending Interest, it MUST be forwarded.

- Aggregating an Interest MUST extend the InterestExpiry time of the Interest record. An implementation MAY keep a single InterestExpiry time for all previous hops or MAY keep the InterestExpiry time per previous hop. In the first case, the forwarder might send a Content Object down a path that is no longer waiting for it, in which case the previous hop (next hop of the Content Object) would drop it.

2.4.3. Content Store Behavior

The Content Store is a special cache that is an integral part of a CCNx forwarder. It is an optional component. It serves to repair lost packets and handle flash requests for popular content. It could be pre-populated or use opportunistic caching. Because the Content Store could serve to amplify an attack via cache poisoning, there are special rules about how a Content Store behaves.

1. A forwarder MAY implement a Content Store. If it does, the Content Store matches a Content Object to an Interest via the normal matching rules (see Section 9).

2. If an Interest has a KeyIdRestr, then the Content Store MUST NOT reply unless it knows the signature on the matching Content Object is correct. It may do this by external knowledge (i.e., in a managed network or system with pre-populated caches) or by having the public key and cryptographically verifying the signature. A Content Store is NOT REQURIED to verify signatures; if it does not, then it treats these cases like a cache miss.

3. If a Content Store chooses to verify signatures, then it MAY do so as follows. If the public key is provided in the Content Object itself (i.e., in the PublicKey field) or in the Interest, the Content Store MUST verify that the public key's SHA-256 hash is equal to the KeyId and that it verifies the signature. A

Content Store MAY verify the digital signature of a Content Object before it is cached, but it is not required to do so. A Content Store SHOULD NOT fetch keys over the network. If it cannot or has not yet verified the signature, it should treat the Interest as a cache miss.

4. If an Interest has an ContentObjectHashRestr, then the Content Store MUST NOT reply unless it knows the the matching Content Object has the correct hash. If it cannot verify the hash, then it should treat the Interest as a cache miss.

5. It must obey the Cache Control directives (see Section 4).

2.4.4. Interest Pipeline

1. Perform the HopLimit check (see Section 2.4.1).

2. Determine if the Interest can be aggregated, as per Section 2.4.2. If it can be, aggregate and do not forward the Interest.

3. If forwarding the Interest, check for a hit in the Content Store, as per Section 2.4.3. If a matching Content Object is found, return it to the Interest's previous hop. This injects the Content Store as per Section 2.4.5.

4. Lookup the Interest in the FIB. Longest prefix match (LPM) is performed name segment by name segment (not byte or bit). It SHOULD exclude the Interest's previous hop. If a match is found, forward the Interest. If no match is found or the forwarder choses to not forward due to a local condition (e.g., congestion), it SHOULD send an InterestReturn message, as per Section 10.

2.4.5. Content Object Pipeline

1. It is RECOMMENDED that a forwarder that receives a content object check that the Content Object came from an expected previous hop. An expected previous hop is one pointed to by the FIB or one recorded in the PIT as having had a matching Interest sent that way.

2. A Content Object MUST be matched to all pending Interests that satisfy the matching rules (see Section 9). Each satisfied pending Interest MUST then be removed from the set of pending Interests.

3. A forwarder SHOULD NOT send more then one copy of the received Content Object to the same Interest previous hop. It may happen, for example, that two Interest ask for the same Content Object in different ways (e.g., by name and by name an KeyId) and that they both come from the same previous hop. It is normal to send the same content object multiple times on the same interface, such as Ethernet, if it is going to different previous hops.

4. A Content Object SHOULD only be put in the Content Store if it satisfied an Interest (and passed rule #1 above). This is to reduce the chances of cache poisoning.

3. Names

A CCNx name is a composition of name segments. Each name segment carries a label identifying the purpose of the name segment, and a value. For example, some name segments are general names and some serve specific purposes, such as carrying version information or the sequencing of many chunks of a large object into smaller, signed Content Objects.

There are three different types of names in CCNx: prefix, exact, and full names. A prefix name is simply a name that does not uniquely identify a single Content Object, but rather a namespace or prefix of an existing Content Object name. An exact name is one which uniquely identifies the name of a Content Object. A full name is one which is exact and is accompanied by an explicit or implicit ConObjHash. The ConObjHash is explicit in an Interest and implicit in a Content Object.

Note that a forwarder does not need to know any semantics about a name. It only needs to be able to match a prefix to forward Interests and match an exact or full name to forward Content Objects. It is not sensitive to the name segment types.

The name segment labels specified in this document are given in the table below. Name Segment is a general name segment, typically occurring in the routable prefix and user-specified content name. Other segment types are for functional name components that imply a specific purpose.

Name	Description
Name Segment	A generic name segment that includes arbitrary octets.
Interest Payload ID	An octet string that identifies the payload carried in an Interest. As an example, the Payload ID might be a hash of the Interest Payload. This provides a way to differentiate between Interests based on the Payload solely through a Name Segment without having to include all the extra bytes of the payload itself.
Application Components	An application-specific payload in a name segment. An application may apply its own semantics to these components. A good practice is to identify the application in a Name segment prior to the application component segments.

Table 1: CCNx Name Segment Types

At the lowest level, a Forwarder does not need to understand the semantics of name segments; it need only identify name segment boundaries and be able to compare two name segments (both label and value) for equality. The Forwarder matches paths segment-by-segment against its forwarding table to determine a next hop.

3.1. Name Examples

This section uses the CCNx URI [CCNxURI] representation of CCNx names. Note that as per the message grammar, an Interest must have a Name with at least one name segment and that name segment must have at least 1 octet of value. A Content Object must have a similar name or no name at all. The FIB, on the other hand, could have 0-length names (a default route), or a first name segment with no value, or a regular name.

Name	Description
ccnx:/	A 0-length name, corresponds to a default route.
ccnx:/NAME=	A name with 1 segment of 0 length, distinct from ccnx:/.
ccnx:/NAME=foo/APP:0=bar	A 2-segment name, where the first segment is of type NAME and the second segment is of type APP:0.

Table 2: CCNx Name Examples

3.2. Interest Payload ID

An Interest may also have a Payload which carries state about the Interest but is not used to match a Content Object. If an Interest contains a payload, the Interest name should contain an Interest Payload ID (IPID). The IPID allows a PIT table entry to correctly multiplex Content Objects in response to a specific Interest with a specific payload ID. The IPID could be derived from a hash of the payload or could be a GUID or a nonce. An optional Metadata field defines the IPID field so other systems could verify the IPID, such as when it is derived from a hash of the payload. No system is required to verify the IPID.

4. Cache Control

CCNx supports two fields that affect cache control. These determine how a cache or Content Store handles a Content Object. They are not used in the fast path, but only to determine if a Content Object can be injected on to the fast path in response to an Interest.

The ExpiryTime is a field that exists within the signature envelope of a Validation Algorithm. It is the UTC time in milliseconds after which the Content Object is considered expired and MUST no longer be used to respond to an Interest from a cache. Stale content MAY be flushed from the cache.

The Recommended Cache Time (RCT) is a field that exists outside the signature envelope. It is the UTC time in milliseconds after which the publisher considers the Content Object to be of low value to cache. A cache SHOULD discard it after the RCT, though it MAY keep it and still respond with it. A cache MAY discard the content object

before the RCT time too; there is no contractual obligation to remember anything.

This formulation allows a producer to create a Content Object with a long ExpiryTime but short RCT and keep re-publishing the same, signed, Content Object over and over again by extending the RCT. This allows a form of "phone home" where the publisher wants to periodically see that the content is being used.

5. Content Object Hash

CCNx allows an Interest to restrict a response to a specific hash. The hash covers the Content Object message body and the validation sections, if present. Thus, if a Content Object is signed, its hash includes that signature value. The hash does not include the fixed or hop-by-hop headers of a Content Object. Because it is part of the matching rules (see Section 9), the hash is used at every hop.

There are two options for matching the content object hash restriction in an Interest. First, a forwarder could compute for itself the hash value and compare it to the restriction. This is an expensive operation. The second option is for a border device to compute the hash once and place the value in a header (ConObjHash) that is carried through the network. The second option, of course, removes any security properties from matching the hash, so SHOULD only be used within a trusted domain. The header SHOULD be removed when crossing a trust boundary.

6. Link

A Link is the tuple {Name, [KeyIdRestr], [ContentObjectHashRestr]}. The information in a Link comprises the fields of an Interest which would retrieve the Link target. A Content Object with PayloadType = "Link" is an object whose payload is one or more Links. This tuple may be used as a KeyLink to identify a specific object with the certificate wrapped key. It is RECOMMENDED to include at least one of KeyIdRestr or Content ObjectHashRestr. If neither restriction is present, then any Content Object with a matching name from any publisher could be returned.

7. Hashes

Several protocol fields use cryptographic hash functions, which must be secure against attack and collisions. Because these hash functions change over time, with better ones appearing and old ones falling victim to attacks, it is important that a CCNx protocol implementation supports hash agility.

In this document, we suggest certain hashes (e.g., SHA-256), but a specific implementation may use what it deems best. The normative CCNx Messages [CCNMessages] specification should be taken as the definition of acceptable hash functions and uses.

8. Validation

8.1. Validation Algorithm

The Validator consists of a ValidationAlgorithm that specifies how to verify the message and a ValidationPayload containing the validation output, e.g., the digital signature or MAC. The ValidationAlgorithm section defines the type of algorithm to use and includes any necessary additional information. The validation is calculated from the beginning of the CCNx Message through the end of the ValidationAlgorithm section. The ValidationPayload is the integrity value bytes, such as a MAC or signature.

Some Validators contain a KeyId, identifying the publisher authenticating the Content Object. If an Interest carries a KeyIdRestr, then that KeyIdRestr MUST exactly match the Content Object's KeyId.

Validation Algorithms fall into three categories: MICs, MACs, and Signatures. Validators using Message Integrity Code (MIC) algorithms do not need to provide any additional information; they may be computed and verified based only on the algorithm (e.g., CRC32C). MAC validators require the use of a KeyId identifying the secret key used by the authenticator. Because MACs are usually used between two parties that have already exchanged secret keys via a key exchange protocol, the KeyId may be any agreed-upon value to identify which key is used. Signature validators use public key cryptographic algorithms such as RSA, DSA, ECDSA. The KeyId field in the ValidationAlgorithm identifies the public key used to verify the signature. A signature may optionally include a KeyLocator, as described above, to bundle a Key or Certificate or KeyLink. MAC and Signature validators may also include a SignatureTime, as described above.

A PublicKeyLocator KeyLink points to a Content Object with a DER-encoded X509 certificate in the payload. In this case, the target KeyId must equal the first object's KeyId. The target KeyLocator must include the public key corresponding to the KeyId. That key must validate the target Signature. The payload is an X.509 certificate whose public key must match the target KeyLocator's key. It must be issued by a trusted authority, preferably specifying the valid namespace of the key in the distinguished name.

9. Interest to Content Object matching

A Content Object satisfies an Interest if and only if (a) the Content Object name, if present, exactly matches the Interest name, and (b) the ValidationAlgorithm KeyId of the Content Object exactly equals the Interest KeyIdRestr, if present, and (c) the computed Content ObjectHash exactly equals the Interest ContentObjectHashRestr, if present.

The matching rules are given by this predicate, which if it evaluates true means the Content Object matches the Interest. Ni = Name in Interest (may not be empty), Ki = KeyIdRestr in the interest (may be empty), Hi = ContentObjectHashRestr in Interest (may be empty). Likewise, No, Ko, Ho are those properties in the Content Object, where No and Ko may be empty; Ho always exists (it is an intrinsic property of the Content Object). For binary relations, we use & for AND and | for OR. We use E for the EXISTS (not empty) operator and ! for the NOT EXISTS operator.

As a special case, if the ContentObjectHashRestr in the Interest specifies an unsupported hash algorithm, then no Content Object can match the Interest so the system should drop the Interest and MAY send an InterestReturn to the previous hop. In this case, the predicate below will never get executed because the Interest is never forwarded. If the system is using the optional behavior of having a different system calculate the hash for it, then the system may assume all hash functions are supported and leave it to the other system to accept or reject the Interest.

```
(!No | (Ni=No)) & (!Ki | (Ki=Ko)) & (!Hi | (Hi=Ho)) & (E No | E Hi)
```

As one can see, there are two types of attributes one can match. The first term depends on the existence of the attribute in the Content Object while the next two terms depend on the existence of the attribute in the Interest. The last term is the "Nameless Object" restriction which states that if a Content Object does not have a Name, then it must match the Interest on at least the Hash restriction.

If a Content Object does not carry the Content ObjectHash as an expressed field, it must be calculated in network to match against. It is sufficient within an autonomous system to calculate a Content ObjectHash at a border router and carry it via trusted means within the autonomous system. If a Content Object ValidationAlgorithm does not have a KeyId then the Content Object cannot match an Interest with a KeyIdRestr.

10. Interest Return

This section describes the process whereby a network element may return an Interest message to a previous hop if there is an error processing the Interest. The returned Interest may be further processed at the previous hop or returned towards the Interest origin. When a node returns an Interest it indicates that the previous hop should not expect a response from that node for the Interest -- i.e., there is no PIT entry left at the returning node.

The returned message maintains compatibility with the existing TLV packet format (a fixed header, optional hop-by-hop headers, and the CCNx message body). The returned Interest packet is modified in only two ways:

- The PacketType is set to InterestReturn to indicate a Feedback message.
- The ReturnCode is set to the appropriate value to signal the reason for the return

The specific encodings of the Interest Return are specified in [CCNMessages].

A Forwarder is not required to send any Interest Return messages.

A Forwarder is not required to process any received Interest Return message. If a Forwarder does not process Interest Return messages, it SHOULD silently drop them.

The Interest Return message does not apply to a Content Object or any other message type.

An Interest Return message is a 1-hop message between peers. It is not propagated multiple hops via the FIB. An intermediate node that receives an InterestReturn may take corrective actions or may propagate its own InterestReturn to previous hops as indicated in the reverse path of a PIT entry.

10.1. Message Format

The Interest Return message looks exactly like the original Interest message with the exception of the two modifications mentioned above. The PacketType is set to indicate the message is an InterestReturn and the reserved byte in the Interest header is used as a Return Code. The numeric values for the PacketType and ReturnCodes are in [CCNMessages].

10.2. ReturnCode Types

This section defines the InterestReturn ReturnCode introduced in this RFC. The numeric values used in the packet are defined in [CCNMessages].

Name	Description
No Route (Section 10.3.1)	The returning Forwarder has no route to the Interest name.
HopLimit Exceeded (Section 10.3.2)	The HopLimit has decremented to 0 and need to forward the packet.
Interest MTU too large (Section 10.3.3)	The Interest's MTU does not conform to the required minimum and would require fragmentation.
No Resources (Section 10.3.4)	The node does not have the resources to process the Interest.
Path error (Section 10.3.5)	There was a transmission error when forwarding the Interest along a route (a transient error).
Prohibited (Section 10.3.6)	An administrative setting prohibits processing this Interest.
Congestion (Section 10.3.7)	The Interest was dropped due to congestion (a transient error).
Unsupported Content Object Hash Algorithm (Section 10.3.8)	The Interest was dropped because it requested a Content Object Hash Restriction using a hash algorithm that cannot be computed.
Malformed Interest (Section 10.3.9)	The Interest was dropped because it did not correctly parse.

Table 3: Interest Return Reason Codes

10.3. Interest Return Protocol

This section describes the Forwarder behavior for the various Reason codes for Interest Return. A Forwarder is not required to generate

any of the codes, but if it does, it MUST conform to this specification.

If a Forwarder receives an Interest Return, it SHOULD take these standard corrective actions. A forwarder is allowed to ignore Interest Return messages, in which case its PIT entry would go through normal timeout processes.

- Verify that the Interest Return came from a next-hop to which it actually sent the Interest.
- If a PIT entry for the corresponding Interest does not exist, the Forwarder should ignore the Interest Return.
- If a PIT entry for the corresponding Interest does exist, the Forwarder MAY do one of the following:
  - Try a different forwarding path, if one exists, and discard the Interest Return, or
  - Clear the PIT state and send an Interest Return along the reverse path.

If a forwarder tries alternate routes, it MUST ensure that it does not use same same path multiple times. For example, it could keep track of which next hops it has tried and not re-use them.

If a forwarder tries an alternate route, it may receive a second InterestReturn, possibly of a different type than the first InterestReturn. For example, node A sends an Interest to node B, which sends a No Route return. Node A then tries node C, which sends a Prohibited. Node A should choose what it thinks is the appropriate code to send back to its previous hop

If a forwarder tries an alternate route, it should decrement the Interest Lifetime to account for the time spent thus far processing the Interest.

10.3.1. No Route

If a Forwarder receives an Interest for which it has no route, or for which the only route is back towards the system that sent the Interest, the Forwarder SHOULD generate a "No Route" Interest Return message.

How a forwarder manages the FIB table when it receives a No Route message is implementation dependent. In general, receiving a No Route Interest Return should not cause a forwarder to remove a route.

The dynamic routing protocol that installed the route should correct the route or the administrator who created a static route should correct the configuration. A forwarder could suppress using that next hop for some period of time.

10.3.2. HopLimit Exceeded

A Forwarder MAY choose to send HopLimit Exceeded messages when it receives an Interest that must be forwarded off system and the HopLimit is 0.

10.3.3. Interest MTU Too Large

If a Forwarder receives an Interest whose MTU exceeds the prescribed minimum, it MAY send an "Interest MTU Too Large" message, or it may silently discard the Interest.

If a Forwarder receives an "Interest MTU Too Large" is SHOULD NOT try alternate paths. It SHOULD propagate the Interest Return to its previous hops.

10.3.4. No Resources

If a Forwarder receives an Interest and it cannot process the Interest due to lack of resources, it MAY send an InterestReturn. A lack of resources could be the PIT table is too large, or some other capacity limit.

10.3.5. Path Error

If a forwarder detects an error forwarding an Interest, such as over a reliable link, it MAY send a Path Error Interest Return indicating that it was not able to send or repair a forwarding error.

10.3.6. Prohibited

A forwarder may have administrative policies, such as access control lists, that prohibit receiving or forwarding an Interest. If a forwarder discards an Interest due to a policy, it MAY send a Prohibited InterestReturn to the previous hop. For example, if there is an ACL that says /parc/private can only come from interface e0, but the Forwarder receives one from e1, the Forwarder must have a way to return the Interest with an explanation.

10.3.7. Congestion

If a forwarder discards an Interest due to congestion, it MAY send a Congestion InterestReturn to the previous hop.

10.3.8. Unsupported Content Object Hash Algorithm

If a Content Object Hash Restriction specifies a hash algorithm the forwarder cannot verify, the Interest should not be accepted and the forwarder MAY send an InterestReturn to the previous hop.

10.3.9. Malformed Interest

If a forwarder detects a structural or syntactical error in an Interest, it SHOULD drop the interest and MAY send an InterestReturn to the previous hop. This does not imply that any router must validate the entire structure of an Interest.

11. IANA Considerations

This memo includes no request to IANA.

12. Security Considerations

The CCNx protocol is a layer 3 network protocol, which may also operate as an overlay using other transports, such as UDP or other tunnels. It includes intrinsic support for message authentication via a signature (e.g. RSA or elliptic curve) or message authentication code (e.g. HMAC). In lieu of an authenticator, it may instead use a message integrity check (e.g. SHA or CRC). CCNx does not specify an encryption envelope, that function is left to a high-layer protocol (e.g. [esic]).

The CCNx message format includes the ability to attach MICs (e.g. SHA-256 or CRC), MACs (e.g. HMAC), and Signatures (e.g. RSA or ECDSA) to all packet types. This does not mean that it is a good idea to use an arbitrary ValidationAlgorithm, nor to include computationally expensive algorithms in Interest packets, as that could lead to computational DoS attacks. Applications should use an explicit protocol to guide their use of packet signatures. As a general guideline, an application might use a MIC on an Interest to detect unintentionally corrupted packets. If one wishes to secure an Interest, one should consider using an encrypted wrapper and a protocol that prevents replay attacks, especially if the Interest is being used as an actuator. Simply using an authentication code or signature does not make an Interests secure. There are several examples in the literature on how to secure ICN-style messaging [mobile] [ace].

As a layer 3 protocol, this document does not describe how one arrives at keys or how one trusts keys. The CCNx content object may include a public key embedded in the object or may use the PublicKeyLocator field to point to a public key (or public key certificate) that authenticates the message. One key exchange specification is CCNxKE [ccnxke] [mobile], which is similar to the TLS 1.3 key exchange except it is over the CCNx layer 3 messages. Trust is beyond the scope of a layer-3 protocol protocol and left to applications or application frameworks.

The combination of an ephemeral key exchange (e.g. CCNxKE [ccnxke]) and an encapsulating encryption (e.g. [esic]) provides the equivalent of a TLS tunnel. Intermediate nodes may forward the Interests and Content Objects, but have no visibility inside. It also completely hides the internal names in those used by the encryption layer. This type of tunneling encryption is useful for content that has little or no cache-ability as it can only be used by someone with the ephemeral key. Short term caching may help with lossy links or mobility, but long term caching is usually not of interest.

Broadcast encryption or proxy re-encryption may be useful for content with multiple uses over time or many consumers. There is currently no recommendation for this form of encryption.

The specific encoding of messages will have security implications. [CCNMessages] uses a type-length-value (TLV) encoding. We chose to compromise between extensibility and unambiguous encodings of types and lengths. Some TLVs use variable length T and variable length L fields to accomodate a wide gamut of values while trying to be byte-efficient. Our TLV encoding uses a fixed length 2-byte T and 2-byte L. Using a fixed-length T and L field solves two problems. The first is aliases. If one is able to encode the same value, such as 0x2 and 0x02, in different byte lengths then one must decide if they mean the same thing, if they are different, or if one is illegal. If they are different, then one must always compare on the buffers not the integer equivalents. If one is illegal, then one must validate the TLV encoding -- every field of every packet at every hop. If they are the same, then one has the second problem: how to specify packet filters. For example, if a name has 6 name components, then there are 7 T's and 7 L's, each of which might have up to 4 representations of the same value. That would be 14 fields with 4 encodings each, or 1001 combinations. It also means that one cannot compare, for example, a name via a memory function as one needs to consider that any embedded T or L might have a different format.

The Interest Return message has no authenticator from the previous hop. Therefore, the payload of the Interest Return should only be used locally to match an Interest. A node should never forward that

Interest payload as an Interest. It should also verify that it sent the Interest in the Interest Return to that node and not allow anyone to negate Interest messages.

Caching nodes must take caution when processing content objects. It is essential that the Content Store obey the rules outlined in Section 2.4.3 to avoid certain types of attacks. Unlike NDN, CCNx 1.0 has no mechanism to work around an undesired result from the network (there are no "excludes"), so if a cache becomes poisoned with bad content it might cause problems retrieving content. There are three types of access to content from a content store: unrestricted, signature restricted, and hash restricted. If an Interest has no restrictions, then the requester is not particular about what they get back, so any matching cached object is OK. In the hash restricted case, the requester is very specific about what they want and the content store (and every forward hop) can easily verify that the content matches the request. In the signature verified case (often used for initial manifest discovery), the requester only knows the KeyId that signed the content. It is this case that requires the closest attention in the content store to avoid amplifying bad data. The content store must only respond with a content object if it can verify the signature -- this means either the content object carries the public key inside it or the Interest carries the public key in addition to the KeyId. If that is not the case, then the content store should treat the Interest as a cache miss and let an endpoint respond.

A user-level cache could perform full signature verification by fetching a public key according to the PublicKeyLocator. That is not, however, a burden we wish to impose on the forwarder. A user-level cache could also rely on out-of-band attestation, such as the cache operator only inserting content that it knows has the correct signature.

The CCNx grammar allows for hash algorithm agility via the HashType. It specifies a short list of acceptable hash algorithms that should be implemented at each forwarder. Some hash values only apply to end systems, so updating the hash algorithm does not affect forwarders -- they would simply match the buffer that includes the type-length-hash buffer. Some fields, such as the ConObjHash, must be verified at each hop, so a forwarder (or related system) must know the hash algorithm and it could cause backward compatibility problems if the hash type is updated. [CCNMessages] is the authoritative source for per-field allowed hash types in that encoding.

A CCNx name uses binary matching whereas a URI uses a case insensitive hostname. Some systems may also use case insensitive matching of the URI path to a resource. An implication of this is

that human-entered CCNx names will likely have case or non-ASCII symbol mismatches unless one uses a consistent URI normalization to the CCNx name. It also means that an entity that registers a CCNx routable prefix, say ccnx:/example.com, would need separate registrations for simple variations like ccnx:/Example.com. Unless this is addressed in URI normalization and routing protocol conventions, there could be phishing attacks.

For a more general introduction to ICN-related security concerns and approaches, see [RFC7927] and [RFC7945]

13. References

13.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <https://www.rfc-editor.org/info/rfc2119>.

13.2. Informative References

[ace] Shang, W., Yu, Y., Liang, T., Zhang, B., and L. Zhang, "NDN-ACE: Access control for constrained environments over named data networking", NDN Technical Report NDN-0036, 2015, <http://new.named-data.net/wp-content/uploads/2015/12/ndn-0036-1-ndn-ace.pdf>.

[befrags] Mosko, M. and C. Tschudin, "ICN "Begin-End" Hop by Hop Fragmentation", 2017, <https://www.ietf.org/archive/id/draft-mosko-icnrg-beginendfragment-02.txt>.

[ccnlite] Tschudin, C., et al., University of Basel, "CCN-Lite V2", 2011-2018, <http://www.ccn-lite.net/>.

[CCNMessages] Mosko, M., Solis, I., and C. Wood, "CCNx Messages in TLV Format (Internet draft)", 2018, <https://www.ietf.org/id/draft-irtf-icnrg-ccnxmessages-07.txt>.

[ccnxke] Mosko, M., Uzun, E., and C. Wood, "CCNx Key Exchange Protocol Version 1.0", 2017, <https://www.ietf.org/archive/id/draft-wood-icnrg-ccnxkeyexchange-02.txt>.

[CCNxURI] Mosko, M. and C. Wood, "The CCNx URI Scheme (Internet draft)", 2017, <http://tools.ietf.org/html/draft-mosko-icnrg-ccnxuri-02>.

[chunking]
Mosko, M., "CCNx Content Object Chunking", 2016, <https://www.ietf.org/archive/id/draft-mosko-icnrg-ccnxchunking-02.txt>.

[cicn] Muscariello, L., et al., Cisco Systems, "Community ICN (CICN)", 2017-2018, <https://wiki.fd.io/view/Cicn>.

[dart] Garcia-Luna-Aceves, J. and M. Mirzazad-Barijough, "A Light-Weight Forwarding Plane for Content-Centric Networks", 2016, <https://arxiv.org/pdf/1603.06044.pdf>.

[EpriseNumbers]
IANA, "IANA Private Enterprise Numbers", 2015, <http://www.iana.org/assignments/enterprise-numbers/enterprise-numbers>.

[esic] Mosko, M. and C. Wood, "Encrypted Sessions In CCNx (ESIC)", 2017, <https://www.ietf.org/id/draft-wood-icnrg-esic-01.txt>.

[flic] Tschudin, C. and C. Wood, "File-Like ICN Collection (FLIC)", 2017, <https://www.ietf.org/archive/id/draft-tschudin-icnrg-flic-03.txt>.

[mobile] Mosko, M., Uzun, E., and C. Wood, "Mobile Sessions in Content-Centric Networks", IFIP Networking, 2017, <http://dl.ifip.org/db/conf/networking/networking2017/1570334964.pdf>.

[ndn] UCLA, "Named Data Networking", 2007, <http://www.named-data.net>.

[nnc] Jacobson, V., Smetters, D., Thornton, J., Plass, M., Briggs, N., and R. Braynard, "Networking Named Content", 2009, <http://dx.doi.org/10.1145/1658939.1658941>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <https://www.rfc-editor.org/info/rfc5234>.

[RFC7927] Kutscher, D., Eum, S., Pentikousis, K., Psaras, I., Corujo, D., Saucez, D., Schmidt, T., and M. Waehlisch, "Information-Centric Networking (ICN) Research Challenges", 2016, <https://trac.tools.ietf.org/html/rfc7927>.

[RFC7945] Pentikousis, K., Ohlman, B., Davies, E., Spirou, S., and G. Boggia, "Information-Centric Networking: Evaluation and Security Considerations", 2016, <https://trac.tools.ietf.org/html/rfc7945>.

[selectors] Mosko, M., "CCNx Selector Based Discovery", 2017, <https://raw.githubusercontent.com/mmosko/ccnx-protocol-rfc/master/docs/build/draft-mosko-icnrg-selectors-01.txt>.

[terminology] Wissingh, B., Wood, C., Afanasyev, A., Zhang, L., Oran, D., and C. Tschudin, "Information-Centric Networking (ICN): CCN and NDN Terminology", 2017, <https://www.ietf.org/id/draft-irtf-icnrg-terminology-00.txt>.

[trust] Tschudin, C., Uzun, E., and C. Wood, "Trust in Information-Centric Networking: From Theory to Practice", 2016, <https://doi.org/10.1109/ICCCN.2016.7568589>.

Authors' Addresses

Marc Mosko
PARC, Inc.
Palo Alto, California 94304
USA

Phone: +01 650-812-4405
Email: marc.mosko@parc.com

Ignacio Solis
LinkedIn
Mountain View, California 94043
USA

Email: nsolis@linkedin.com

Christopher A. Wood
University of California Irvine
Irvine, California 92697
USA

Phone: +01 315-806-5939
Email: woodc1@uci.edu

ICNRG
Internet-Draft
Intended status: Informational
Expires: March 10, 2020

B. Wissingh
TNO
C. Wood
University of California Irvine
A. Afanasyev
Florida International University
L. Zhang
UCLA
D. Oran
Network Systems Research & Design
C. Tschudin
University of Basel

September 7, 2019

Information-Centric Networking (ICN): CCNx and NDN Terminology
draft-irtf-icnrg-terminology-05

Abstract

Information Centric Networking (ICN) is a novel paradigm where network communications are accomplished by requesting named content, instead of sending packets to destination addresses. Named Data Networking (NDN) and Content-Centric Networking (CCNx) are two prominent ICN architectures. This document provides an overview of the terminology and definitions that have been used in describing concepts in these two implementations of ICN. While there are other ICN architectures, they are not part of the NDN and CCNx concepts and as such are out of scope for this document. This document is a product of the Information-Centric Networking Research Group (ICNRG).

Table of Contents

1. Introduction

Information-centric networking (ICN) is an architecture to evolve the Internet infrastructure from the existing host-centric design to a data-centric architecture, where accessing data by name becomes the essential network primitive. The goal is to let applications refer to data independently of their location or means of transportation, which enables native multicast delivery, ubiquitous in-network caching and replication of data objects.

As the work on this topic continues to evolve, many new terms are emerging. The goal of this document is to collect the key terms with a corresponding definition as they are used in the CCNx and NDN projects. Other ICN projects such as NetInf, or MobilityFirst are not covered and will be the subject of other documents.

To help provide context for the individual defined terms, in this draft we first sketch the bigger picture of an ICN network by introducing the basic concepts and identifying the major components of the architecture in Section 2, after which, in Section 3, ICN related terms are listed by different categories.

This document represents the consensus of the Information-Centric Networking Research Group (ICNRG). It has been reviewed extensively by the Research Group (RG) members active in the specific areas of work covered by the document. It is not an IETF product and is not intended for standardization in the IETF.

2. A Sketch of the Big Picture of ICN

In networking terms, an ICN is a delivery infrastructure for named data. For other complementing views see Section 4.

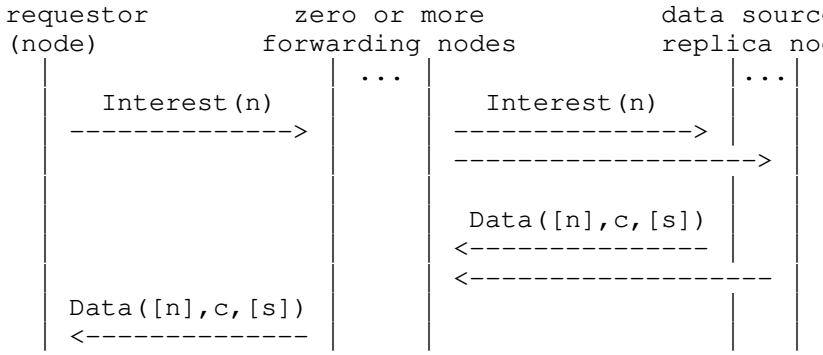


Figure 1: Request-Reply Protocol of ICN networking. Legend: n=name, c=content, s=signature.

The following list describes the basic ICN concepts needed to discuss the implementation of this service abstraction.

Request-Reply Protocol (Interest and Data Packet):

An ICN's lookup service is implemented by defining two types of network packet formats: Interest packets that request content by name, and Data packets that carry the requested content. The returned Data packet must match the request's parameters (e.g.,

having a partially or fully matching name). If the request is ambiguous and several Data packets would satisfy the request, the ICN network returns only one matching Data packet (flow balance between Interest and Data packets over individual links).

Packet and Content Names

Without an irrefutable binding between the name of a Data packet and its content, Data packet names would be useless for fetching specific content. In ICN, verification of a Data packet's name-to-content binding is achieved through cryptographic means, either by (1) a cryptographic signature that explicitly binds an application-chosen name to a Data packet's content, or (2) relying on an implicit name (cryptographic hash of the Data packet with or without application-chosen name) that the data consumer obtained through other means.

Data Authenticity and Encryption:

Any data consumer and network element can validate the authenticity of a Data packet by verifying its cryptographic name-to-content binding. In contrast, whether a Data packet's content (payload) itself is encrypted or not is irrelevant to the ICN network. The use and management of content encryption keys is an application-layer concern.

Trust:

Data authenticity is distinct from data trustworthiness, though the two concepts are related. A packet is authentic if it has a valid name-to-content binding. A packet is trustworthy, i.e., it comes from a reputable or trusted origin, if this binding is valid in the context of a trust model.

Segmenting and Versioning:

An ICN network will be engineered for some packet size limit. As application-level data objects will often be considerably larger, objects must be segmented into multiple Data packets. The names for these Data packets can, for example, be constructed by choosing one application-level object name to which a different suffix is added for each segment. The same method can be used to handle different versions of an application-level object by including a version number into the name of the overall object.

Packet and Frame:

NDN and CCNx introduce Protocol Data Units (PDUs) which typically are larger than the maximum transmission unit of the underlying networking technology. We refer to PDUs as packets and the (potentially fragmented) packet parts that traverse MTU-bound links as frames. Handling link-layer technologies which lead to fragmentation of ICN packets is done inside the ICN network and is not visible at the service interface.

ICN Node:

A node within an ICN network can fulfill the role of a data producer, a data consumer, and/or a forwarder for Interest and Data packets. When a forwarder has connectivity to neighbor nodes, it performs Interest and Data packet forwarding in real time. It can also behave like a packet mule, that is it may carry an Interest or Data packet for some time before forwarding it to next node. An ICN node may also run routing protocols to assist its Interest forwarding decisions.

Forwarding Plane:

The canonical way of implementing packet forwarding in an ICN network relies on three data structures that capture a node's state: a Forwarding Interest Table (FIB), a Pending Interest Table (PIT), and a Content Store (CS). It also utilizes Interest forwarding strategies which takes input from both FIB and measurements to make Interest forwarding decisions. When a node receives an Interest packet, it checks its CS and PIT to find a matching entry; if no match is found, the node records the Interest in its PIT and forwards the Interest to the next hop(s) towards the requested content, based on the information in its FIB.

3. Terms by category

3.1. Generic terms

Information-Centric Networking (ICN):

A networking architecture that retrieves Data packets as response to Interest packets. Content-Centric Networking (CCNx 1.x) and Named Data Networking (NDN) are two realizations (designs) of the ICN architecture.

Data packet immutability:

After a data packet is created, it cannot change. If the content carried in the data packet is mutable, versioning should be used,

so that each version uniquely identifies an immutable instance of the content. This allows disambiguation of coordination in distributed systems.

3.2. Terms related to ICN Nodes

ICN Interface:

A generalization of the network interface that can represent a physical network interface (ethernet, wifi, bluetooth adapter, etc.), an overlay inter-node channel (IP/UDP tunnel, etc.), or an intra-node inter-process communication (IPC) channel to an application (unix socket, shared memory, intents, etc.).

Common aliases include: face.

ICN Consumer:

An ICN entity that requests Data packets by generating and sending out Interest packets towards local (using intra-node interfaces) or remote (using inter-node interfaces) ICN Forwarders.

Common aliases include: consumer, information consumer, data consumer, consumer of the content.

ICN Producer:

An ICN entity that creates Data packets and makes them available for retrieval.

Common aliases include: producer, publisher, information publisher, data publisher, data producer.

ICN Forwarder:

An ICN entity that implements stateful forwarding.

Common aliases include: ICN router.

Data Mule:

An ICN entity that temporarily stores and potentially carries an Interest or Data packet before forwarding it to next ICN entity.

3.3. Terms related to the Forwarding plane

Stateful forwarding:

A forwarding process that records incoming Interest packets in the PIT and uses the recorded information to forward the retrieved Data packets back to the consumer(s). The recorded information can also be used to measure data plane performance, e.g., to adjust interest forwarding strategy decisions.

Common aliases include: ICN Data plane, ICN Forwarding.

Forwarding strategy:

A module of the ICN stateful forwarding (ICN data) plane that implements a decision on where/how to forward the incoming Interest packet. The forwarding strategy can take input from the Forwarding Information Base (FIB), measured data plane performance parameters, and/or use other mechanisms to make the decision.

Common aliases include: Interest forwarding strategy.

Upstream (forwarding):

Forwarding packets in the direction of Interests (i.e., Interests are forwarded upstream): consumer, router, router, ..., producer.

Downstream (forwarding):

Forwarding packets in the opposite direction of Interest forwarding (i.e., Data and Interest Nacks are forwarded downstream): producer, router, ..., consumer(s).

Interest forwarding:

A process of forwarding Interest packets using the Names carried in the Interests. In case of Stateful forwarding, creating an entry in the PIT. The forwarding decision is made by the Forwarding Strategy.

Interest aggregation:

A process of combining multiple Interest packets with the same Name and additional restrictions for the same Data into a single PIT entry. Not the same as Interest suppression.

Common aliases include: Interest collapsing.

Data forwarding:

A process of forwarding the incoming Data packet to the interface(s) recorded in the corresponding PIT entry (entries) and removing the corresponding PIT entry (entries).

Satisfying an Interest:

An overall process of returning content that satisfies the constraints imposed by the Interest, most notably a match in the provided Name.

Interest match in FIB (longest prefix match):

A process of finding a FIB entry with the longest Name (in terms of Name components) that is a prefix of the specified Name.

Interest match in PIT (exact match):

A process of finding a PIT entry that stores the same Name as specified in the Interest (including Interest restrictions, if any).

Data match in PIT (all match):

A process of finding (a set of) PIT entries that can be satisfied with the specified Data packet.

Interest match in CS (any match):

A process of finding an entry in router's Content Store that can satisfy the specified Interest.

Pending Interest Table (PIT):

A database that records received and not yet satisfied Interests with the interfaces from where they were received. The PIT can also store interfaces to where Interests were forwarded, and information to assess data plane performance. Interests for the same Data are aggregated into a single PIT entry.

Forwarding Information Base (FIB):

A database that contains a set of prefixes, each prefix associated with one or more faces that can be used to retrieve Data packets with Names under the corresponding prefix. The list of faces for each prefix can be ranked, and each face may be associated with additional information to facilitate forwarding strategy decisions.

Content Store (CS):

A database in an ICN router that provides caching.

In-network storage:

An optional process of storing a Data packet within the network (opportunistic caches, dedicated on/off path caches, and managed in-network storage systems), so it can satisfy an incoming Interest for this Data packet. The in-network storages can optionally advertise the stored Data packets in the routing plane.

Opportunistic caching:

A process of temporarily storing a forwarded Data packet in the router's memory (RAM or disk), so it can be used to satisfy future Interests for the same Data, if any.

Common aliases include: on-path in-network caching

Managed caching:

A process of temporarily, permanently, or scheduled storing of a selected (set of) Data packet(s).

Common aliases include: off-path in-network storage

Managed in-network storage:

An entity acting as an ICN publisher that implements managed caching.

Common aliases include: repository, repo

ICN Routing plane:

An ICN protocol or a set of ICN protocols to exchange information about Name space reachability.

ICN Routing Information Base (RIB):

A database that contains a set of prefix-face mappings that are produced by running one or multiple routing protocols. The RIB is used to populate the FIB.

3.4. Terms related to Packet Types

Interest packet:

A network-level packet that expresses the request for a data packet using either an exact name or a name prefix. An Interest packet may optionally carry a set of additional restrictions (e.g., Interest selectors). An Interest may be associated with additional information to facilitate forwarding and can include Interest lifetime, hop limit, forwarding hints, labels, etc. In different ICN designs, the set of additional associated information may vary.

Common aliases include: Interest, Interest message, information request

Interest Nack:

A packet that contains the Interest packet and optional annotation, which is sent by the ICN Router to the interface(s) the Interest was received from. Interest Nack is used to inform downstream ICN nodes about inability to forward the included Interest packet. The annotation can describe the reason.

Common aliases include: network NACK, Interest return.

Data packet:

A network-level packet that carries payload, uniquely identified by a name, and is directly secured.

Common aliases include: data, data object, content object, content object packet, data message, named data object, named data.

Link:

A type of Data packet whose body contains the Name of another Data packet. This inner Name is often a Full Name, i.e., it specifies the Packet ID of the corresponding Data packet, but this is not a requirement.

Common aliases include: pointer.

Manifest:

A type of Data packet that contains Full Name Links to one or more Data Packets. Manifests group collections of related Data packets under a single Name. This has the additional benefit of amortizing the signature verification cost for each Data packet referenced by the inner Links. Manifests typically contain additional metadata, e.g., the size (in bytes) of each linked Data packet and the cryptographic hash digest of all Data contained in the linked Data packets.

3.5. Terms related to Name Types

Name:

A Data packet identifier. An ICN name is hierarchical (a sequence of name components) and usually is semantically meaningful, making it expressive, flexible and application-specific (akin to a HTTP URL). A Name may encode information about application context, semantics, locations (topological, geographical, hyperbolic, etc.), a service name, etc.

Common aliases include: data name, interest name, content name.

Name component:

A sequence of octets and optionally a numeric type representing a single label in the hierarchical structured name.

Common aliases include: name segment (as in CCNx).

Packet ID:

A unique cryptographic identifier for a Data packet. Typically, this is a cryptographic hash digest of a data packet (such as SHA256), including its name, payload, meta information, and signature.

Common aliases include: implicit digest.

Selector:

A mechanism (condition) to select an individual Data packet from a collection of Data packets that match a given Interest that requests data using a prefix or exact Name.

Common aliases include: interest selector, restrictor, interest restrictor.

Nonce:

A field of an Interest packet that transiently names an Interest instance (instance of Interest for a given name).

Exact Name:

A name that is encoded inside a Data packet and which typically uniquely identifies this Data packet.

Full Name:

An exact Name with the Packet ID of the corresponding Data packet.

Prefix Name:

A Name that includes a partial sequence of Name components (starting from the first one) of a Name encoded inside a Data packet.

Common aliases include: prefix.

3.6. Terms related to Name Usage

Naming conventions:

A convention, agreement, or specification for the Data packet naming. a Naming convention structures a namespace.

Common aliases include: Naming scheme, ICN naming scheme, namespace convention

Hierarchically structured naming:

The naming scheme that assigns and interprets a Name as a sequence of labels (Name components) with hierarchical structure without an assumption of a single administrative root. A structure provides useful context information for the Name.

Common aliases include: hierarchical naming, structured naming.

Flat naming:

The naming scheme that assigns and interprets a Name as a single label (Name component) without any internal structure. This can be considered a special (or degenerated) case of structured names.

Segmentation:

A process of splitting large application content into a set of uniquely named data packets. When using hierarchically structured names, each created data packet has a common prefix and additional component representing the segment (chunk) number.

Common aliases include: chunking

Versioning:

A process of assigning a unique Name to the revision of the content carried in the Data packet. When using a hierarchically structured Name, the version of the Data packet can be carried in a dedicated Name component (e.g., prefix identifies data, unique version component identifies the revision of the data).

Fragmentation:

A process of splitting PDUs into frames so that they can be transmitted over the link with a smaller MTU size.

3.7. Terms related to Data-Centric Security

Data-Centric Security:

A security property associated with the Data packet, including data (Data-Centric) integrity, authenticity, and optionally confidentiality. These security properties stay with the data packet regardless where it is stored and how it is retrieved.

Common aliases include: directly securing data packet

Data Integrity

A cryptographic mechanism to ensure the consistency of the Data packet bits. The Data integrity property validates that the Data packet content has not been corrupted during transmission, e.g., over lossy or otherwise unreliable paths.

Data Authenticity

A cryptographic mechanism to ensure trustworthiness of a Data packet, based on a selected (e.g., by a consumer/producer) trust model. Typically, data authenticity is assured through the use of asymmetric cryptographic signatures (e.g., RSA, ECDSA), but can also be realized using symmetric signatures (e.g., HMAC) within trusted domains.

Data Confidentiality

A cryptographic mechanism to ensure secrecy of a Data packet. Data confidentiality includes separate mechanisms: content confidentiality and Name confidentiality

Content Confidentiality

A cryptographic mechanism to prevent an unauthorized party to get access to the plain-text payload of a Data packet. Can be realized through encryption (symmetric, asymmetric, hybrid) and proper distribution of the decryption keys to authorized parties.

Name Confidentiality

A cryptographic mechanism to prevent an observer of Interest-Data exchanges (e.g., intermediate router) from gaining detailed meta information about the Data packet. This mechanism can be realized using encryption (same as content confidentiality) or obfuscation mechanisms.

4. Semantics and Usage

The terminology described above is the manifestation of intended semantics of NDN and CCNx operations (what do we expect the network to do?). In this section we summarize the most commonly proposed use cases and interpretations.

4.1. Data Transfer

The networking view of NDN and CCNx is that the request/reply protocol implements a basic, unreliable data transfer service for single, named packets.

4.2. Data Transport

Data transfer can be turned into a data transport service for application-level objects by additional logic. This transport logic must understand and construct the series of names needed to reassemble the segmented object. Various flavors of transport can be envisaged (reliable, streaming, mailbox, etc).

4.3. Lookup Service

In a more distributed systems view of the basic request/reply protocol, NDN and CCNx provide a distributed lookup service: given a key value (=name), the service will return the corresponding value.

4.4. Database Access

A lookup service can be turned into into a database access protocol by using the namespace structure to specify names as access keys into a database. A name prefix therefore stands for a collection or table of a database, while the rest of the name specifies the query expression to be executed.

4.5. Remote Procedure Call

The names as defined here for Interests and Data can refer to Remote Procedure call functions, their input arguments, and their results.

Interest match in FIB (longest prefix match):

A process of finding a FIB entry with the longest Name (in terms of Name components) that is a prefix of the specified Name.

Interest match in PIT (exact match):

A process of finding a PIT entry that stores the same Name as specified in the Interest (including Interest restrictions, if any).

Data match in PIT (all match):

A process of finding (a set of) PIT entries that can be satisfied with the specified Data packet.

Interest match in CS (any match):

A process of finding an entry in router's Content Store that can satisfy the specified Interest.

5. IANA Considerations

There are no IANA considerations related to this document.

6. Security Considerations

This document introduces no new security considerations.

7. Informational References

[I-D.irtf-icnrg-disaster]
Seedorf, J., Arumaithurai, M., Tagami, A., Ramakrishnan, K., and N. Blefari-Melazzi, "Research Directions for Using ICN in Disaster Scenarios", draft-irtf-icnrg-disaster-04 (work in progress), February 2019.

[RFC7476] Pentikousis, K., Ed., Ohlman, B., Corujo, D., Boggia, G., Tyson, G., Davies, E., Molinaro, A., and S. Eum, "Information-Centric Networking: Baseline Scenarios", RFC 7476, DOI 10.17487/RFC7476, March 2015, <http://www.rfc-editor.org/info/rfc7476>.

[RFC7927] Kutscher, D., Ed., Eum, S., Pentikousis, K., Psaras, I., Corujo, D., Saucez, D., Schmidt, T., and M. Waehlisch, "Information-Centric Networking (ICN) Research Challenges", RFC 7927, DOI 10.17487/RFC7927, July 2016, <http://www.rfc-editor.org/info/rfc7927>.

[RFC7933] Westphal, C., Ed., Lederer, S., Posch, D., Timmerer, C., Azgin, A., Liu, W., Mueller, C., Detti, A., Corujo, D., Wang, J., Montpetit, M., and N. Murray, "Adaptive Video Streaming over Information-Centric Networking (ICN)", RFC 7933, DOI 10.17487/RFC7933, August 2016, <http://www.rfc-editor.org/info/rfc7933>.

[RFC7945] Pentikousis, K., Ed., Ohlman, B., Davies, E., Spirou, S., and G. Boggia, "Information-Centric Networking: Evaluation and Security Considerations", RFC 7945, DOI 10.17487/RFC7945, September 2016, <http://www.rfc-editor.org/info/rfc7945>.

[RFC8569] Mosko, M., Solis, I., and C. Wood, "Content-Centric Networking (CCNx) Semantics", RFC 8569, DOI 10.17487/RFC8569, July 2019, <https://www.rfc-editor.org/info/rfc8569>.

[RFC8609] Mosko, M., Solis, I., and C. Wood, "Content-Centric Networking (CCNx) Messages in TLV Format", RFC 8609, DOI 10.17487/RFC8609, July 2019, <https://www.rfc-editor.org/info/rfc8609>.

Appendix A. Acknowledgments

Mark Mosco provided much guidance and helpful precision in getting these terms carefully formed and the definitions precise. Marie-Jose Montpetit did a through IRSG review, which helped a lot to finalise the text.

Authors' Addresses

Bastiaan Wissingh
TNO

EMail: bastiaan.wissingh@tno.nl

Christopher A. Wood
University of California Irvine

EMail: woodc1@uci.edu

Alex Afanasyev
Florida International University

EMail: aa@cs.fiu.edu

Lixia Zhang
UCLA

EMail: lixia@cs.ucla.edu

David Oran
Network Systems Research & Design

EMail: daveoran@orandom.net

Christian Tschudin
University of Basel

EMail: christian.tschudin@unibas.ch

ICNRG
Internet-Draft
Intended status: Experimental
Expires: December 3, 2016

M. Mosko
PARC, Inc.

June 1, 2016

CCNx Content Object Chunking
draft-mosko-icnrg-ccnxchunking-02

Abstract

This document specifies a chunking protocol for dividing a user payload into CCNx Content Objects. This includes specification for the naming convention to use for the chunked payload and a field added to a Content Object to represent the last chunk of an object.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on December 3, 2016.

Copyright Notice

Table of Contents

1. Introduction

CCNx Content Objects [CCNSemantics] are sized to amortize cryptographic operations over user data while simultaneously staying a reasonable size for transport over today's networks. This means a Content Object is usually within common UDP or jumbo Ethernet size. If a publisher has a larger amount of data to associate with a single Name, the data should be chunked with this chunking protocol. This protocol uses state in the Name and in an optional field within the Content Object. A chunked object may also have an external metadata content object that describes the original pre-chunked object.

CCNx uses two types of messages: Interests and Content Objects [CCNSemantics]. An Interest carries the hierarchically structured variable-length identifier (HSVLI), or Name, of a Content Object and serves as a request for that object. If a network element sees multiple Interests for the same name, it may aggregate those Interests. A network element along the path of the Interest with a matching Content Object may return that object, satisfying the Interest. The Content Object follows the reverse path of the Interest to the origin(s) of the Interest. A Content Object contains the Name, the object's Payload, and the cryptographic information used to bind the Name to the payload.

This specification adds a new segment to the Name TLV for conveying the chunk number. It updates [CCNMessages]. It also provides guidelines for the usage of the Key Locator in chunked objects.

Packets are represented as 32-bit wide words using ASCII art. Because of the TLV encoding and optional fields or sizes, there is no concise way to represent all possibilities. We use the convention that ASCII art fields enclosed by vertical bars "|" represent exact bit widths. Fields with a forward slash "/" are variable bitwidths, which we typically pad out to word alignment for picture readability.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. Chunking

Chunking, as used in this specification, means serializing user data into one or more chunks, each encapsulated in a CCNx Content Object. A chunk is a contiguous byte range within the user data. One segment in the Name of that Content Object represents the chunk number. A field in the Content Object - only mandatory in the final chunk - represents the end of the stream. Chunks are denoted by a serial counter, beginning at 0 and incrementing by 1 for each contiguous chunk. The chunking ends at the final chunk. No valid user data exists beyond the final chunk, and reading beyond the final chunk MUST NOT return any user data.

Chunking MUST use a fixed block size, where only the final chunk MAY use a smaller block size. This is required to allow a reader to seek to a specific byte offset once it knows the block size. The blocksize may be inferred from the size of the first chunk of user data. The first chunk of user data may not be chunk 0.

Because of the requirement for a fixed block size, the inclusion of certain cryptographic fields in the same content objects as user data would throw off the ability to seek. Therefore, it is RECOMMENDED that all required cryptographic data, such as public keys or key name links, be included in the leading chunks before the first byte of user data. User data SHOULD then run continuously and with the same block size through the remainder of the content objects.

This draft introduces a new Name path segment TLV type, called the ChunkNumber name segment. The ChunkNumber name segment is the serial order of the chunks. It MUST begin at 0 and MUST be incremented by 1. The ChunkNumber name segment is appended to the base name of the user data, and is usually the last name segment.

The new Content Object field is the EndChunkNumber. It MUST be included in the Content Object which is the last chunk of user data, but SHOULD be present at the earliest time it is known. The value of the EndChunkNumber should be the network byte order value of the last ChunkNumber. For example, if 3000 bytes of user data is split with a 1200 byte block size, there will be 3 chunks: 0, 1, and 2. The EndChunkNumber is 2.

The EndChunkNumber may be updated in later Chunks to a larger value, as long as it has not yet reached the end. The EndChunkNumber SHOULD NOT decrease. If a publisher wishes to close a stream before reaching the End Chunk, it should publish empty Content Objects to fill out to the maximum EndChunkNumber ever published. These padding chunks MUST contain the true EndChunkNumber.

2.1. Cryptographic material

Chunk 0 SHOULD include the public key or key name link used to verify the chunked data. It is RECOMMENDED to use the same key for the whole set of chunked data. If a publisher uses multiple keys, then the public key or key name link for all keys SHOULD be in the leading chunks before any user data.

The rationel for putting all cryptographic data up front is because the protocol requires using a fixed block size for all user data to enable seeking in the chunked stream.

2.2. Examples

Here are some examples of chunked Names using the Labeled Content Identifier URI scheme in human readable form (ccnx:).

In this example, the content producer publishes a JPG that takes 4 Chunks. The EndChunkNumber is missing in the first content object (Chunk 0), but is known and included when Chunk 1 is published. It is omitted in Chunk 2, then appears in Chunk 3, where it is mandatory.

```
ccnx:/Name=parc/Name=picture.jpg/Chunk=0  --
ccnx:/Name=parc/Name=picture.jpg/Chunk=1  EndChunkNumber=3
ccnx:/Name=parc/Name=picture.jpg/Chunk=2  --
ccnx:/Name=parc/Name=picture.jpg/Chunk=3  EndChunkNumber=3
```

In this example, the publisher is writing an audio stream that ends before expected so the publisher fills empty Content Objects out to the maximum ChunkNumber, stating the correct EndChunkNumber. Chunks 4, 5, and 6 do not contain any new user data.

```
ccnx:/Name=parc/Name=talk.wav/Chunk=0  --
ccnx:/Name=parc/Name=talk.wav/Chunk=1  EndChunkNumber=6
ccnx:/Name=parc/Name=talk.wav/Chunk=2  --
ccnx:/Name=parc/Name=talk.wav/Chunk=3  EndChunkNumber=3
ccnx:/Name=parc/Name=talk.wav/Chunk=4  EndChunkNumber=3
ccnx:/Name=parc/Name=talk.wav/Chunk=5  EndChunkNumber=3
ccnx:/Name=parc/Name=talk.wav/Chunk=6  EndChunkNumber=3
```

3. TLV Types

This section specifies the TLV types used by CCNx chunking.

3.1. Name Types

CCNx chunking uses two new Name types: Chunk Number and Chunk Metadata.

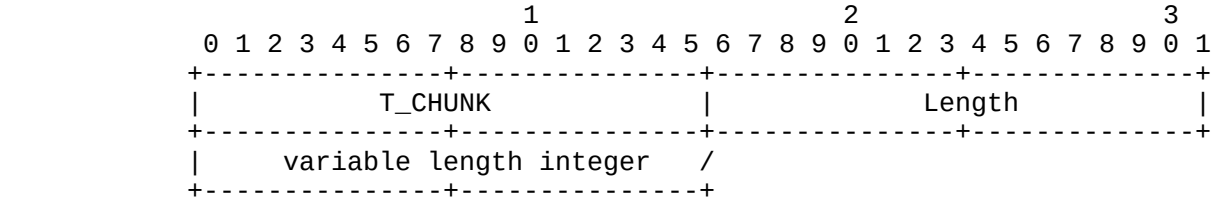
Type	Abbrev	Name	Description
%x0010	T_CHUNK	Chunk Number (Section 3.1.1)	The current Chunk Number, is an unsigned integer in network byte order without leading zeros. The value of zero is represented as the single byte %x00.

Table 1: Name Types

3.1.1. Chunk Number

The current chunk number, as an unsigned integer in network byte order without leading zeros. The value of zero is represented as the single byte %x00.

In ccnx: URI form, it is denoted as "Chunk".



3.2. Protocol Information

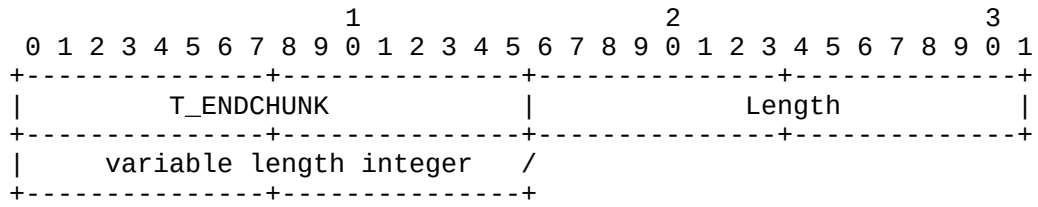
CCNx chunking introduces one new TLV for use in a Content Object.

Type	Abbrev	Name	Description
%x000C	T_ENDCHUNK	EndChunkNumber (Section 3.1.1)	The last Chunk number, as an unsigned integer in network byte order without leading zeros. The value of zero is represented as the single byte %x00.

Table 2: Content Object Types

3.2.1. EndChunkNumber

The ending chunk number, as an unsigned integer in network byte order without leading zeros. The value of zero is represented as the single byte %x00.



4. Acknowledgements

5. IANA Considerations

The draft adds new types to the CCNx Name Segment Types registry and the CCNx Content Object Types registry.

6. Security Considerations

This draft does not put any requirements on how chunked data is signed or validated.

7. References

7.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

7.2. Informative References

[CCNMessages] Mosko, M., Solis, I., and C. Wood, "CCNx Messages in TLV Format (Internet draft)", 2016, <http://tools.ietf.org/html/draft-irtf-icnrg-ccnxmessages-02>.

[CCNSemantics] Mosko, M., Solis, I., and C. Wood, "CCNx Semantics (Internet draft)", 2016, <http://tools.ietf.org/html/draft-mosko-icnrg-ccnxsemantics-03>.

[CCNx] PARC, Inc., "CCNx Open Source", 2007, <http://www.ccnx.org>.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, DOI 10.17487/RFC2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <http://www.rfc-editor.org/info/rfc3552>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

Author’s Address

Marc Mosko
PARC, Inc.
Palo Alto, California 94304
USA

Phone: +01 650-812-4405
Email: marc.mosko@parc.com

ICNRG
Internet-Draft
Intended status: Experimental
Expires: October 8, 2016

M. Mosko
C. Wood
PARC, Inc.

April 6, 2016

The CCNx URI Scheme
draft-mosko-icnrg-ccnxurischeme-01

Abstract

This document defines an RFC3986 URI compliant identifier called a Labeled Segment URI in which name segments carry a label. This allows differentiation between unrelated resources with similar identifiers. This document also specifies the CCNx URI scheme, called "ccnx:," which conforms to the labeled segment encoding rules presented here. The CCNx URI scheme applies specific labels to each name segment of a URI to disambiguate between resources with similar names. This document defines a specific set of segment labels with label semantics.

Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on October 8, 2016.

Copyright Notice

Table of Contents

1. Introduction

A Labeled Segment is an URI [RFC3986] compliant convention that allows an application or protocol to embed labels in name segments, thus disambiguating the resource identified by the path. Labeled Segment URIs also allow for query and fragment components to follow the Labeled Segment form.

Some protocols may wish to disambiguate name segments between different identifier spaces, such as "version" and "page". Other protocols may wish to use a type system such as "/str=parc/int=7" and "/str=parc/str=7". Labeled Segment URIs provide an unambiguous and flexible representation in systems that allow resources with otherwise similar names.

It is not sufficient to leave the determination of type to application-specific conventions. In a networked system with multiple applications accessing resources generated by other applications, there needs to be a set of common conventions. For example, if one application uses a base 64 encoding of a frame number, e.g. base64(0xbdea), and another uses "ver=" to represent a document version, there is an ambiguity because base64(0xbdea) is the string "ver=".

Labeled Segments defines "ls-segment" as "label[:param]=value", where the value only contains unreserved, percent-encoded, or certain sub-delim characters. In the previous example, one application would say "/frame=%BD%EA" and the other would say "/ver=".

In this document, we use URI [RFC3986] terminology, therefore a URI and CCNx Name are both composed of a URI path, which is a collection of name segments. We do not use the term "name component" as was common in old CCNx. In this document, the word "segment" alone means "name segment."

URIs conforming to the CCNx URI scheme carry a label for each name segment. The contents of each name segment must conform to the label semantics. Example segment types are "Binary Segment", "Name", and "KeyId".

We use Labeled Segment URIs as the canonical, human-readable representation. There is an unambiguous, one-to-one correspondence between an absolute LS-URI path and a Labeled Name. Relative URI representations are removed during encoding, so no relative name ends up in wire format. Some labels are URIs that are IRI [RFC3987] compatible.

Labeled Names shall be used everywhere a Name is used in CCNx, such

as in the Name of an Interest or Content Object. They are also used in Links, KeyLocators, or any other place requiring a name. When encoded for the wire, a binary representation is used, depending on the specific wire format codec, which is outside the scope of this document.

This document specifies:

- the ccnx scheme.
- a canonical URI representation.

Formal grammars use the ABNF [RFC5234] notation.

TODO: We have not adopted Requirements Language yet.

1.1. Requirements Language

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

2. URI path segment grammar for label=value pairs

2.1. Labeled Segments

This section describes the formal grammar for Labeled Segments using ABNF [RFC5234] notation. We do not impose restrictions on the length of labels or values. The semantics of values are URI scheme specific, here we only describe the meta-structure of Labeled Segments. We begin by reviewing some definitions from [RFC3986] that define an absolute path URI.

```
URI           = scheme ":" hier-part [ "?" query ] [ "#" fragment ]
hier-part     = "//" authority path-abempty
                / path-absolute
                / <other path types>
path-absolute = "/" [ segment-nz *( "/" segment ) ]
segment       = *pchar
segment-nz    = 1*pchar
pchar         = unreserved / pct-encoded / sub-delims / ":" / "@"
query         = *( pchar / "/" / "?" )
fragment      = *( pchar / "/" / "?" )
pct-encoded   = "%" HEXDIG HEXDIG
unreserved    = ALPHA / DIGIT / "-" / "." / "_" / "~"
reserved      = gen-delims / sub-delims
gen-delims    = ":" / "/" / "?" / "#" / "[" / "]" / "@"
sub-delims    = "!" / "$" / "&" / "'" / "(" / ")"
                    / "*" / "+" / "," / ";" / "="
```

Labeled Segments defines a new segment type that provides unambiguous representation of a segment's label and its value. We define the top-level LS-URI as the same form as a URI, wherein each part conforms to the Label Segment grammar, which is a subset of the URI grammar.

```
LS-URI           = scheme ":" ls-hier-part ["?" ls-query]
                   ["#" fragment]
ls-hier-part     = ["//" authority] ls-path-absolute
ls-path-absolute = "/" [ first-segment *( "/" ls-segment ) ]
first-segment    = ls-segment-nz
ls-segment       = lpv-segment / v-segment
lpv-segment      = label [":" param] "=" *s-value-nz
v-segment        = *s-value-nz
ls-segment-nz    = lpv-segment-nz / v-segment-nz
lpv-segment-nz   = label [":" param] "=" s-value-nz
v-segment-nz     = s-value-nz
label            = alpha-t / num-t
param            = alpha-t / num-t
s-value-nz       = 1*(s-pchar)

ls-query         = *1 ( lpv-component / v-component
                        *( "&" (lpv-component / v-component) ) )
lpv-component    = label [":" param] "=" q-value
v-component      = q-value
q-value          = *(q-pchar)

alpha-t          = ALPHA *(ALPHA / DIGIT)
num-t            = dec-t / hex-t
dec-t            = 1*(DIGIT)
hex-t            = "0x" 1*(HEXDIG)
ls-pchar         =  unreserved / pct-encoded / ls-sub-delims
s-pchar          = ls-pchar / ":" / "@" / "&"
q-pchar          = ls-pchar / ":" / "@" / "/"
ls-sub-delims    = "!" / "$" / "'" / "(" / ")"
                       / "*" / "+" / "," / ";"
```

A Labeled Segment URI (LS-URI) contains a scheme that uses Labeled Segments, an optional authority, a labeled segment absolute path (ls-path-aboslute), an optional labeled segment query (ls-query), and a fragment. The authority is URI scheme specific and the fragment is independent of the URI scheme.

the ls-path-aboslute is a first-segment followed by zero or more "/" ls-segment. The first-segment may be empty or a non-zero ls-segment (ls-segment-nz). If it is empty, it corresponds to a 0-lenght name which typically is a default route. It is distinct from a 1-segment name with no value (which is not allowed). The first-segment MUST either be empty (the 0-lenght name) or MUST have a value. If the first-segment is an ls-segment-nz, then it will have a value.

An ls-segment may (lpv-segment) or may not (v-segment) have a label. A particular LS-URI scheme MUST define how unlabeled segments are processed, and MAY disallow them. A v-segment is an implied type.

Once the implied type is resolved, it functions like an lpv-segment.

An lpv-segment has a label, optional parameter, and optional value (s-value-nz). An empty value is a 0-length name segment with a defined type. This is distinct from a 0-length first-segment, which has neither type nor value.

lpv-segment values come from the s-pchar set, which excludes the "=" equal sign. This means that the only equal sign in a name segment must be the delimiter between the label:param and the value. Within the value, an equal sign must be percent encoded.

lpv-segment labels and values may be alpha-numeric identifiers or numbers (decimal or hexadecimal). For example, one scheme may define the labels "name", "version", and "frame". A version may be of types "date" or "serial", meaning that the version is either a date or a monotonic serial number. Some examples of resulting LS-URIs are: "/name=parc/name=csl/version:date=20130930" or "/name=alice_smith/version:serial=299". The parameters may also indicate an instance of a label, such as "/name=books/year:1=1920/year:3=1940", where there are scheme or application semantics associated with "year:1" and "year:3".

lpv-segment labels and parameters may also be numbers. For example, a protocol with a binary and URI representation may not have pre-defined all possible labels. In such cases, it could render unknown labels as their binary value, such as "/name=marc/x2003=green".

The ls-query component is a non-hierarchical set of components separated by "&". Each ls-query component is either a lpv-component or a v-component, similar to segments. They are based on q-value, which uses q-pchar that excludes "&", but includes "/". This allows an LS-URI scheme to use type query parameters.

Labeled Segments allow for dot-segments "." and ".." in a v-segment. They operate as normal. A single dot "." refers to the current hierarchy level and may be elided when the URI is resolved. Double dot ".." segments pop off the previous non-dot segment. An lpv-segment with a value of "." or ".." is not a dot-segment. It means that the value of the given label is "." or "..". For example /a=parc/b=csl/.. is equivalent to "/a=parc/b=csl", but the LS-URI "/a=parc/b=csl/c=.." does not contain a dot-segment.

2.2. URI comparison

An LS-URI scheme MUST specify the normalization rules to be used, following the methods of Section 6 [RFC3986]. At minimum, an LS-URI scheme SHOULD do the following:

- Normalize unrestricted percent-encodings to the unrestricted form.
- Normalize num-t to either dec-t or hex-t.
- If the scheme allows for value-only segments or query components and interprets them as a default type, they should be normalized to having the type specified.
- If the scheme allows for undefined labels and represents them, for example, as num-t, then it should normalize all labels to their corresponding num-t. If "name", for example, is known to be %x50 in a binary encoding of the URI, then all labels should be compared using their numeric value.

3. Application to CCNx Names

3.1. The ccnx Scheme

This section describes the CCNx URI scheme "ccnx:" for Labeled Names. A Labeled Name assigns a semantic type or label to each segment of the hierarchical content Name.

Unless otherwise specified, a name segment is an arbitrary sequence of octets.

Several name segment labels are binary unsigned integers. These are always encoded as variable length sequences of 1 or more octets in network byte order using the shortest representation (i.e. no leading %x00). The value of "0" is encoded as the single byte of "%x00". A zero-length sequence must be interpreted as "not present."

The CCNx Name segment types are:

- Name Segment: A generic name segment that includes arbitrary octets.

- Application Type N: An application may use application-specific parameters, numbered as integers, where N is from 0 to a system-specific maximum, not less than 255. These are represented as "App:1=value", for example.

It is common for an information centric networking protocol, such as CCNx or NDN, to use a binary on-the-wire representation for messages. Such protocols, if they use the ccnx: scheme, must have an appropriate codec that unambiguously represents Labeled Content Information in the chosen wire format. Relative dot-segments should not occur in the wire format, they should be resolved before encoding.

3.2. URI Representation

Typed Names use a standard RFC 3986 representation following the LS-URI convention. A name segment consists of any "unreserved" characters plus percent-encoded characters. Reserved characters must be percent encoded.

Within an absolute path, each segment consists of an "ls-segment" (c.f. LS-URI). A labeled segment is a type and a name component value, with a URI representation of "type=value". The "type=" portion may be omitted if it is type Name.

Some name types take a parameter, such as the Application types.

They are represented as "A:nnn=value", where the "nnn" is the application type number and value is the name component.

A CCNx URI MUST NOT include an Authority, Query, or Fragment. It is an error to include them.

Dot-segments (relative name components) are resolved when the URI is converted to a Typed Name. The "." dot-segment is removed. The ".." dot-segment is removed along with the previous non-dot-segment.

Type	Display	Name
'Name'	Hexadecimal	Name Segment
'IPID'	Hexadecimal	Interest Payload Identifier segment
'App:0' - 'App:255'	Hexadecimal	Application Component

Table 1: The CCNx URI Scheme Types

3.2.1. Examples

A name / is
ccnx:/ and is a 0-length name.

A name /Name= is
ccnx:/Name= and is a 1-segment name of 0-length.

A name /foo/bar.
ccnx:/Name=foo/Name=bar
ccnx:/foo/Name=bar
ccnx:/foo/bar

A name /foo/bar with key %xA0.
ccnx:/Name=foo/Name=bar/App:1=0xA0

A name /foo/bar with version %xA0 and App:2 value 0x09.
ccnx:/foo/bar/Version=0xA0/App:2=0x09

A name /foo/.., where the ".." is a literal name component,
not a relative dot-segment.
ccnx:/foo/Name=..

A name /foo/bar with application type 0 "hello"
and application type 1 "world".
ccnx:/Name=foo/Name=bar/App:0=hello/App:1=world

3.3. ccnx: URI comparison

While most comparisons are done using a wire format representation of a ccnx: URI, some applications may compare the CCNx URI using their URI representation. This section defines the rules for comparing ccnx: URIs using the methods of Section 6 [RFC3986]

Comparing typed name URIs must be done with:

- Syntax-based normalization
- Case normalization: normalize the representation of percent encodings. ccnx: does not use the host portion of the URI, and should be ignored if present.
- Percent encoding normalization: Percent encodings of unreserved characters must be converted to the unreserved character.
- Path segment normalization: dot-segments must be resolved first.

- Scheme-based normalization: The authority should be removed and the path represented as an absolute path.

- Protocol-based normalization: Should not be done. A trailing slash indicates a zero-length terminal name component and signifies a different name.

- typed-name-segment normalization: All segments should be presented with their type, do not elide the "N=" for Name components.

- Binary unsigned integer normalization: remove any leading %x00 from numbers, leaving only the terminal %x00 for "0".

- type parameters: they must have their percent encodings normalized. If they are integers, such as for the ’A’ type, they must not have leading zeros.

4. IRI Considerations

International Resource Identifiers extend the unreserved character set to include characters above U+07F and encode them using percent encoding. This extension is compatible with the ccnx: schema. It applies only to the "value" portion of an ls-segment.

The canonical name is determined by the URI representation of the IRI, after applying the rules of Section 3.1 of [RFC3987] and resolving dot-segments. The canonical name thus includes the URI representation of language markers, including the bidirectional components.

The value of a UTF-8 Name segment should be interpreted using IRI rules, including bidirectional markers. They may be displayed using localized formats.

Binary unsigned integer types are not interpreted under IRI rules, they are specifically percent encoded numbers. They may be displayed using a localized format.

5. Acknowledgements

6. IANA Considerations

This memo includes no request to IANA.

All drafts are required to have an IANA considerations section (see Guidelines for Writing an IANA Considerations Section in RFCs [RFC5226] for a guide). If the draft does not require IANA to do anything, the section contains an explicit statement that this is the case (as above). If there are no requirements for IANA, the section will be removed during conversion into an RFC by the RFC Editor.

7. Security Considerations

All drafts are required to have a security considerations section. See RFC 3552 [RFC3552] for a guide.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

8.2. Informative References

[RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations", BCP 72, RFC 3552, DOI 10.17487/RFC3552, July 2003, <http://www.rfc-editor.org/info/rfc3552>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, DOI 10.17487/RFC3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC3987] Duerst, M. and M. Suignard, "Internationalized Resource Identifiers (IRIs)", RFC 3987, DOI 10.17487/RFC3987, January 2005, <http://www.rfc-editor.org/info/rfc3987>.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, DOI 10.17487/RFC5226, May 2008, <http://www.rfc-editor.org/info/rfc5226>.

[RFC5234] Crocker, D., Ed. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, DOI 10.17487/RFC5234, January 2008, <http://www.rfc-editor.org/info/rfc5234>.

Authors' Addresses

Marc Mosko
PARC, Inc.
Palo Alto, California 94304
USA

Phone: +01 650-812-4405
Email: marc.mosko@parc.com

Christopher A. Wood
PARC, Inc.
Palo Alto, California 94304
USA

Phone: +01 650-812-4421
Email: christopher.wood@parc.com

icnrg
Internet-Draft
Intended status: Standards Track
Expires: May 4, 2017

M. Mosko
E. Uzun
C. Wood
PARC

October 31, 2016

CCNx Key Exchange Protocol Version 1.0
draft-wood-icnrg-ccnxkeyexchange-01

Abstract

This document specifies Version 1.0 of the CCNx Key Exchange (CCNxKE) protocol. The CCNxKE protocol allows two peers to establish a shared, forward-secure key for secure and confidential communication. The protocol is designed to prevent eavesdropping, tampering, and message forgery between two peers. It is also designed to minimize the number of rounds required to establish a shared key. In the worst case, it requires two RTTs between a consumer and producer to establish a shared key. In the best case, one RTT is required before sending any application data. This document outlines how to derive the keys used to encrypt traffic. An annex provides an example peer-to-peer transport protocol for exchanging encrypted CCNx communications.

Status of This Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

This Internet-Draft will expire on May 4, 2017.

Copyright Notice

Table of Contents

1. Introduction

DISCLAIMER: This is a WIP draft of CCNxKE and has not yet seen rigorous security analysis.

CCNx Key Exchange (CCNxKE) establishes ephemeral forward secure keys between two peers, called the consumer (client) and producer (server). The underlying cryptography of CCNxKE is similar to TLS 1.3, though there are some protocol changes due to the ICN nature of CCNxKE. CCNxKE also supports the concept of a MoveToken, which allows the authenticating producer to shift a session to one (or more) co-operating replicas.

CCNxKE does not specify how the keys are used. It only specifies how to derive the traffic secret that could be used to encrypt/decrypt data. The draft [draft-wood-icnrg-tlvencap] specifies one way to use the traffic secret to carry out communications in a session. Annex A also sketches out an example CCNx protocol for exchanging encrypted messages, though it is not part of this standard. Other protocols may use CCNxKE.
For example, a producer and replica may use CCNxKE to establish a shared key to use in Move Tokens. Two routers may use CCNxKE to establish MACSEC keys. A consumer and publisher could establish a symmetric key while on-line then publish content later for an off-line consumer. In short, the use of CCNxKE is not limited to a TLS-like transport protocol.

CCNxKE allows upper-layer data to be returned in Round 3, like TLS 1.3. In this sense, one can achieve 3 RTT (worst case) or 1 RTT (best case) communcations. The data put in this response is up to the protocol using CCNxKE and may or may not be used.

CCNxKE is not a substitue for data authenticity, such as Content Object provenance via signatures, group encryption of cached objects, or DRM protections. CCNxKE only creates a private, ephemeral tunnel between a consumer and a producer. CCNxKE expects that the encrypted communications protocol still carries normal CCNx packets with normal CCNx attributes such as signatures.

Some types of ICN communications require emphemeral, forward secure encryption. Typical examples are on-line banking, real-time voice, or on-line shopping. Other applications may need different types of encryption and thus not use CCNxKE. There is currently no standard way for CCNx peers to exchange emphemeral, forward secure keys, thus this RFC specifies the standard mechanism that should be used by all CCNx peers for such keys. CCNxKE is built on the CCNx 1.0 protocol and only relies upon standard Interest and Content Objects as a vehicle for communication.

In this document, the term 'CCNxKE session' refers to the key exchange session. It does not refer to a transport protocol session (like TLS) that uses the derived keys.

This protocol has the following four main properties:

- Each peer's identity can be authenticated using asymmetric, or public key, cryptography (e.g., RSA [RSA], ECDSA [ECDSA], etc.). Server authentication is mandatory whereas mutual authentication is optional.

- The negotiation of a forward-secure shared secret is protected from eavesdroppers and man-in-the-middle (MITM) attacks.

- The negotiation is reliable: no attacker can modify the negotiation communication without being detected by the parties to the communication.

- The state of a CCNxKE session can be securely migrated between an endpoint performing authentication and that which provides content using a "move token." This allows authentication and authorization to be separated from encryption for a session, enabling different systems to complete these steps.

Usage of CCNxKE is entirely independent of upper-layer application protocols. CCNxKE may be used for any purpose that requires producer authentication and shared emphemeral forward-secure keys.

CCNxKE also introduces a new type of cookie based on reverse hash chains [HASHCHAIN] to help limit the amount of significant server work done in response to a client or consumer Interest. TCP-based protocols, such as TLS [TLS13], use the TCP 3-way handshake for such proof. UDP-based protocols, such as QUIC [QUIC] and DTLS 1.2 [DTLS12], use an optional session address token or cookie that must be presented by the client (consumer) to prove ownership of an address during a key exchange procedure. Without source addresses, our cookie technique ensures that the same entity which requested server information, e.g., the public configuration data, is the same entity that wishes to complete a key exchange.

The main contribution of this work is adapting key exchange principles to the pull-based CCNx communication model. CCNxKE only assumes that a consumer knows a first name prefix to initiate the key exchange. The first Interest does not need to be a CCNxKE packet -- the producer can signal back to the consumer that it requires a transport protocol using CCNxKE in the response.

This specification does not subsume other ICN-compliant key exchange protocols. Nor does its existence imply that all encryption in an ICN must be based on sessions. It was designed specifically to solve the problem of session-based encryption in ICN.

1.1. Conventions and Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "NOT RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [RFC2119].

The following terms are used:

Consumer/Client: The CCN consumer initiating the CCNxKE key exchange via a first Interest.

Producer/Server: The CCN producer receiving or accepting the CCNxKE key exchange request request Interest.

Sender: An endpoint that originates a message.

Receiver: An endpoint that is receiving messages.

Peer: An endpoint. When discussing a particular endpoint, "peer" refers to the endpoint that is remote to the primary subject of discussion.

Connection: A network path of n >= 1 hops between the consumer and producer.

Endpoint: Either the consumer or producer of the connection.

Handshake: A series of message exchanges between two peers that is used to perform a task (e.g., perform key exchange and derivation).

Session: An association between a consumer and a producer resulting from a CCNxKE handshake.

DH: A Diffie Hellman key exchange procedure [RFC2631] [DH].

Key Share: One half of the shared-secret provided by one peer performing a DH key exchange.

Forward-secure: The property that compromising any long-term secrets (e.g., cryptographic keys) does not compromise any session keys derived from those long-term secrets.

CONFIG information: A data structure created by a producer which contains long-term cryptographic material and associated information needed by a client to initiate a key-exchange with the producer.

HELLO exchange: An exchange between a consumer and producer wherein the consumer retrieves the CONFIG information from the producer.

Payload: The payload section of a CCNxMessage as defined in [CCNxMessages].

KEPayload: A payload for information used in the CCNxKE protocol which is a generic key-value store. The KEPayload is _not_ the CCNxMessage payload.

CCNxName: A CCNxName as defined in [CCNxMessages].

Semi-static: Short-term.

Short-term Secret (SS): A secret which is derived from the server’s semi-static DH share and the client’s fresh DH share.

Forward-secure Secret (FSK): A secret which is derived from fresh (i.e., generated on demand at random) DH shares from both the consumer and producer for the given connection.

HKDF: Hash-based key-derivation function [RFC5869].

2. Goals

The goals of the CCNxKE protocol, in order of priority, are as follows:

1. Cryptographic security: CCNxKE should be used to securely establish a session and all related shared secrets between two peers. Cryptographic properties of interest include: (a) forward-secure session key derivation and (b) (state and computational) denial-of-service prevention at the producer (see [RFC4987]) that is no worse than DTLS 1.2 [DTLS12]}. For property (a), different keys (and relevant algorithm parameters such as IVs) are established for each communication direction, i.e., from consumer to producer and producer to consumer. For property (b), we use a new type of stateless cookie inspired by that of DTLS 1.2.

2. Interoperability: Independent programmers should be able to develop applications utilizing CCNxKE that can successfully exchange cryptographic parameters without knowledge of one another’s code.

3. Extensibility: CCNxKE seeks to provide a framework into which new public key and symmetric key methods and algorithms can be incorporated without breaking backwards compatibility or requiring all clients to implement new functionality. Moreover, the protocol should be able to support a variety of peer authentication protocols, e.g., EAP-TLS, EAP-PWD, or a simple challenge-response protocol.

4. Relative efficiency: CCNxKE tries to create sessions with minimal computation, bandwidth, and message complexity. In particular, it seeks to create sessions with as few end-to-end round trips as possible, and also provide support for accelerated session establishment and resumption when appropriate. At most 2 round-trip-times (RTTs) should be used to establish a session key, with the possibility of 1-RTT accelerated starts and resumption.

3. Scope

This document and the CCNxKE protocol are influenced by the TLS 1.3 [TLS13], QUIC [QUIC], and DTLS 1.2 [DTLS12] protocols. The reader, however, does not need a detailed understanding of those protocols to understand this document. Moreover, where appropriate, references to related protocols are made for brevity and technical clarity. This document is intended primarily for readers who will be implementing the protocol and for those doing cryptographic analysis of it. The specification has been written with this in mind and it is intended to reflect the needs of those two groups.

Unlike TLS, this document does not specify the transport protocol. It specifies the establishment of a session ID and shared keys. Other documents specify the use of CCKxKE within a transport protocol.

This document is not intended to supply any details of service definition or of interface definition, although it does cover select areas of policy as they are required for the maintenance of solid security.

4. Presentation Language

This document uses a presentation language of remote calls (i.e. packet messages) similar to the format used by TLS [TLS13].

5. CCNxKE Overview

5.1. Connection Establishment Latency

CCNxKE operates in three rounds, where each round requires a single RTT to complete. The full execution of the protocol therefore requires 2 RTTs before a session is fully established. The full version is used when consumers have no a priori information about the producer. An accelerated one round version is used when the consumer has valid configuration information and a source cookie from the producer; this variant requires 1 RTT before a session is established.

5.2. Connection Migration and Resumption

CCN end hosts lack the notion of addresses. Thus, the producer endpoint for a given execution of the CCNxKE protocol is one which can authoritatively serve as the owner of a particular namespace. For example, a consumer may wish to establish a session with a producer who owns the /company/foo namespace. The specific end host which partakes in the protocol instance is not specified, by virtue of the fact that all CCNxKE messages are based on well-defined names. This enables the producer end-host which partakes in the protocol to change based on the name of the CCNxKE messages. Consequently, to maintain correctness, it is important that a single execution of the protocol operates within the same trusted context; this does not mean that the same producer end-host is required to participate in all three steps of the protocol. Rather, it means that the end-host responding to a CCNxKE message must be trusted by the consumer to complete the exchange. CCNxKE is designed to enable this sort of producer migration.

For example, a consumer may use an initial name like ’/parc/index.html’ that works like an IP any cast address and could got to one of several systems. CCNxKE allows the responding endpoint to include a localized name to ensure that subsequent messages from the consumer come back to the same producer. CCNxKE also allows the key exchange peer to securely hand-off the session to a content producer peer via another name and session token once the client is authenticated and keying material is exchanged.

5.3. Re-Transmissions, Timeouts, and Replay Prevention

CCNxKE timeouts and retransmissions are handled using the approach in [RFC6347]. One primary difference is that timer values may need to be adjusted (elongated) due to prefix shifts and the need for a producer to transfer security information between different machines.

Replay attack prevention is also an optional feature, and if used, MAY be done using one of the following two approaches at the receiver (producer):

- IPSec AH [RFC4302] and ESP [RFC4303] style replay detection based on sliding windows and monotonically increasing sequence numbers for windows. Note that the sliding window inherently limits the performance of the protocol to the window size, since only a finite number of messages may be received within a given window (based on the window size).

- The optimized anti-replay algorithm of [RFC6479].

5.4. Loss Sensitivity

CCNxKE messages are transferred using standard CCN Interest and Content Objects and are therefore subject to loss as any datagram. This means that traffic encrypted with keys derived from CCNxKE must be stateless. They cannot depend on in-order arrival. This problem is solved by two mechanisms: (1) by prohibiting stream ciphers of any kind and (2) adding sequence numbers to each message that allow the receiver to identify and use the correct cryptographic state to decrypt the message. Moreover, sequence numbers permit anti-replay mechanisms similar to those used in DTLS [DTLS12] as mentioned above.

6. The CCNxKE Protocol

This section describes the CCNxKE protocol in detail at the message level. The specific encoding of those messages is given later. CCNxKE could be adapted to different wire format encodings, such as those used by the NDN protocol.

The following assumptions are made about peers participating in the CCNxKE protocol:

- Consumers know the namespace prefix of the producer for which they wish to execute the CCNxKE protocol.

- CCNxKE protocol information is carried in a distinguished field outside of the payload of CCN messages. This is done to distinguish key exchange material with application data in a message. This is necessary for 1 RTT packets that carry both keying material and application payload.

- CCNxKE does not require any special behavior of intermediate systems to forward packets.

- CCNxKE packets generally should not be cached for significant periods of time, as use normal protocol methods to limit caching. Part of this is achieved through the use of consumer-specific nonces in names.

6.1. Round Overview

CCNxKE is composed of three rounds. The purpose of each round is described below.

- Round 1: Perform a bare HELLO exchange to obtain the extensions (parameters) for the key exchange provided by the producer and a source cookie to prove ownership of the "source" of the request.

- Round 2: Perform the initial FULL-HELLO exchange to establish a forward-secure key used for future communication, i.e., Interest and Content Object exchanges in the context of the newly established session.

- Round 3: Send the first bit of application data and (optionally) transfer resumption cookie(s) from the producer to the consumer.

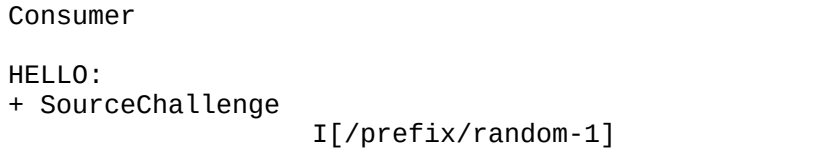
Conceptually, there are two secrets established during a single execution of CCNxKE:

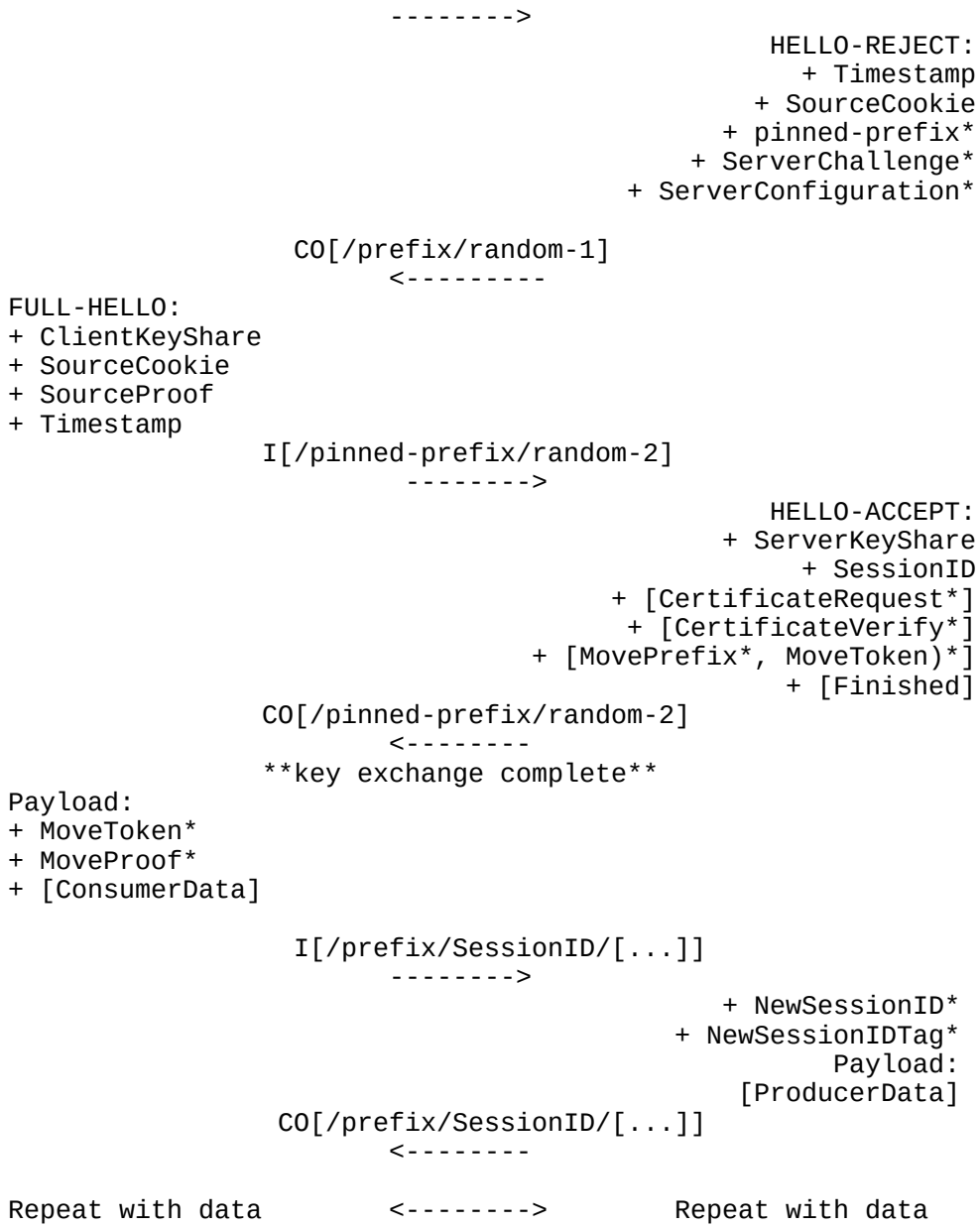
- Static Secret (SS): A secret which is derived in one of two ways: (a) from the client and server ephemeral key shares and (b) from the server’s semi-static share and the client’s ephemeral key share. Keying material derived from SS in option (a) is not forward secure.

- Ephemeral Secret (ES): A secret which is derived from both the client and server ephemeral key shares.

Depending on the mode in which CCNxKE is used, these secrets can be established in a variety of ways. Key derivation details are outlined in Section Section 9.

All secrets are derived with the appropriate amount of randomness [RFC4086]. An overview of the messages sent in each of the three rounds to establish and use these secrets is shown in Figure Figure 1 below. This diagram omits some parts of each message for brevity.





* Indicates optional or situation-dependent messages that are not always sent.

{} Indicates messages protected using keys derived from the short-term secret (SS).

() Indicates messages protected using keys derived from the ephemeral secret (ES).

[] Indicates messages protected using keys derived from the traffic secret (TS).

Figure 1: High-level message flow for full CCNxKE protocol with a maximum 2-RTT delay.

In the following sections, we will describe the format of each round in this protocol in more detail.

We do not specify the encoding of CCNxKE data sent in Interest and Content Object payloads. Any viable encoding will suffice, so long as both parties agree upon the type. For example, the payload could be structured and encoded as a JSON object, e.g.,

{ "ClientKeyShare" : 0xaa, "SourceCookie" : 0xbb, "SourceProof" : 0xbb, ... }

For now, we assume some valid encoding mechanism is used to give structure to message payloads. Moreover, we assume that these payloads are carried in a distinguished CCNxKE payload field contained in the Interest and Content Objects.

6.2. Round 1

The purpose of Round 1 is to acquire a cookie to binding the exchange to the initial consumer and the public configuration information contained in the ServerConfiguration structure. This information is used in the second round when performing the actual key exchange. To that end, the format of the Round 1 message is trivial. First, the client issues an Interest with the following name

/prefix/random-1

where random-1 is a randomly generated 64-bit nonce. This interest carries a KEPayload with the following information:

HELLO Field	Description	Optional?
SourceChallenge	A random value generated to prove ownership of the consumer's "source"	No

Upon receipt of this interest, the producer responds with a HELLO-REJECT Content Object whose KEPayload has the following fields:

HELLO-REJECT Field	Description	Optional?
Timestamp	Current server timestamp	No
SourceCookie	A cookie that binds the consumer's challenge to the current timestamp	No
PinnedPrefix	A new prefix that pins the key exchange to a particular server	Yes
ServerConfiguration	The public server configuration information	Yes
ServerChallenge	A random value for the consumer to include in its CertificateVerify if the server requires client authentication	Yes

The Timestamp and SourceCookie are used in Round 2. Their derivation is described later. If the server provides a PinnedPrefix then the consumer must use this prefix in Round 2 in lieu of the Round 1 name prefix. (This is because the PinnedPrefix identifies a particular endpoint that is capable of completing the key exchange.)

The ServerConfiguration information is a semi-static catalog of information that consumers may use to complete future key exchanges with the producer. The fields of the ServerConfiguration information are shown below.

ServerConfiguration Field	Description	Optional?
KEXS	Supported elliptic-curve key-exchange algorithms	No
AEAD	Supported AEAD algorithms	No
PUBS	List of public values (for key exchange algorithm) encoded appropriately for the given group	No
EXPRY	Expiration timestamp (i.e., longevity of the ServerConfiguration structure)	No
VER	Version of the CONFIG structure	Yes
CERT	Server certificate	No
SIG	Signature produced by the server over the entire ServerConfiguration message	No

The KEXS is a data structure that enumerates the elliptic curve key-exchange algorithms that are supported by the producer (see [QUIC] for more details). Currently, only the following curves are supported:

- Curve25519

- P-256

Selection criteria for these curves is given at http://safecurves.cr.yp.to/.

The AEAD structure enumerates the supported AEAD algorithms used for symmetric-key authenticated encryption after the session has been established. Currently, the only supported algorithms are:

- AES-GCM-(128,192,256) [GCM]: a 12-byte tag is used, where the first four bytes are taken from the FSK key-derivation step and the last eight are taken from the initial consumer nonce.

- Salsa20 [SALSA20] (stream cipher) with Poly1305 (MAC).

The key sizes and related parameters are provided with the AEAD tag in the CONFIG structure.

The PUBS structure contains the public values for the initial key exchange. Both Curve25519 and P-256 provide their own set of accepted parameters. Thus, the only values provided here are the random curve elements used in the DH operation.

The EXPRY value is an absolute timestamp that indicates the longevity of the ServerConfiguration.

The CERT and SIG values contain the server's certificate and a signature generated over the entire ServerConfiguration field. This signature is generated with the corresponding private key.

6.3. Round 2

The purpose of Round 2 is to perform the initial FULL-HELLO exchange to establish a forward-secure key used for future communication. It is assumed that the consumer already has the ServerConfiguration information that is provided from the producer in Round 1. It is also assumed that the consumer has a

Moreover, assume that nonce2 is a ephemeral nonce provided by the producer in Round 1. Then, the consumer issues an Interest with the following name:

```
    /prefix/random-2
```

and a KEPayload with the following information:

FULL-HELLO Field	Description	Optional?
ClientKeyShare	The client's key share for the key exchange	No
SourceCookie	SourceCookie provided by the server in Round 1	No
SourceProof	The SourceCookie construction proof provided by the client	No
Timestamp	The timestamp provided by the server in Round 1	No
ConsumerPrefix	The consumer's prefix that can be used for the producer to	Yes

	send interests to the consumer	
PreSharedKey	A pre-shared key that can be configured between a consumer and producer	Yes
ResumptionCookie	The ResumptionCookie derived from a past session	Yes
{MoveChallenge}	A move challenge generated identically to the SourceChallenge	Yes
{AlgChoice}	Algorithm (KEXS and AEAD) options choice (a list of tags echoed from the ServerConfiguration)	No
{Proof}	Proof of demand (i.e., a sorted list of types of proof the consumer will expect)	No
{CCS}	Compressed certificate set that the consumer possesses	No
{ConsumerData}	Application data encrypted under a key derived from SS (in a 1-RTT exchange)	Yes
ServerNameIndication	A server name indication (as a CCNxName) defined in Section 3 of [RFC6066]	Yes
Certificate	The client’s certificate	Yes
CertificateVerify	A signature generated over the entire FULL-HELLO message	Yes

((TODO: provide more details about each of these fields))

Upon receipt of this interest, the producer performs the DH computation to compute ES and SS, decrypts all protected fields in the consumer’s KEPayload, and validates the algorithm choice selection (AlgChoice). If any of these steps fail, the producer replies with with a HELLO-REJECT Content Object whose KEPayload contains a REJ flag and the reason of the error. The REJ flag and value are encrypted by the SS (if possible).

If the above steps complete without failure or error, then the producer responds with a Content Object whose KEPayload has the following fields:

HELLO-ACCEPT Field	Description	Optional?
SessionID	Cleartext session identifier	No
ServerKeyShare	Server’s key share for the ES derivation	No
{ServerExtensions}	Additional extensions provided by the server, encrypted under ES	Yes
[ResumptionCookie]	Resumption cookie encrypted under a TS-derived key	Yes
{(MovePrefix,MoveToken)}	Third CCNxName prefix and token to use when moving to session establishment	Yes
CertificateRequest*	Server certificate that matches the type of proof provided by the client	Yes
CertificateVerify*	Signature generated over the entire HELLO-ACCEPT message	Yes

If a MovePrefix and MoveToken tuple is provided then in the HELLO-ACCEPT message then a CertificateVerify (signature) MUST also be provided in the response.

6.4. Round 3

In Round 3, the consumer sends interests whose name and optional Payload are encrypted using one of the forward-secure keys derived after Round 2. In normal operation, the producer will respond with Content Objects whose Payloads are encrypted using a different forward-secure key. That is, interests and Content Objects are encrypted and authenticated using two separate keys. The producer may also optionally provide a new resumption cookie (RC) with a Content Object response. This is used to keep the consumer’s

resumption cookie fresh and to also support 0 RTT resumption. In this case, the producer's Content Object response has the following fields:

- Payload: the actual Content Object payload data encrypted with the producer's forward-secure key.

- ResumptionCookie: A new resumption cookie to be used for resuming this session in the future.

The producer is free to choose the frequency at which new resumption cookies are issued to the consumer.

The producer may also reply with a new SessionID. This is done if the client presented a MoveToken and MoveProof. A NewSessionID must be accompanied with a NewSessionIDTag, which is equal to the HMAC of NewSessionID computed with the traffic-secret key. A client MUST then use NewSessionID instead of SessionID after verifying the NewSessionIDTag.

7. Alternative Exchanges

CCNxKE also supports one-round key exchange and session resumption. These variants are outlined below. The key material differences are described later. In these variants, we use message ExchangeSourceCookie to denote the following exchange:

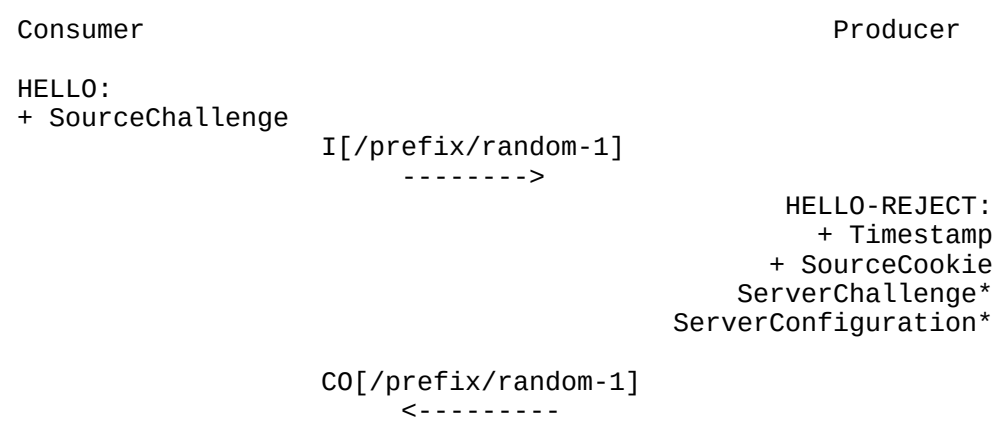
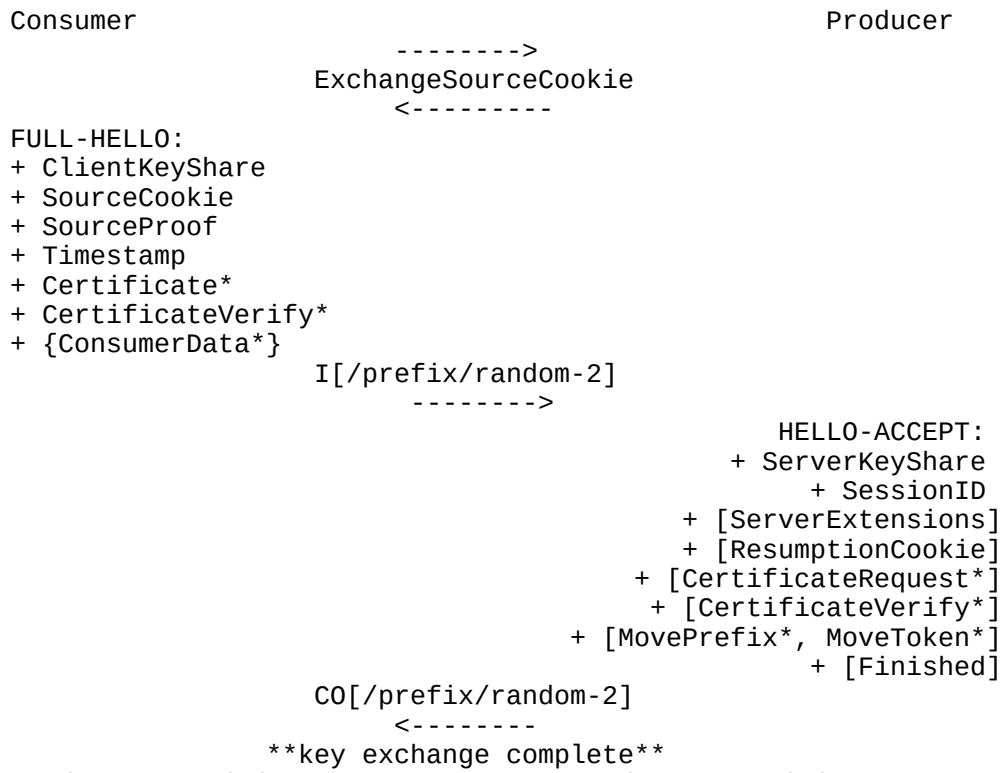


Figure 2: SourceCookie exchange -- ExchangeSourceCookie.

7.1. One-RTT Exchange



* Indicates optional or situation-dependent messages that are not always sent.

{} Indicates messages protected using keys derived from the short-term secret (SS).

() Indicates messages protected using keys derived from the ephemeral secret (ES).

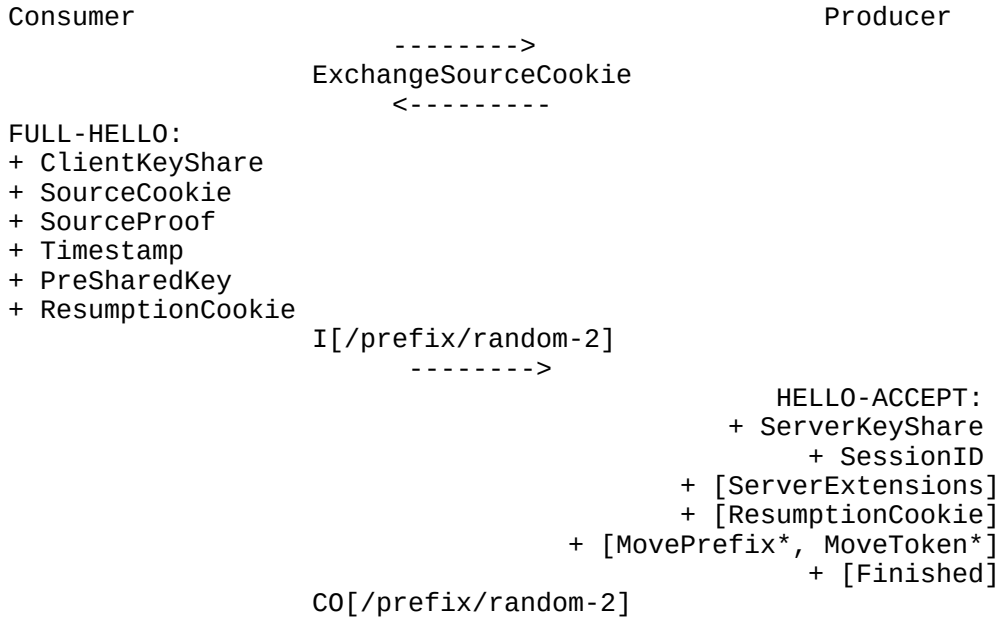
[] Indicates messages protected using keys derived from the traffic secret (TS).

Figure 3: Exchange with 1 RTT.

As with TLS, the initial application data is protected with the

8. Resumption and PSK Mode

In this mode, the client uses its ResumptionCookie to re-create a previous session. The client also provides a key share in case the server opts to fall back and establish a fresh key. If the server accepts the ResumptionCookie then it MUST issue a new SessionID and ResumptionCookie for future use with the client.



* Indicates optional or situation-dependent messages that are not always sent.

{} Indicates messages protected using keys derived from the short-term secret (SS).

() Indicates messages protected using keys derived from the ephemeral secret (ES).

[] Indicates messages protected using keys derived from the traffic secret (TS).

Figure 4: Exchange with 1 RTT.

9. Secret Derivation

In this section we describe how secrets used in the protocol are derived. We cover the SourceCookie, MoveToken, SessionID, ResumptionCookie, and the actual traffic keys.

9.1. SourceCookie Derivation

The intention of the SourceCookie is to prove that a client is sending interests from a legitimate location before any server computation is done. Without this, a Denial of Service attack could be carried out by sending interests to the server with the intention of triggering wasted computation. TCP-based protocols prevent this with the SYN-flood cookie mechanism. Protocols based on UDP use cookies that bind to the client address [DTLS12]. Since CCN lacks any notion of a source address, these cookie mechanisms do not apply. Instead, we need a way for clients to prove that they initiated a key exchange from the "same origin." We now describe the cookie mechanism that gives us this guarantee.

Instead of a source address, a SourceCookie is computed using a challenge provided by a consumer. To create this challenge, a consumer first generates a a randomly generated 256-bit string X. The consumer then computes SourceChallenge = SHA256(X). Upon receipt of this challenge, the producer generates a SourceCookie as follows:

SourceCookie = HMAC(k, SourceChallenge || timestamp)

where timestamp is the current server timestamp and k is the server’s secret key. To prove ownership of the "source," the consumer then provides the SourceCookie and a SourceProof in the round 2 Interest. The SourceProof is set to the value X used to derive the SourceChallenge. Upon receipt of the SourceProof, the server verifies the following equality:

SourceCookie = HMAC(k, SHA256(SourceProof) || timestamp)

If this check passes, then the server continues with the computationally expensive part of the key exchange protocol.

9.2. Move Derivation

The MoveChallenge and MoveProof are computed identically to the SourceChallenge and SourceProof. The MoveToken, however, is left as an opaque bit string. Extensions may be specified to describe how to compute this value.

9.3. SessionID and ResumptionCookie Properties, Derivation, and Usage

The purpose of the session identifier SessionID is to uniquely identify a single session for the producer and consumer. A Producer MAY use a random bit string or MAY use the method described in this section or MAY use another proprietary method to distinguish clients.

We provide a more secure creation of the SessionID since it is used with the ResumptionCookie derivation (defined later). Specifically, the SessionID is derived as the encryption of the hash digest of a server secret, TS, and an optional prefix (e.g., MovePrefix).

Encryption is done by the using a long-term secret key owned by the server used for only this purpose, i.e., it is not used for consumer traffic encryption. Mechanically, this derivation is:

SessionID = Enc(k1, H(TS || (Prefix3))),

where k1 is the long-term producer key.

For the resumption cookie, we require that it must be able to be used to recover the TS for a given session. Without TS, correct session communication is not possible. We derive it as the encryption of the hash digest of the server secret, TS, and the optional (MovePrefix, MoveToken) tuple (if created for the session). The producer must use a long-term secret key for this encryption. Mechanically, this derivation is:

ResumptionCookie = Enc(k2, TS || ((Prefix3 || MoveToken))),

where k2 is again a long-term producer key. Note that it may be the case that k1 = k2 (see above), though this is not required.

With this SessionID and ResumptionCookie, the consumer then resumes a session by providing both the SessionID and ResumptionCookie to the producer. This is done to prove to the producer that the consumer who knows the SessionID is also in possession of the correct ResumptionCookie. The producer verifies this by computing

(TS || ((Prefix3 || MoveToken))) = Dec(k2, ResumptionCookie)

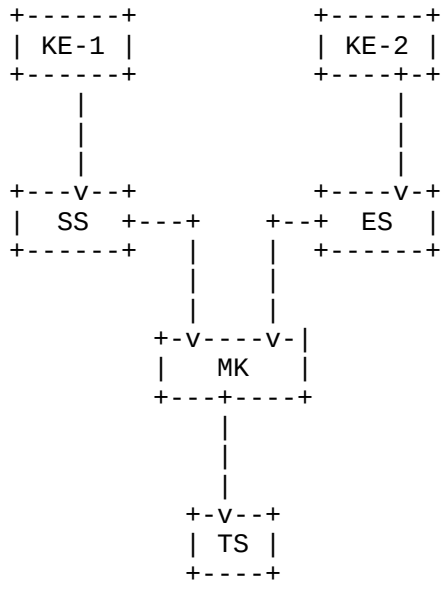
and checking the following equality

SessionID = Enc(k1, H(TS || (Prefix3)))

If equality holds, the producer uses the TS recovered from ResumptionCookie to re-initialize the previous session with the consumer.

9.4. Key Derivation

CCNxKE adopts the key schedule and derivation techniques defined in TLS 1.3 [TLS13]. Specifically, it uses the SS and ES to establish a common master secret (MS) and, from that, the traffic secret (TS). These dependencies are shown below.



In this figure, KE-1 and KE-2 are two "sources" of keying material. The following table shows what these two sources are in different key exchange scenarios.

Key Exchange	KE-1	KE-2
Full handshake	ClientKeyShare and ServerKeyShare DH	ClientKeyShare and ServerKeyShare DH
Handshake with 1-RTT	ClientKeyShare and ServerConfiguration public share DH	ClientKeyShare and ServerKeyShare DH
PSK	Pre-shared key	Pre-shared key

Given the values for SS and ES, the remaining derivation steps are below as defined in [TLS13]. They are repeated here for posterity.

1. xSS = HKDF-Extract(0, SS). Note that HKDF-Extract always produces a value the same length as the underlying hash function.

2. xES = HKDF-Extract(0, ES)

3. mSS = HKDF-Expand-Label(xSS, "expanded static secret", handshake_hash, L)

4. mES = HKDF-Expand-Label(xES, "expanded ephemeral secret", handshake_hash, L)

5. master_secret = HKDF-Extract(mSS, mES)

6. traffic_secret_0 = HKDF-Expand-Label(master_secret, "traffic secret", handshake_hash, L)

In all computations, the value "handshake_hash" is defined as the SHA256 hash digest of all CCNxKE messages contained up to the point of derivation. More details are given in Section 7.3.1 of [TLS13].

Updating the traffic secret using the re-key message (defined later) increments traffic_secret_N to traffic_secret_(N+1). This update procedure works as follows:

traffic_secret_N+1 = HKDF-Expand-Label(traffic_secret_N, "traffic secret", "", L)

9.5. Secret Generation and Lifecycle

The secrets (keys and IVs) used to encrypt and authenticate traffic are derived from the traffic secret. The explicit derivation formula, as is defined in [TLS13], is as follows:

secret = HKDF-Expand-Label(Secret, phase + ", " + purpose, handshake_context, key_length)

In this context, secret can be a key or IV. This formula is used when deriving keys based on a non-forward-secure SS and the forward-secure TS. The following table enumerates the values for "phase", and "handshake_context" to be used when defining keys for different purposes.

Record Type	Secret	Phase	Handshake Context
1-RTT Handshake	xSS	"early handshake key expansion"	HELLO + ServerConfiguration + Server Certificate
1-RTT Data	xSS	"early application data key expansion"	HELLO + ServerConfiguration + Server Certificate
Application Data	TS	"application data key expansion"	HELLO ... Finished

Moreover, the following table indicates the values of "purpose" used in the generation of each secret.

Secret	Purpose
Client Write Key	"client write key"
Server Write Key	"server write key"
Client Write IV	"client write IV"
Server Write IV	"server write IV"

((TODO: should we add examples for each of the above variants?))

10. Re-Key Message

Either the client and server can trigger a key update by sending an Interest or Content Object with a KEPayload field containing the flag KeyUpdate. The KEPayload will be encrypted by the traffic key. Upon receipt, the recipient MUST update the traffic secret as defined above and re-compute the traffic encryption and authentication keys. The previous traffic key must be securely discarded.

11. Application Data Protocol

Once traffic keys and the associated IVs are derived from the CCNxKE protocol, all subsequent Interest and Content Object messages are encrypted. Packet encryption uses the TLV encapsulation mechanism specified in [TLVENCAP]. For Interest encryption, the Salt in

[TLVENCAP] is set to the packet sequence number. The same substitution is done for Content Object encryption. Similarly, the KeyId field is substituted with the SessionID derived by the CCNxKE protocol. Packet sequence numbers are 64-bit numbers initialized to 0 when after the traffic secret is calculated. Each message increments and uses the sequence number when sending a new datagram (Interest). The sequence number for an Interest matches that of the Content Object response.

12. Security Considerations

For CCNxKE to be able to provide a secure connection, both the consumer and producer systems, keys, and applications must be secure. In addition, the implementation must be free of security errors.

The system is only as strong as the weakest key exchange and authentication algorithm supported, and only trustworthy cryptographic functions should be used. Short public keys and anonymous servers should be used with great caution. Implementations and users must be careful when deciding which certificates and certificate authorities are acceptable; a dishonest certificate authority can do tremendous damage.

13. References

13.1. Normative References

[CCNxMessages] Mosko, M. and I. Solis, "CCNx Messages in TLV Format", January 2016, <https://tools.ietf.org/html/draft-irtf-icnrg-ccnxmessages-01>.

[DH] Diffie, W. and M. Hellman, "New Directions in Cryptography", IEEE Transactions on Information Theory, V.IT-22 n.6 , June 1977.

[DTLS12] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", January 2012, <https://tools.ietf.org/html/rfc6347>.

[ECDSA] American National Standards Institute, "Public Key Cryptography for the Financial Services Industry: The Elliptic Curve Digital Signature Algorithm (ECDSA)", ANSI ANS X9.62-2005, November 2005.

[GCM] Dworkin, M., "Recommendation for Block Cipher Modes of Operation: Galois/Counter Mode (GCM) and GMAC", NIST Special Publication 800-38D, November 2007.

[QUIC] Iyengar, J. and I. Swett, "QUIC: A UDP-Based Secure and Reliable Transport for HTTP/2", December 2015.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", BCP 14, RFC 2119, DOI 10.17487/RFC2119, March 1997, <http://www.rfc-editor.org/info/rfc2119>.

[RFC2631] Rescorla, E., "Diffie-Hellman Key Agreement Method", RFC 2631, DOI 10.17487/RFC2631, June 1999, <http://www.rfc-editor.org/info/rfc2631>.

[RFC4086] Eastlake 3rd, D., Schiller, J., and S. Crocker, "Randomness Requirements for Security", BCP 106, RFC 4086, DOI 10.17487/RFC4086, June 2005, <http://www.rfc-editor.org/info/rfc4086>.

[RFC4302] Kent, S., "IP Authentication Header", RFC 4302, DOI 10.17487/RFC4302, December 2005, <http://www.rfc-editor.org/info/rfc4302>.

[RFC4303] Kent, S., "IP Encapsulating Security Payload (ESP)", RFC 4303, DOI 10.17487/RFC4303, December 2005, <http://www.rfc-editor.org/info/rfc4303>.

[RFC4987] Eddy, W., "TCP SYN Flooding Attacks and Common Mitigations", RFC 4987, DOI 10.17487/RFC4987, August 2007, <http://www.rfc-editor.org/info/rfc4987>.

[RFC5869] Krawczyk, H. and P. Eronen, "HMAC-based Extract-and-Expand Key Derivation Function (HKDF)", RFC 5869, DOI 10.17487/RFC5869, May 2010, <http://www.rfc-editor.org/info/rfc5869>.

[RFC6066] Eastlake 3rd, D., "Transport Layer Security (TLS) Extensions: Extension Definitions", RFC 6066, DOI 10.17487/RFC6066, January 2011, <http://www.rfc-editor.org/info/rfc6066>.

[RFC6347] Rescorla, E. and N. Modadugu, "Datagram Transport Layer Security Version 1.2", RFC 6347, DOI 10.17487/RFC6347, January 2012, <http://www.rfc-editor.org/info/rfc6347>.

[RFC6479] Zhang, X. and T. Tsou, "IPsec Anti-Replay Algorithm without Bit Shifting", RFC 6479, DOI 10.17487/RFC6479, January 2012, <http://www.rfc-editor.org/info/rfc6479>.

[RSA] Rivest, R., Shamir, A., and L. Adleman, "A Method for Obtaining Digital Signatures and Public-Key Cryptosystems", Communications of the ACM v. 21, n. 2, pp. 120-126., February 1978.

[SALSA20] Bernstein, D., "Salsa20 specification", www.http://cr.yp.to/snuffle/spec.pdf , April 2005.

[TLS13] Rescorla, E., "The Transport Layer Security (TLS) Protocol Version 1.3", December 2015, <https://tools.ietf.org/html/draft-ietf-tls-tls13-11>.

[TLVENCAP] Mosko, M. and C. Wood, "CCNx Packet Encapsulation", n.d., <https://github.com/PARC/ccnx-tlvencap-rfc>.

13.2. Informative References

[HASHCHAIN] L. Lamport, "Password Authentication with Insecure Communication", ANSI Communications of the ACM 24.11, pp 770-772, November 1981.

[RFC5077] Salowey, J., Zhou, H., Eronen, P., and H. Tschofenig, "Transport Layer Security (TLS) Session Resumption without Server-Side State", RFC 5077, DOI 10.17487/RFC5077, January 2008, <http://www.rfc-editor.org/info/rfc5077>.

Authors' Addresses

M. Mosko
PARC

EMail: marc.mosko@parc.com

Ersin Uzun
PARC

EMail: ersin.uzun@parc.com

Christopher A. Wood
PARC

EMail: christopher.wood@parc.com

ICN Research Group
Internet-Draft
Intended status: Informational
Expires: October 24, 2016

Y. Zhang
D. Raychadhuri
WINLAB, Rutgers University
L. Grieco
Politecnico di Bari (DEI)
E. Baccelli
INRIA
J. Burke
UCLA REMAP
R. Ravindran
G. Wang
Huawei Technologies
A. Lindren
B. Ahlgren
SICS Swedish ICT
O. Schelen
Lulea University of Technology

April 22, 2016

Requirements and Challenges for IoT over ICN
draft-zhang-icnrg-icniot-requirements-01

Abstract

The Internet of Things (IoT) promises to connect billions of objects to the Internet. After deploying many stand-alone IoT systems in different domains, the current trend is to develop a common, "thin waist" of protocols forming a horizontal unified, defragmented IoT platform. Such a platform will make objects accessible to applications across organizations and domains. Towards this goal, quite a few proposals have been made to build a unified host-centric IoT platform as an overlay on top of today’s host-centric Internet. However, there is a fundamental mismatch between the host-centric nature of todays Internet and the information-centric nature of the IoT system. To address this mismatch, we propose to build a common set of protocols and services, which form an IoT platform, based on the Information Centric Network (ICN) architecture, which we call ICN-IoT. ICN-IoT leverages the salient features of ICN, and thus provides seamless mobility support, security, scalability, and efficient content and service delivery.

This draft describes representative IoT requirements and ICN challenges to realize a unified ICN-IoT framework. Towards this, we first identify a list of important requirements which a unified IoT architecture should have to support tens of billions of objects, then we discuss how the current IP-IoT overlay fails to meet these requirements, followed by discussion on suitability of ICN for IoT.

Though we see most of the IoT requirements can be met by ICN, we discuss specific challenges ICN has to address to satisfy them. Then we provide discussion of popular IoT scenarios including the "smart" home, campus, grid, transportation infrastructure, healthcare, Education, and Entertainment for completeness, as specific scenarios requires appropriate design choices and architectural considerations towards developing an ICN-IoT solution.

Status of This Memo

Copyright Notice

Table of Contents

1. IoT Motivation

During the past decade, many standalone Internet of Things (IoT) systems have been developed and deployed in different domains. The recent trend, however, is to evolve towards a globally unified IoT platform, in which billions of objects connect to the Internet, available for interactions among themselves, as well as interactions with many different applications across boundaries of administration and domains. Building a unified IoT platform, however, poses great challenges on the underlying network and systems. To name a few, it needs to support 50-100 Billion networked objects [1], many of which are mobile. The objects will have extremely heterogeneous means of

connecting to the Internet, often with severe resource constraints. Interactions between the applications and objects are often real-time and dynamic, requiring strong security and privacy protections. In addition, IoT applications are inherently information centric (e.g., data consumers usually need data sensed from the environment without any reference to the sub-set of motes that will provide the asked information). Taking a general IoT perspective, we first discuss the IoT requirements generally applicable to many well known scenarios. We then discuss how the current IP overlay models fail to meet these requirements. We follow this by key ICN features that makes it a better candidate to realize an unified IoT framework. We then discuss IoT challenges from an ICN perspective and requirements posed towards its design. Final discussion focuses on IoT scenarios and their unique challenges.

2. IoT Architectural Requirements

A unified IoT platform has to support interactions among a large number of mobile devices across the boundaries of organizations and domains. As a result, it naturally poses stringent requirements in every aspect of the system design. Below, we outline a few important requirements that a unified IoT platform has to address.

2.1. Naming

The first step towards realizing a unified IoT platform is the ability to assign names that are unique within the scope and lifetime of each device, data items generated by these devices, or a group of devices towards a common objective. Naming has the following requirements: first, names need to be persistent (within one or more contexts) against dynamic features that are common in IoT systems, such as lifetime, mobility or migration; second, names need to be secure based on application requirements; third, names should provide advantages to application authors in comparison with traditional host address based schemes.

2.2. Scalability

Cisco predicts there will be around 50 Billion IoT devices such as sensors, RFID tags, and actuators, on the Internet by 2020 [1]. As mentioned above, a unified IoT platform needs to name every entity such as data, device, service etc. Scalability has to be addressed at multiple levels of the IoT architecture including naming, security, name resolution, routing and forwarding level. In addition, mobility adds further challenge in terms of scalability. Particularly with respect to name resolution the system should be able to register/update/resolve a name within a short latency.

2.3. Resource Constraints

IoT devices can be broadly classified into two groups: resource-sufficient and resource-constrained. In general, there are the following types of resources: power, computing, storage, bandwidth, and user interface.

Power constraints of IoT devices limit how much data these devices can communicate, as it has been shown that communications consume more power than other activities for embedded devices. Flexible techniques to collect the relevant information are required, and uploading every single produced data to a central server is undesirable. Computing constraints limit the type and amount of processing these devices can perform. As a result, more complex processing needs to be conducted in cloud servers or at opportunistic points, example at the network edge, hence it is important to balance local computation versus communication cost.

Storage constraints of the IoT devices limit the amount of data that can be stored on the devices. This constraint means that unused sensor data may need to be discarded or stored in aggregated compact form time to time. Bandwidth constraints of the IoT devices limit the amount of communication. Such devices will have the same implication on the system architecture as with the power constraints; namely, we cannot afford to collect single sensor data generated by the device and/or use complex signaling protocols.

User interface constraints refer to whether the device is itself capable of directly interacting with a user should the need arise (e.g., via a display and keypad or LED indicators) or requires the network connectivity, either global or local, to interact with humans.

The above discussed device constraints also affect application performance with respect to latency and jitter. This in particular applies to satellite or other space based devices.

2.4. Traffic Characteristics

IoT traffic can be broadly classified into local area traffic and wide area traffic. Local area traffic is between nearby devices. For example, neighboring cars may work together to detect potential hazards on the highway, sensors deployed in the same room may collaborate to determine how to adjust the heating level in the room. These local area communications often involve data aggregation and filtering, have real time constraints, and require fast device/data/service discovery and association. At the same time, the IoT platform has to also support wide area communications. For example,

in Intelligent Transportation Systems, re-routing operations may require a broad knowledge of the status of the system, traffic load, availability of freights, whether forecasts and so on. Wide area communications require efficient data/service discovery and resolution services.

While traffic characteristics for different IoT systems are expected to be different, certain IoT systems have been analyzed and shown to have comparable uplink and downlink traffic volume in some applications such as [2], which means that we have to optimize the bandwidth/energy consumption in both directions. Further, IoT traffic demonstrates certain periodicity and burstiness [2]. As a result, when provisioning the system, the shape of the traffic volume has to be properly accounted for.

2.5. Contextual Communication

Many IoT applications shall rely on dynamic contexts in the IoT system to initiate communication between IoT devices. Here, we refer to a context as attributes applicable to a group of devices that share some common features, such as their owners may have a certain social relationship or belong to the same administrative group, or the devices may be present in the same location. For example, cars traveling on the highway may form a "cluster" based upon their temporal physical proximity as well as the detection of the same event. These temporary groups are referred to as contexts. IoT applications need to support interactions among the members of a context, as well as interactions across contexts.

Temporal context can be broadly categorized into two classes, long-term contexts such as those that are based upon social contacts as well as stationary physical locations (e.g., sensors in a car/ building), and short-term contexts such as those that are based upon temporary proximity (e.g., all taxicabs within half a mile of the Time Square at noon on Oct 1, 2013). Between these two classes, short-term contexts are more challenging to support, requiring fast formation, update, lookup and association.

2.6. Handling Mobility

There are several degrees of mobility in a unified IoT platform, ranging from static as in fixed assets to highly dynamic in vehicle-to-vehicle environments.

Mobility in the IoT platform can mean 1) the data producer mobility (i.e., location change), 2) the data consumer mobility, 3) IoT Network mobility (e.g., a body-area network in motion as a person is walking); and 4) disconnection between the data source and

destination pair (e.g., due to unreliable wireless links). The requirement on mobility support is to be able to deliver IoT data below an application's acceptable delay constraint in all of the above cases, and and if necessary to negotiate different connectivity or security constraints specific to each mobile context.

2.7. Storage and Caching

Storage and caching plays a very significant role depending on the type of IoT ecosystem, also a function subjected to privacy and security guidelines. In a unified IoT platform, depending on application requirements, content caching may or may not be policy driven. If caching is pervasive, intermediate nodes don't need to always forward a content request to its original creator; rather, locating and receiving a cached copy is sufficient for IoT applications. This optimization can greatly reduce the content access latencies.

Furthermore considering hierarchical nature of IoT systems, ICN architectures enable a more flexible, heterogeneous and potentially fault-tolerant approach to storage providing persistence at multiple levels.

In network storage and caching, however, has the following requirements on the IoT platform. The platform needs to support the efficient resolution of cached copies. Further the platform should strive for the balance between caching, content security/privacy, and regulations.

2.8. Security and Privacy

In addition to the fundamental challenge of trust management, a variety of security and privacy concerns also exist in ICNs.

The unified IoT platform makes physical objects accessible to applications across organizations and domains. Further, it often integrates with critical infrastructure and industrial systems with life safety implications, bringing with it significant security challenges and regulatory requirements [11].

Security and privacy thus become a serious concern, as does the flexibility and usability of the design approaches. Beyond the overarching trust management challenge, security includes data integrity, authentication, and access control at different layers of the IoT platform. Privacy means that both the content and the context around IoT data need to be protected. These requirements will be driven by various stake holders such as industry, government, consumers etc.

2.9. Communication Reliability

IoT applications can be broadly categorized into mission critical and non-mission critical. For mission critical applications, reliable communication is one of the most important features as these applications have strong QoS requirements. Reliable communication requires the following capabilities for the underlying system: (1) seamless mobility support in the face of extreme disruptions (DTN), (2) efficient routing in the presence of intermittent disconnection, (3) QoS aware routing, (4) support for redundancy at all levels of a system (device, service, network, storage etc.), and (5) support for rich communication patterns (unlike the tree-like routing structure supported by RPL developed by ROLL WG).

2.10. Self-Organization

The unified IoT platform should be able to self-organize to meet various application requirements, especially the capability to quickly discover heterogeneous and relevant (local or global) devices/data/services based on the context. This discovery can be achieved through an efficient platform-wide publish-subscribe service, or through private community grouping/clustering based upon trust and other security requirements. In the former case, the publish-subscribe service must be efficiently implemented, able to support seamless mobility, in- network caching, name-based routing, etc. In the latter case, the IoT platform needs to discover the private community groups/clusters efficiently.

Another aspect of self-organization is decoupling the sensing Infrastructure from applications. In a unified IoT platform, various applications run on top of a vast number of IoT devices. Upgrading the firmware of the IoT devices is a difficult work. It is also not practical to reprogram the IoT devices to accommodate every change of the applications. The infrastructure and the application specific logics need to be decoupled. A common interface is required to dynamically configure the interactions between the IoT devices and easily modify the application logics on top of the sensing infrastructure [23] [24].

2.11. Ad hoc and Infrastructure Mode

Depending upon whether there is communication infrastructure, an IoT system can operate either in ad-hoc or infrastructure mode.

For example, a vehicle may determine to report its location and status information to a server periodically through cellular connection, or, a group of vehicles may form an ad-hoc network that collectively detect road conditions around them. In the cases where

infrastructure is unavailable, one of the participating nodes may choose to become the temporary gateway.

The unified IoT platform needs to design a common protocol that serves both modes. Such a protocol should be able to provide: (1) energy-efficient topology discovery and data forwarding in the ad-hoc mode, and (2) scalable name resolution in the infrastructure mode.

2.12. Open API

General IoT applications involve sensing, processing, and secure content distribution occurring at various timescales and at multiple levels of hierarchy depending on the application requirements. This requires open APIs to be generic enough to support commonly used interactions between consumers, content producer, and IoT services, as opposed to proprietary APIs that are common in today's systems. Examples include pull, push, and publish/subscribe mechanisms using common naming, payload, encryption and signature schemes.

2.13. IoT Platform Management

An IoT platforms' service, control, and data plane will be governed by its own management infrastructure which includes distributed and centralized middleware, discovery, naming, self-configuring, analytic functions, and information dissemination to achieve specific IoT system objectives [18][19][20]. Towards this new IoT management mechanisms and service metrics need to be developed to measure the success of an IoTdeployment. Considering an IoT systems' defining characteristics such as, its potential large number of IoT devices, ephemeral nature to save power, mobility, and ad hoc communication, autonomic self-management mechanisms become very critical. Further considering its hierarchical information processing deployment model, the platform needs to orchestrate computational tasks according to the involved sensors and the available computation resources which may change over time. An efficient computation resource discovery and management protocol is required to facilitate this process. The trade-off between information transmission and processing is another challenge.

3. State of the Art

Over the years, many stand-alone IoT systems have been deployed in various domains. These systems usually adopt a vertical silo architecture and support a small set of pre-designated applications. A recent trend, however, is to move away from this approach, towards a unified IoT platform in which the existing silo IoT systems, as well as new systems that are rapidly deployed. This will make their data and services accessible to general Internet applications (as in

ETSI- M2M and oneM2M standards). In such a unified platform, resources can be accessed over Internet and shared across the physical boundaries of the enterprise. However, current approaches to achieve this objective are based upon Internet overlays, whose inherent inefficiencies due to IP protocol [8] hinders the platform from satisfying the IoT requirements outlined earlier (particularly in terms of scalability, security, mobility, and self-organization)

3.1. Silo IoT Architecture

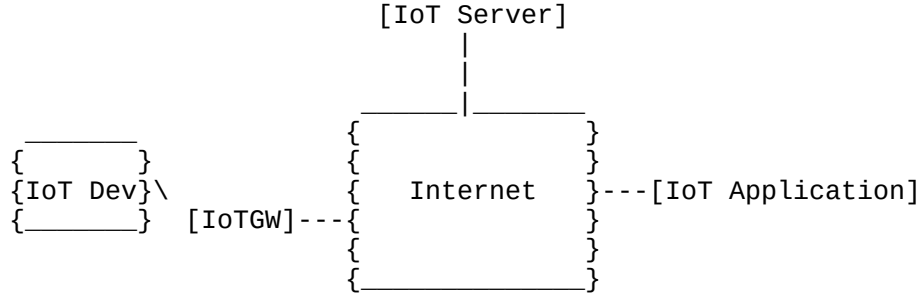


Figure 1:Silo architecture of standalone IoT systems

A typical standalone IoT system is illustrated in Figure 1, which includes devices, a gateway, a server and applications. Many IoT devices have limited power and computing resources, unable to directly run normal IP access network (Ethernet, WIFI, 3G/LTE etc.) protocols. Therefore they use the IoT gateway to the server. Through the IoT server, applications can subscribe to data collected by devices, or interact with devices.

There have been quite a few popular protocols for standalone IoT systems, such as DF-1, MelsecNet, Honeywell SDS, BACnet, etc. However, these protocols are operating at the device-level abstraction, instead of information driven, leading to a highly fragmented protocol space with limited interoperability.

3.2. Overlay Based Unified IoT Solutions

The current approach to a unified IoT platform is to make IoT gateways and servers adopt standard APIs. IoT devices connect to the Internet through the standard APIs and IoT applications subscribe and receive data through standard control and data APIs. Building on top of today’s Internet as an overlay, this is the most practical approach towards a unified IoT platform. There are ongoing

standardization efforts including ETSI[3], oneM2M[4]. Network operators can use frameworks to build common IOT gateways and servers for their customers. In addition, IETF's CORE working group [5] is developing a set of protocols like CoAP (Constrained Application Protocol) [49], that is a lightweight protocol modeled after HTTP [50] and adapted specifically for the Internet of Things (IoT). CoAP adopts the Representational State Transfer (REST) architecture with Client-Server interactions. It uses UDP as the underlying transport protocol with reliability and multicast support. Both CoAP and HTTP are considered as the suitable application level protocols for Machine-to-Machine communications, as well as IoT. For example, oneM2M (which is one of leading standards for unified M2M platform) has both the protocol bindings to HTTP and CoAP for its primitives. Figure 2 shows the architecture adopted in this approach.

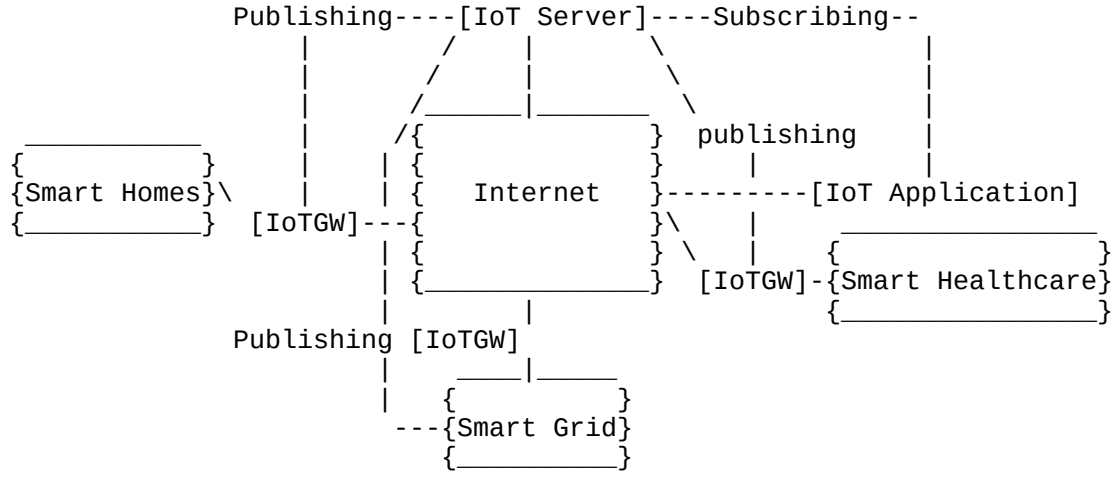


Figure 2: Implementing an open IoT platform through standarized APIs on the IoT gateways and the server

3.2.1. Weaknesses of the Overlay-based Approach

The above overlay-based approach can work with many different protocols, but the system is built upon today's IP network, which has inherent weaknesses towards supporting a unified IoT system. As a result, it cannot satisfy some of the requirements we outlined in Section 2:

- Naming. In current overlays for IoT systems the naming scheme is host centric, i.e., the name of a given resource/service is linked

to the one of device that can provide it. In turn, device names are coupled to IP addresses, which are not persistent in mobile scenarios. On the other side, in IoT systems the same service/ resource could be provided by many different devices thus requiring a different design rationale.

- Trust. Trust management schemes are still relatively weak, focusing on securing communication channels rather than managing the data that needs to be secured directly.

- Mobility. The overlay-based approach uses IP addresses as names at the network layer, which hinders the support for device/service mobility or flexible name resolution. Further the Layer 2/3 management, and application-layer addressing and forwarding required to deploy current IoT solutions limit the scalability and management of these systems.

- Resource constraints. The overlay-based approach requires every device to send data to an aggregator or to the IoT server. Resource constraints of the IoT devices, especially in power and bandwidth, could seriously limit the performance of this approach.

- Traffic Characteristics. In this approach, applications are written in a host-centric manner suitable for point-to-point communication. IoT requires multicast support that is challenging in overlay systems today.

- Contextual Communications. This overlay-based approach cannot react to dynamic contextual changes in a timely fashion. The main reason is that context lists are kept at the IoT server in this approach, and they cannot help efficiently route requests information at the network layer.

- Storage and Caching. The overlay-based approach supports application-centric storage and caching but not what ICN envisions at the network layer, or flexible storage enabled via name-based routing or name-based lookup.

- Self-Organization. The overlay-based approach is topology-based as it is bound to IP semantics, and thus does not sufficiently satisfy the self-organization requirement. In addition to topological self-organization, IoT also requires data- and service-level self-organization [59], which is not supported by the overlay approach.

- Ad-hoc and infrastructure mode. As mentioned above, the overlay-based approach lacks self-organization, and thus does not provide efficient support for the ad-hoc mode.

4. Advantages of using ICN for IoT

A key concept of ICN is the ability to name data independently from the current location at which it is stored, which simplifies caching and enables decoupling of sender and receiver. Using ICN to design an architecture for IoT data potentially provides such advantages compared to using traditional host-centric networks. This section highlights general benefits that ICN could provide to IoT networks.

- Naming of Devices, Data and Services. The heterogeneity of both network equipment deployed and services offered by IoT networks leads to a large variety of data, services and devices. While using a traditional host-centric architecture, only devices or their network interfaces are named at the network level, leaving to the application layer the task to name data and services. In many common applications of IoT networks, data and services are the main goal, and specific communication between two devices is secondary. The network distributes content and provides a service, instead of establishing a communication link between two devices. In this context, data content and services can be provided by several devices, or group of devices, hence naming data and services is often more important than naming the devices. This naming mechanism also enables self-configuration of the IoT system.

- Distributed Caching and Processing. While caching mechanisms are already used by other types of overlay networks, IoT networks can potentially benefit even more from caching and in-network processing systems, because of their resource constraints. Wireless bandwidth and power supply can be limited for multiple devices sharing a communication channel, and for small mobile devices powered by batteries. In this case, avoiding unnecessary transmissions with IoT devices to retrieve and distribute IoT data to multiple places is important, hence processing and storing such content in the network can save wireless bandwidth and battery power. Moreover, as for other types of networks, applications for IoT networks requiring shorter delays can benefit from local caches and services to reduce delays between content request and delivery.

- Decoupling between Sender and Receiver. IoT devices may be mobile and face intermittent network connectivity. When specific data is requested, such data can often be delivered by ICN without any consistent direct connectivity between devices. Apart from using structured caching systems as described previously, information can also be spread by forwarding data opportunistically.

5. ICN Challenges for IoT

This section outlines some of the ICN specific challenges [71] that must be considered when defining an IoT framework over ICN, and describes some of the trade offs that will be involved.

ICN integrates content/service/host abstraction, name-based routing, compute, caching/storage as part of the network infrastructure connecting consumers and services which meets most of the requirements discussed above; however IoT requires special considerations given heterogeneity of devices and interfaces such as for constrained networking [38][70], data processing, and content distribution models to meet specific application requirements which we identify as challenges in this section.

5.1. Naming Devices, Data, and Services

The ICN approach of named data and services (i.e., device independent naming) is typically desirable when retrieving IoT data. However, data centric naming may also pose challenges.

- Naming of devices: Naming devices is often important in an IoT network. The presence of actuators requires clients to act specifically on a device, e.g. to switch it on or off. Also, managing and monitoring the devices for administration purposes requires devices to have a specific name allowing to identify them uniquely. There are multiple ways to achieve device naming, even in systems that are data centric by nature. For example, in systems that are addressable or searchable based on metadata or sensor content, the device identifier can be included as a special kind of metadata or sensor reading.

- Size of data/service name: In information centric applications, the size of the data is typically larger than its name. For the IoT, sensors and actuators are very common, and they can generate or use data as small as a short integer containing a temperature value, or a one-byte instruction to switch off an actuator. The name of the content for each of these pieces of data has to uniquely identify the content. For this reason, many existing naming schemes have long names that are likely to be longer than the actual data content for many types of IoT applications. Furthermore, naming schemes that have self certifying properties (e.g., by creating the name based on a hash of the content), suffer from the problem that the object can only be requested when the object has been created and the content is already known, thus requiring some form of indexing service. While this is an acceptable overhead for larger data objects, it is infeasible for use when the object size is on the order of a few bytes.

- Hash-based content name: Hash algorithms are commonly used to name content in order to verify that the content is the one requested. This is only possible in contexts where the requested object is already existing, and where there is a directory service to look up names. This approach is suitable for systems with large data objects where it is important to verify the content.

- Metadata-based content name: Relying on metadata allows to generate a name for an object before it is created. However this mechanism requires metadata matching semantics.

- Naming of services: Similarly to naming of devices or data, services can be referred to with a unique identifier, provided by a specific device or by someone assigned by a central authority as the service provider. It can however also be a service provided by anyone meeting some certain metadata conditions. Example of services include content retrieval, that takes a content name/ description as input and returns the value of that content, and actuation, that takes an actuation command as input and possibly returns a status code afterwards.

- Trust: We need to ensure the name of a network element is issued by a trustworthy issuer in the context of the application, such as a trusted organization in [44]. Further the validity of each piece of data published by an authorized entity in the namespace should be verifiable - e.g., by following a hierarchical chain-of-trust to a root that is acceptable for the application. See [54]s for an example.

- Flexibility: Further challenges arise for hierarchical naming schema: referring to requirements on "constructible names" and "on-demand publishing" [28][29]. The former entails that each user is able to construct the name of a desired data item through specific algorithms and that it is possible to retrieve information also using partially specified names. The latter refers the possibility to request a content that has not yet been published in the past, thus triggering its creation.

- Control/scoping : Some information could be accessible only within a given scope. This challenge is very relevant for smart home and health monitoring applications, where privacy issues play a key role and the local scope of a home or healthcare environment may be well-defined. However, perimeter- and channel-based access control is often violated in current networks to enable over-the-wire updates and cloud-based services, so scoping is unlikely to replace a need for data-centric security in ICN.

- Confidentiality: As names can reveal information about the nature of the communication, mechanisms for name confidentiality should be available in the ICN-IoT architecture.

5.2. Name Resolution

Inter-connecting numerous IoT entities, as well as establishing reachability to them, requires a scalable name resolution system considering several dynamic factors like mobility of end points, service replication, in-network caching, failure or migration [37] [40] [41] [57]. The objective is to achieve scalable name resolution handling static and dynamic ICN entities with low complexity and control overhead. In particular, the main requirements/challenges of a name space (and the corresponding Name Resolution System where necessary) are [31] [33]:

- Scalability: The first challenge faced by ICN-IoT name resolution system is its scalability. Firstly, the approach has to support billions of objects and devices that are connected to the Internet, many of which are crossing administrative domain boundaries. Second of all, in addition to objects/devices, the name resolution system is also responsible for mapping IoT services to their network addresses. Many of these services are based upon contexts, hence dynamically changing, as pointed out in [37]. As a result, the name resolution should be able to scale gracefully to cover a large number of names/services with wide variations (e.g., hierarchical names, flat names, names with limited scope, etc.). Notice that, if hierarchical names are used, scalability can be also supported by leveraging the inherent aggregation capabilities of the hierarchy. Advanced techniques such as hyperbolic routing [53] may offer further scalability and efficiency.

- Deployability and interoperability: Graceful deployability and interoperability with existing platforms is a must to ensure a naming schema to gain success on the market [7]. As a matter of fact, besides the need to ensure coexistence between IP-centric and ICN-IoT systems, it is required to make different ICN-IoT realms, each one based on a different ICN architecture, to interoperate.

- Latency: For real-time or delay sensitive M2M application, the name resolution should not affect the overall QoS. With reference to this issue it becomes important to circumvent too centralized resolution schema (whatever the naming style, i.e, hierarchical or flat) by enforcing in-network cooperation among the different entities of the ICN-IoT system, when possible [58]. In addition, fast name lookup are necessary to ensure soft/hard real time

services [60][61][62]. This challenge is especially important for applications with stringent latency requirements, such as health monitoring, emergency handling and smart transportation [63].

- Locality and network efficiency: During name resolution the named entities closer to the consumer should be easily accessible (subject to the application requirements). This requirement is true in general because, whatever the network, if the edges are able to satisfy the requests of their consumers, the load of the core and content seek time decrease, and the overall system scalability is improved. This facet gains further relevance in those domains where an actuation on the environment has to be executed, based on the feedbacks of the ICN-IoT system, such as in robotics applications, smart grids, and industrial plants [59].

- Agility: Some data items could disappear while some other ones are created so that the name resolution system should be able to effectively take care of these dynamic conditions. In particular, this challenge applies to very dynamic scenarios (e.g., VANETs) in which data items can be tightly coupled to nodes that can appear and disappear very frequently.

5.3. Caching/Storage

In-network caching helps bring data closer to consumers, but its usage differs in constrained and infrastructure part of the IoT network. Caching in constrained networks is limited to small amounts in the order of 10KB, while caching in infrastructure part of the network can allow much larger chunks.

Caching in ICN-IoT faces several challenges:

- The main challenge is to determine which nodes on the routing path should cache the data. According to [33], caching the data on a subset of nodes can achieve a better gain than caching on every en-route routers. In particular, the authors propose a "selective caching" scheme to locate those routers with better hit probabilities to cache data. According to [34], selecting a random router to cache data is as good as caching the content everywhere. In [55], the authors suggest that edge caching provides most of the benefits of in-network caching typically discussed in NDN, with simpler deployment. However, it and other papers consider workloads that are analogous to today’s CDNs, not the IoT applications considered here. Further work is likely required to understand the appropriate caching approach for IoT applications.

- Another challenge in ICN-IoT caching is what to cache for IoT applications. In many IoT applications, customers often access a stream of sensor data, and as a result, caching a particular sensor data item may not be beneficial. In [36], the authors suggest to cache IoT services on intermediate routers, and in [37], the authors suggest to cache control information such as pub/sub lists on intermediate nodes. In addition, it is yet unclear what caching means in the context of actuation in an IoT system. For example, it could mean caching the result of a previous actuation request (using other ICN mechanisms to suppress repeated actuation requests within a given time period), or have little meaning at all if actuation uses authenticated requests as in [56].

- Another challenge is that the efficiency of Distributed Caching may be application dependent. When content popularity is heterogeneous, some content is often requested repeatedly. In that case, the network can benefit from caching. Another case where caching would be beneficial is when devices with low duty cycle are present in the network and when access to the cloud infrastructure is limited. However, using distributed caching mechanisms in the network is not useful when each object is only requested at most once, as a cache hit can only occur for the second request and later. It may also be less beneficial to have caches distributed throughout ICN nodes in cases when there are overlays of distributed repositories, e.g., a cloud or a Content Distribution Network (CDN), from which all clients can retrieve the data. Using ICN to retrieve data from such services may add some efficiency, but in case of dense occurrence of overlay CDN servers the additional benefit of caching in ICN nodes would be lower. Another example is when the name refers to an object with variable content/state. For example, when the last value for a sensor reading is requested or desired, the returned data should change every time the sensor reading is updated. In that case, ICN caching may increase the risk that cache inconsistencies result in old data being returned.

5.4. Routing and Forwarding

Routing in ICN-IoT differs from routing in traditional IP networks in that ICN routing is based upon names instead of locators. Broadly speaking, ICN routing can be categorized into the following two categories: direct name-based routing and indirect routing using a name resolution service (NRS).

- In direct name-based routing, packets are forwarded by the name of the data [57][38][42] or the name of the destination node [43]. Here, the main challenge is to keep the ICN router state required

to route/forward data low. This challenge becomes more serious when a flat naming scheme is used due to the lack of aggregation capabilities.

- In indirect routing, packets are forwarded based upon the locator of the destination node, and the locator is obtained through the name resolution service. In particular, the name-locator binding can be done either before routing (i.e., static binding) or during routing (i.e., dynamic binding). For static binding, the router state is the same as that in traditional routers, and the main challenge is the need to have fast name resolution, especially when the IoT nodes are mobile. For dynamic binding, ICN routers need to main a name-based routing table, hence the challenge of keeping the state information low. At the same time, the need of fast name resolution is also critical. Finally, another challenge is to quantify the cost associated with mobility management, especially static binding vs. dynamic binding.

During a network transaction, either the data producer or the consumer may move away and thus we need to handle the mobility to avoid information loss. ICN may differentiate mobility of a data consumer from that of a producer:

- When a consumer moves to a new location after sending out the request for Data, the Data may get lost, which requires the consumer to simply resend the request, a technique used by direct routing approach. Indirect routing approach doesn’t differentiate between consumer and producer mobility [57], also network caching can improve data recovery for this approach.

- If the data producer itself has moved, the challenge is to control the control overhead while searching for a new data producer (or for the same data producer in its new position). To this end, flooding techniques could be used, but an intra-domain level only, otherwise the network stability would be seriously impaired. For handling mobility across different domains, more sophisticated approaches could be used, including the adoption of a SDN-based control plane.

5.5. Contextual Communication

Contextualization through metadata in ICN control or application payload allows IoT applications to adapt to different environments. This enables intelligent networks which are self-configurable and enable intelligent networking among consumers and producers [36]. For example, let us look at the following smart transportation scenario: "James walks on NYC streets and wants to find an empty cab closest to his location." In this example, the context is the

relative locations of James and taxi drivers. A context service, as an IoT middleware, processes the contextual information and bridges the gap between raw sensor information and application requirements. Alternatively, naming conventions could be used to allow applications to request content in namespaces related to their local context without requiring a specific service, such as /local/geo/ mgrs/4QFJ/123/678 to retrieve objects published in the 100m grid area 4QFJ 123 678 of the military grid reference system (MGRS). In both cases, trust providers may emerge that can vouch for an application's local knowledge.

However, extracting contextual information on a real-time basis is very challenging:

- We need to have a fast context resolution service through which the involved IoT devices can continuously update its contextual information to the application (e.g., each taxi's location and Jame's information in the above example). Or, in the namespace driven approach, mechanisms for continuous nearest neighbor queries in the namespace need to be developed.

- The difficulty of this challenge grows rapidly when the number of devices involved in a context as well as the number of contexts increases.

5.6. In-network Computing

In-network computing enables ICN routers to host heterogeneous services catering to various network functions and applications needs. Contextual services for IoT networks require in-network computing, in which each sensor node or ICN router implements context reasoning [36]. Another major purpose of in-network computing is to filter and cleanse sensed data in IoT applications, that is critical as the data is noisy as is [44].

Named Function Networking [64] describes an extension of the ICN concept to named functions processed in the network, which could be used to generate data flow processing applications well-suited to, for example, time series data processing in IoT sensing applications. Related to this, is the need to support efficient function naming. Functions, input parameters, and the output result could be encapsulated in the packet header, the packet body, or mixture of the two (e.g. [24]). If functions are encapsulated in packet headers, the naming scheme affects how a computation task is routed in the network, which IoT devices are involved in the computation task (e.g. [35]), and how a name is decomposed into smaller computation tasks and deployed in the network for a better performance.

Another is challenge is related to support computing-aware routing. Normal routing is for forwarding requests to the nearest source or cache and return the data to the requester, whereas the routing for in-network computation has a different purpose. If the computation task is for aggregating sensed data, the routing strategy is to route the data to achieve a better aggregation performance [32].

In-network computing also includes synchronization challenges. Some computation tasks may need synchronizations between sub-tasks or IoT devices, e.g. a device may not send data as soon as it is available because waiting for data from the neighbours may lead to a better aggregation result; some devices may choose to sleep to save energy while waiting for the results from the neighbours; while aggregating the computation results along the path, the intermediate IoT devices may need to choose the results generated within a certain time window.

5.7. Security and Privacy

Security and privacy is crucial to all the IoT applications applications including the use cases discussed in Section 5. In one recent demonstration,it was shown that passive tire pressure sensors in cars could be hacked and used as a gateway into the automotive system [38]. The ICN paradigm is information-centric as opposed to state-of-the-art host-centric internet. Besides aspects like naming, content retrieval and caching this also has security implications. ICN advocates the model of trust in content rather than trust in network hosts. This brings in the concept of Object Security which is contrary to session-based security mechanisms such as TLS/DTLS prevalent in the current host-centric internet. Object Security is based on the idea of securing information objects unlike session-based security mechanisms which secure the communication channel between a pair of nodes. This reinforces an inherent characteristic of ICN networks i.e. to decouple senders and receivers. In the context of IoT, the Object Security model has several concrete advantages. Many IoT applications have data and services are the main goal and specific communication between two devices is secondary. Therefore, it makes more sense to secure IoT objects instead of securing the session between communicating endpoints. Though ICN includes data-centric security features the mechanisms have to be generic enough to satisfy multiplicity of policy requirements for different applications. Furthermore security and privacy concerns have to be dealt in a scenario-specific manner with respect to network function perspective spanning naming, name-resolution, routing, caching, and ICN-APIs. In general, we feel that security and privacy protection in IoT systems should mainly focus on the following aspects: confidentiality, integrity, authentication and non-repudiation, and availability.

Implementing security and privacy methods faces different challenges in the constrained and infrastructure part of the network.

- In the resource-constrained nodes, energy limitation is the biggest challenge. As an example, let us look at a typical sensor tag. Suppose the tag has a single 16-bit processor, often running at 6 MHz to save energy, with 512Bytes of RAM and 16KB of flash for program storage. Moreover, it has to deliver its data over a wireless link for at least 10,000 hours on a coin cell battery. As a result, traditional security/privacy measures are impossible to be implemented in the constrained part. In this case, one possible solution might be utilizing the physical wireless signals as security measures [46] [36].

- In the infrastructure part, we have several new threats introduced by ICN-IoT [52]:

  1. We need to ensure the name of a network element is issued by a trustworthy organization entity such as in [48], or by its trusted delegate. As name securely binds to data in ICN, security constraints of content that has not yet been published yet should also be taken into consideration.

  2. An intruder may gain access or gather information from a resource it is not entitled to. As a consequence, an adversary may examine, remove or even modify confidential information.

  3. An intruder may mimic an authorized user or network process. As a result, the intruder may forge signatures, or impersonate a source address.

  4. An adversary may manipulate the message exchange process between network entities. Such manipulation may involve replay, rerouting, mis-routing and deletion of messages.

  5. An intruder may insert fake/false sensor data into the network. The consequence might be an increase in delay and performance degradation for network services and applications.

5.8. Self-Orgnization

General IoT deployments involves heterogeneous IoT systems or subsystems within a particular scenario. Here scope-based self-organization is required to ensure logical isolation between the IoT subsystems, which should be enabled at different levels -- device/service discovery, naming, topology construction, routing over logical ICN topologies, and caching [69]. These challenges are

extended to constrained devices as well and they should be energy and device capability aware. In the infrastructure part, intelligent name-based routing, caching, in-network computing techniques should be studied to meet the scope-based self-configuration needs of ICN-IoT.

5.9. Communications Reliability

ICN offers many ingredients for reliable communication such as multi-home interest anycast over heterogeneous interfaces, caching, and forwarding intelligence for multi-path routing leveraging state-based forwarding in protocols like CCN/NDN. However these features have not been analyzed from the QoS perspective when heterogeneous traffic patterns are mixed in a router, in general QoS for ICN is an open area of research [71]. In-network reliability comes at the cost of a complex network layer; hence the research challenges here is to build redundancy and reliability in the network layer to handle a wide range of disruption scenarios such as congestion, short or long term disconnection, or last mile wireless impairments. Also an ICN network should allow features such as opportunistic store and forward mechanism to be enabled only at certain points in the network, as these mechanisms also entail overheads in the control and forwarding plane overhead which will adversely affect application throughput.

5.10. Energy Efficiency

All the optimizations for other components of the ICN-IoT system (described in earlier subsections) can lead to optimized energy efficiency. As a result, we refer the readers to read sections 5.1-5.9 for challenges associated with energy efficiency for ICN-IoT.

6. Appendix

Several types of IoT applications exists, where the goal is efficient and secure management and communication among objects in the system and with the physical world through sensors, RFIDs and other devices. Below we list a few popular IoT applications. We omit the often used term "smart", though it applies to each IoT scenario below, and posit that IoT-style interconnection of devices to make these environments "smart" in today’s terms will simply be the future norm.

6.1. Homes

The home [10] is a complex ecosystem of IoT devices and applications including climate control, home security monitoring, smoke detection, electrical metering, health/wellness, and entertainment systems. In a unified IoT platform, we would inter-connect these systems through the Internet, such that they can interact with each other and make

decisions at an aggregated level. Also, the systems can be accessed and manipulated remotely. Challenges in the home include topology independent service discovery, common protocol for heterogeneous device/application/service interaction, policy based routing/ forwarding, service mobility as well as privacy protection. Notably, the ease-of-use expectations and training of both users and installers also presents challenges in user interface and user experience design that are impacted by the complexity of network configuration, brittleness to change, configuration of trust management, etc. Finally, it is unlikely that there will be a single "home system", but rather a collection of moderately inter-operable collaborating devices. In addition, several IoT-enabled homes could form a smart district where it becomes possible to bargain resources and trade with utility suppliers.

Homes [12][13] faces the following challenges that are hard to address with IP-based overlay solutions: (1) context-aware control: home systems must make decisions (e.g., on how to control, when to collect data, where to carry out computation, when to interact with end-users, etc.) based upon the contextual information [14]; (2) inter-operability: home systems must operate with devices that adopt heterogeneous naming, trust, communication, and control systems; (3) mobility: home systems must deal with mobility caused by the movement of sensors or data receivers; (4) security: a home systems must be able to deal with foreign devices, handle a variety of user permissions (occupants of various types, guests, device manufacturers, installers and integrators, utility and infrastructure providers) and involve users in important security decisions without overwhelming them; (5) user interface / user experience: homes need to provide reasonable interfaces to their highly heterogeneous IoT networks for users with a variety of skill levels, backgrounds, cultures, interests, etc.

Smart homes have the following specific requirement for the underlying architecture:

- Smart homes require names that can enable local and wide area interactions; Also, security, privacy, and access control is particularly important for smart homes.

- Smart homes may use in-network caching at gateway to enable efficient content access.

- In smart homes, we need local, intra-domain and inter-domain routing protocols.

- In smart homes many control loops and actions depend heavily on the context, and the contexts evolve with time, e.g., temperature, weather, number of occupants, etc.

- In smart homes, local services can provide value-added contributions to a standardized home gateway network, through features such as reporting, context-based control, coordination with mobile devices, etc.

- In smart homes, the access to networked information should be shielded to protect the privacy of people, for example, cross-correlation of device activity patterns to infer higher-level activity information.

6.2. Enterprise

Enterprise building deployments, from university campuses [15] [65] [66] [67] to industrial facilities and retail complexes, drive an additional set of scalability, security, and integration requirements beyond the home, while requiring much of its ease of use and flexibility. Additionally, they bring requirements for integration with business IT systems, though often with the additional support of in-house engineering support.

Increasing number of enterprises are equipped with sensing and communication devices inside buildings, laboratories, and plants, at stadiums, in parking lots, on school buses, etc. A unified IoT platform must integrate many aspects of human interaction, H2M and M2M communication, within the enterprise, and thus enable many IoT applications that can benefit a large body of enterprise affiliates. The challenges in smart enterprise include efficient and secure device/data/resource discovery, inter-operability between different control systems, throughput scaling with number of devices, and unreliable communication due to mobility and telepresence.

Enterprises face the following challenges that are hard to address with IP-based overlay solutions: (1) efficient device/data/ resource discovery: enterprise devices must be able to quickly and securely discover requested device, data, or resources; (2) scalability: a enterprise system must be able to scale efficiently with the number and type of sensors and devices across not only a single building but multi-national corporations (for example); (3) mobility: a enterprise system must be able to deal with mobility caused by movement of devices; (4) security: security for IoT applications in the enterprise should integrate with other enterprise-wide security components.

6.3. Smart Grid

Central to the so-called "smart grid"[16] is data flow and information management, achieved by using sensors and actuators, which enables important capabilities such as substation and distribution automation. In a unified IoT platform, data collected from different smart grids can be integrated to achieve more optimizations that include reliability, real-time control, secure communications, and data privacy.

Deployment of the smart grid [17] [21] faces the following issues that are hard to address with IP-based overlay solutions: (1) scalability: future electrical grids must be able to scale gracefully to manage a large number of heterogeneous devices; (2) real time: grids must be able to perform real-time data collection, data processing and control; (3) reliability: grids must be resilient to hardware/software/networking failures; (4) security: grids and associated systems are often considered critical infrastructure -- they must be able to defend against malicious attacks, detect intrusion, and route around disruption.

Smart grids have the following specific requirements for the underlying IoT architecture:

- Smart grids require names and name resolution system that can enable networked control loops, real-time control, and security.
- Smart grids may use in-network caching to back up valuable data improving reliability.
- In smart grids, we often require very timely data delivery. Therefore, it is important to be able to locate the closest information. In addition, routing/forwarding robustness and resilience is also critical.
- In smart grids, contextual information such as location, time, voltage fluctuations, depending on the specific segment of the grid, can be used to optimize several power distribution objectives.
- In smart grids, we often rely on in-network computing to increase the scalability and efficiency of the system, putting computation closer to the data sources.
- In smart grids, energy consumptions profiles should never be disclosed at a fine granularity as it can be used to violate user privacy.

6.4. Transportation

We are currently witnessing the increasing integration of sensors into cars, other vehicles transportation systems [22]. Current production cars already carry many sensors ranging from rain gauges and accelerometers over wheel rotation/traction sensors, to cameras. These sensors can not only be used for internal vehicle functions, but they could also be networked and leveraged for applications such as monitoring external traffic/road conditions. Further, we can build vehicle-to- infrastructure (V2I),Vehicle-to-Roadside (V2R), and vehicle-to- vehicle (V2V) communications that enable many more applications for safety, convenience, entertainment, etc. The challenges for transportation include fast data/device/service discovery and association, efficient communications with mobility, trustworthy data collection and exchange.

Transportation [22][25] faces the following challenges that are hard to address with IP-based overlay solutions: (1) mobility: a transportation system must deal with a large number of mobile nodes interacting through a combination of infrastructure and ad hoc communication methods; ; also, during the journey the user might cross several realms, each one implementing different stacks (whether ICN or IP); (2) real-time and reliability: transportation systems must be able to operate in real-time and remain resilient in the presence of failures; (3) in-network computing/filtering: transportation systems will benefit from in-network computing/ filtering as such operations can reduce the end-to-end latency; (4) inter-operatibility: transportation systems must operate with heterogeneous device and protocols; (5) security: transportation systems must be resilient to malicious physical and cyber attacks.

Smart transportation applications have the following specific requirements for the underlying IoT architecture:

- Smart transportation systems require names and name resolution system to be able to handle extreme mobility, short latency and security. In addition, the mobility patterns of transportation systems increase the likelyhood that a user migrates from one network realm to another one during the journey. In this case, names and NRS should be designed in such a way to enable interoperability between different heterogeneous ICN realms and/or ICN and IP realms [68].

- Smart transportation may implement in-network caching on vehicles for efficient information dissemination

- In smart transportation, vehicle-to-vehicle ad-hoc communication is required for efficient information dissemination.

- In smart transportation, many different contexts exist, intertwined to each other and highly changing, which include location - both geographical and jurisdictional, time - absolute and relative to a schedule, traffic, speed, etc.

- In smart transportation, in-network computing is very useful to make vehicle become an active element of the system and to improve response time and scalability.

- In smart transportation, the habits of users can be inferred by looking at their movement patterns -- privacy protection is essential.

6.5. Healthcare

As more embedded medical devices, or devices that can monitor human health become increasingly deployed, healthcare is becoming a viable alternative to traditional healthcare solutions [26]. Further, consumer applications for managing and interacting with health data are a burgeoning area of research and commercial applications. For future health applications, a unified IoT platform is critical for improved patient care and consumer health support by sharing data across systems, enabling timely actuations, and lowering the time to innovation by simplifying interaction across devices from many manufacturers. Challenges in healthcare include real-time interactions, high reliability, short communication latencies, trustworthy, security and privacy, and well as defining and meeting the regulatory requirements that should impact new devices and their interconnection. In addition to this dimension, assistive robotics applications are gaining momentum to provide 24/24 7/7 assistance to patients [59].

Healthcare [26][27] faces the following challenges that are hard to address with IP-based overlay solutions: (1) real-time and reliability: healthcare systems must be able to operate on real-time and remain resilient in the presence of failures; (2) inter-operability: healthcare systems must operate with heterogeneous devices and protocols; (3) security: healthcare systems must be resilient to malicious physical and cyber attacks and meet the regulatory requirement for data security and interoperability; (4) privacy: user trust in healthcare systems is critical, and privacy considerations paramount to garner adoption and continued user; (5) user interface / user experience: the highly heterogeneous nature of real-world healthcare systems, which will continue to increase through the introduction of IoT devices, presents significant challenges in interface design that may have architectural implications.

Smart healthcare applications have the following specific requirements for the underlying IoT architecture:

- Smart healthcare system requires names and name resolution system to enable real- time interactions, dependability, and security.
- Smart healthcare may use in-network caching for rapid information dissemination.
- In smart healthcare, timely and dependable routing and information forwarding is the key.
- In smart healthcare several contexts can be used to delineate between levels of care and urgency, for example delineating between chronic, everyday, urgent, and emergency situations. Such contexts can evolve rapidly with significant impact to individuals health. Hence timely and accurate detection of contexts is critical.
- In smart healthcare, in-network computing can help resolve contexts and ensure security and dependability, as well as provide low-latency responses to urgent situations.
- In smart healthcare, personal medical data about patients should remain shielded to protect their privacy, implementing both regulatory requirements and current industry best practices.

6.6. Education

IoT technologies enable the instrumentation of a variety of environments (from greenhouses to industrial plants, homes and vehicles) to support not only their everyday operation but an understanding of how they operate -- a fundamental contribution to education. The diverse uses of hobbyist-oriented micro-controller platforms (e.g., the Arduino) and embedded systems (e.g., the Raspberry PI) point to a burgeoning community that should be supported by the next generation IoT platform because of its fundamental importance to formal and informal education.

Educational uses of IoT deployments include both learning about the operation of the system itself as well as the systems being observed and controlled. Such deployments face the following challenges that are hard to address with IP-based overlay solutions: (1) relatively simple communications patterns are obscured by many layers of translation from the host-based addressing of IP (and layer 2 configuration below) to the name-oriented interfaces provided by developers; (2) security considerations with overlay deployments and channel-based limit access to systems where read-only use of data is

not a security risk; (3) real-time communication helps make the relationship between physical phenomena and network messages easier to understand in many simple cases; (4) integration of devices from a variety of sources and manufacturers is currently quite difficult because of varying standards for basic communication, and limits experimentation; (5) programming interfaces must be carefully developed to expose important concepts clearly and in light of current best practices in education.

Smart campus systems have the following specific requirements for the underlying IoT architecture:

- Smart campus systems usually consist of heterogeneous IoT services, thus requiring names and name resolution system to enable resource/ service ownership, and be application-centric.
- Smart campus systems may use in-network caching to enable social interactions and efficient content access.
- In smart campus, inter-domain routing protocols are required which often need short latency.
- In smart campus, due to the existence of many services, relevant contextual inputs can be used to improve the quality and efficiency of different services.
- In smart campus, in-network computing services can be used to provide context for different applications.
- In smart campus, it is required to differentiate among different profiles and to allocate different rights and protection levels to them.

6.7. Entertainment, arts, and culture

IoT technologies can contribute uniquely to both the worldwide entertainment market and the fundamental human activity of creating and sharing art and culture. By supporting new types of human-computer interaction, IoT can enable new gaming, film/video, and other "content" experiences, integrating them with, for example, the lighting control of the smart home, presentation systems of the smart enterprise, or even the incentive mechanisms of smart healthcare systems (to, say, encourage and measure physical activity).

Entertainment, arts, and culture applications generate a variety of challenges for IoT: (1) notably, the ability to securely "repurpose" deployed smart systems (e.g., lighting) to create experiences; (2) low-latency communication to enable end-user responsiveness; (3)

integration with infrastructure-based sensing (e.g., computer vision) to create comprehensive interactive environments or to provide user identity information; (4) time synchronization with audio/video playback and rendering in 3D systems (5) simplicity of development and experimentation, to enable the cost- and time-efficient integration of IoT into experiences being designed without expert engineers of IoT systems; (6) security, because of integration with personal devices and smart environments, as well as billing systems.

7. Informative References

[1] Cisco System Inc., CISCO., "Cisco visual networking index: Global mobile data traffic forecast update.", 2009-2014.

[2] Shafig, M., Ji, L., Liu, A., Pang, J., and J. Wang, "A first look at cellular machine-to-machine traffic: large scale measurement and characterization.", Proceedings of the ACM Sigmetrics , 2012.

[3] The European Telecommunications Standards Institute, ETSI., "http://www.etsi.org/.", 1988.

[4] Global Intiative for M2M Standardization, oneM2M., "http://www.onem2m.org/.", 2012.

[5] Constrained RESTful Environments, CoRE., "https://datatracker.ietf.org/wg/core/charter/.", 2013.

[6] Ghodsi, A., Shenker, S., Koponen, T., Singla, A., Raghavan, B., and J. Wilcox, "Information-Centric Networking: Seeing the Forest of the Trees.", Hot Topics in Networking , 2011.

[7] Dong, L., Zhang, Y., and D. Raychaudhuri, "Enhance Content Broadcast Efficiency in Routers with Integrated Caching.", Proceedings of the IEEE Symposium on Computers and Communications (ISCC) , 2011.

[8] NSF FIA project, MobilityFirst., "http://www.nets-fia.net/", 2010.

[9] Kim, B., Lee, S., Lee, Y., Hwang, I., and Y. Rhee, "Mobiiscape: Middleware Support for Scalable Mobility Pattern Monitoring of Moving Objects in a Large-Scale City.", Journal of Systems and Software, Elsevier, 2011.

[10] Dietrich, D., Bruckne, D., Zucker, G., and P. Palensky, "Communication and Computation in Buildings: A Short Introduction and Overview", IEEE Transactions on Industrial Electronics, 2010.

[11] Keith, K., Falco, F., and K. Scarfone, "Guide to Industrial Control Systems (ICS) Security", NIST, Technical Report 800-82 Revision 1, 2013.

[12] Darianian, M. and Martin. Michael, "Smart home mobile RFID-based Internet-of-Things systems and services.", IEEE, ICACTE, 2008.

[13] Zhu, Q., Wang, R., Chen, Q., Chen, Y., and W. Qin, "IOT Gateway: Bridging Wireless Sensor Networks into Internet of Things", IEEE/IFIP, EUC, 2010.

[14] Biswas, T., Chakrabort, A., Ravindran, R., Zhang, X., and G. Wang, "Contextualized information-centric home network", ACM, Siggcomm, 2013.

[15] Huang, R., Zhang, J., Hu, Y., and J. Yang, "Smart Campus: The Developing Trends of Digital Campus", 2012.

[16] Yan, Y., Qian, Y., Hu, Y., and J. Yang, "A Survey on Smart Grid Communication Infrastructures: Motivations, Requirements and Challenges", IEEE Communications Survey and Tutorials, 2013.

[17] Miao, Y. and Y. Bu, "Research on the Architecture and Key Technology of Internet of Things (loT) Applied on Smart Grid", IEEE, ICAEE, 2010.

[18] Castro, M. and A. Jara, "An analysis of M2M platforms: challenges and opportunities for the Internet of Things", IMIS, 2012.

[19] Gubbi, J., Buyya, R., and S. Marusic, "Internet of Things (IoT): A vision, architectural elements, and future directions", Future Generation Computer Systems, 2013.

[20] Vandikas, K. and V. Tsiatsis, "Performance Evaluation of an IoT Platform. In Next Generation Mobile Apps, Services and Technologies(NGMAST)", Next Generation Mobile Apps, Services and Technologies (NGMAST), 2014.

[21] Zhang, Y., Yu, R., Nekovee, M., Liu, Y., Xie, S., and S. Gjessing, "Cognitive Machine-to-Machine Communications: Visions and Potentials for the Smart Grid", IEEE, Network, 2012.

[22] Zhou, H., Liu, B., and D. Wang, "Design and Research of Urban Intelligent Transportation System Based on the Internet of Things", Springer Link, 2012.

[23] Alessandrelli, D., Petracca, M., and P. Pagano, "T-Res: enabling reconfigurable in-network processing in IoT-based WSNs.", International Conference on Distributed Computing in Sensor Systems (DCOSS) , 2013.

[24] Kovatsch, M., Mayer, S., and B. Ostermaier, "Moving application logic from the firmware to the Cloud: towards the thin server architecture for the internet of things.", in Proc. 6th Int. Conf. on Innovative Mobile and Internet Services in Ubiquitous Computing (IMIS) , 2012.

[25] Zhang, M., Yu, T., and G. Zhai, "Smart Transport System Based on the Internet of Things", Applied Mechanics and Materials, 2012.

[26] Zhang, A., Yu, R., Nekovee, M., and S. Xie, "The Internet of Things for Ambient Assisted Living", IEEE, ITNG, 2010.

[27] Savola, R., Abie, H., and M. Sihvonen, "Towards metrics-driven adaptive security management in E-health IoT applications.", ACM, BodyNets, 2012.

[28] Jacobson, V., Smetters, D., Plass, M., Stewart, P., Thornton, J., and R. Braynard, "VoCCN: Voice-over Content-Centric Networks", ACM, ReArch, 2009.

[29] Piro, G., Cianci, I., Grieco, L., Boggia, G., and P. Camarda, "Information Centric Services in Smart Cities", ACM, Journal of Systems and Software, 2014.

[30] Ravindran, R., Biswas, T., Zhang, X., Chakrabort, A., and G. Wang, "Information-centric Networking based Homenet", IEEE/IFIP, 2013.

[31] Dannewitz, C., D’ Ambrosio, M., and V. Vercellone, "Hierarchical DHT-based name resolution for information-centric networks", 2013.

[32] Fasoloy, E., Rossiy, M., and M. Zorziy, "In-network Aggregation Techniques for Wireless Sensor Networks: A Survey", IEEE Wireless Communications, 2007.

[33] Chai, W., He, D., and I. Psaras, "Cache "less for more" in information-centric networks", ACM, IFIP, 2012.

[34] Eum, S., Nakauchi, K., Murata, M., Shoji, Yozo., and N. Nishinaga, "Catt: potential based routing with content caching for icn", IEEE Communication Magazine, 2012.

[35] Drira, W. and F. Filali, "Catt: An NDN Query Mechanism for Efficient V2X Data Collection", Eleventh Annual IEEE International Conference on Sensing, Communication, and Networking Workshops (SECON Workshops), 2014.

[36] Eum, S., Shvartzshnaider, Y., Francisco, J., Martini, R., and D. Raychaudhuri, "Enabling internet-of-things services in the mobilityfirst future internet architecture", IEEE, WoWMoM, 2012.

[37] Sun, Y., Qiao, X., Cheng, B., and J. Chen, "A low-delay, lightweight publish/subscribe architecture for delay-sensitive IOT services", IEEE, ICWS, 2013.

[38] Baccelli, E., Mehlis, C., Hahm, O., Schmidt, T., and M. Wahlisch, "Information Centric Networking in the IoT:Experiments with NDN in the Wild", ACM, ICN Siggcomm, 2014.

[39] Gronbaek, I., "Architecture for the Internet of Things (IoT): API and interconnect", IEEE, SENSORCOMM, 2008.

[40] Tian, Y., Liu, Y., Yan, Z., Wu, S., and H. Li, "RNS-A Public Resource Name Service Platform for the Internet of Things", IEEE, GreenCom, 2012.

[41] Roussos, G. and P. Chartier, "Scalable id/locator resolution for the iot", IEEE, iThings,CPSCom, 2011.

[42] Amadeo, M. and C. Campolo, "Potential of information-centric wireless sensor and actor networking", IEEE, ComManTel, 2013.

[43] Nelson, S., Bhanage, G., and D. Raychaudhuri, "GSTAR: generalized storage-aware routing for mobilityfirst in the future mobile internet", ACM, MobiArch, 2011.

[44] Trappe, W., Zhang, Y., and B. Nath, "MIAMI: methods and infrastructure for the assurance of measurement information", ACM, DMSN, 2005.

[45] Rouf, I., Mustafa, H., Taylor, T., Oh, S., Xu, W., Gruteser, M., Trappe, W., and I. Seskar, "Security and privacy vulnerabilities of in-car wireless networks: A tire pressure monitoring system case study", USENIX, 2010.

[46] Liu, R. and W. Trappe, "Securing Wireless Communications at the Physical Layer", Springer, 2010.

[47] Xiao, L., Greenstein, L., Mandayam, N., and W. Trappe, "Using the physical layer for wireless authentication in time-variant channels", IEEE Transactions on Wireless Communications, 2008.

[48] Sun, S., Lannom, L., and B. Boesch, "Handle system overview", IETF, RFC3650, 2003.

[49] Shelby, Z., Hartke, K., and C. Bormann, "The Constrained Application Protocol (CoAP)", RFC 7252, DOI 10.17487/RFC7252, June 2014, <http://www.rfc-editor.org/info/rfc7252>.

[50] Fielding, R., Ed. and J. Reschke, Ed., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", RFC 7230, DOI 10.17487/RFC7230, June 2014, <http://www.rfc-editor.org/info/rfc7230>.

[51] Sun, S., "Hypertext Transfer Protocol (HTTP/1.1): Message Syntax and Routing", 2014.

[52] Liu, X., Trappe, W., and Y. Zhang, "Secure Name Resolution for Identifier-to-Locator Mappings in the Global Internet", IEEE, ICCCN, 2013.

[53] Boguna, M., Fragkiskos, P., and K. Dmitri, "Sustaining the internet with hyperbolic mapping", Nature Communications, 2010.

[54] Shang, W., "Securing building management systems using named data networking", IEEE Network 2014.

[55] Fayazbakhsh, S. and et. et al, "Less pain, most of the gain: Incrementally deployable icn", ACM, Siggcomm, 2013.

[56] Burke, J. and et. et al, "Securing instrumented environments over Content-Centric Networking: the case of lighting control", INFOCOM, Computer Communications Workshop, 2013.

[57] Li, S., Zhang, Y., Dipankar, R., and R. Ravindran, "A comparative study of MobilityFirst and NDN based ICN-IoT architectures", IEEE, QShine, 2014.

[58] Grieco, L., Alaya, M., and K. Drira, "Architecting Information Centric ETSI-M2M systems", IEEE, Pervasive and Computer Communications Workshop (PERCOM), 2014.

[59] Grieco, L., Rizzo, A., Colucci, R., Sicari, S., Piro, G., Di Paola, D., and G. Boggia, "IoT-aided robotics applications: technological implications, target domains and open issues", Computer Communications, Volume 54, 1 December, 2014.

[60] Quan, Wei., Xu, C., Guan, J., Zhang, H., and L. Grieco, "Scalable Name Lookup with Adaptive Prefix Bloom Filter for Named Data Networking", IEEE Communications Letters, 2014.

[61] Wang, Yi., Pan, T., Mi, Z., Dai, H., Guo, X., Zhang, T., Liu, B., and Q. Dong, "NameFilter: Achieving fast name lookup with low memory cost via applying two-stage Bloom filters", INFOCOM, 2013.

[62] So, W., Narayanan, A., Oran, D., and Y. Wang, "Toward fast NDN software forwarding lookup engine based on Hash tables", ACM, ANCS, 2012.

[63] Amadeo, M., Campolo, C., Iera, A., and A. Molinaro, "Named data networking for IoT: An architectural perspective", IEEE, EuCNC, 2014.

[64] Sifalakis, M., Kohler, B., Christopher, C., and C. Tschudin, "An information centric network for computing the distribution of computations", ACM, ICN Sigcomm, 2014.

[65] Lu, R., Lin, X., Zhu, H., and X. Shen, "SPARK: a new VANET-based smart parking scheme for large parking lots", INFOCOM, 2009.

[66] Wang, H. and W. He, "A reservation-based smart parking system", The First International Workshop on Cyber-Physical Networking Systems, 2011.

[67] Qian, L., "Constructing Smart Campus Based on the Cloud Computing and the Internet of Things", Computer Science 2011.

[68] Project, BonVoyage., "From Bilbao to Oslo, intermodal mobility solutions, interfaces and applications for people and goods, supported by an innovative communication network", Call H2020-MG-2014, 2015-2018.

[69] Li, S., Zhang, Y., Raychaudhuri, D., Ravindran, R., Zheng, Q., Wang, GQ., and L. Dong, "IoT Middleware over Information-Centric Network", Global Communications Conference (GLOBECOM) ICN Workshop, 2015.

[70] Li, S., Chen, J., Yu, H., Zhang, Y., Raychaudhuri, D., Ravindran, R., Gao, H., Dong, L., Wang, GQ., and H. Liu, "MF-IoT: A MobilityFirst-Based Internet of Things Architecture with Global Reachability and Communication Diversity", IEEE International Conference on Internet-of-Things Design and Implementation (IoTDI), 2016.

[71] Campolo, C., Corujo, D., Iera, A., and R. Aguiar, "Information-centric Networking for Internet-of-things: Challenges and Opportunities", IEEE Networks, Jan , 2015.

Authors' Addresses

Prof.Yanyong Zhang
WINLAB, Rutgers University
671, U.S 1
North Brunswick, NJ 08902
USA

Email: yyzhang@winlab.rutgers.edu

Prof. Dipankar Raychadhuri
WINLAB, Rutgers University
671, U.S 1
North Brunswick, NJ 08902
USA

Email: ray@winlab.rutgers.edu

Prof. Luigi Alfredo Grieco
Politecnico di Bari (DEI)
Via Orabona 4
Bari 70125
Italy

Email: alfredo.grieco@poliba.it

Prof. Emmanuel Baccelli
INRIA
Room 148, Takustrasse 9
Berlin 14195
France

Email: Emmanuel.Baccelli@inria.fr

Jeff Burke
UCLA REMAP
102 East Melnitz Hall
Los Angeles, CA 90095
USA

Email: jburke@ucla.edu

Ravishankar Ravindran
Huawei Technologies
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: ravi.ravindran@huawei.com

Guoqiang Wang
Huawei Technologies
2330 Central Expressway
Santa Clara, CA 95050
USA

Email: gq.wang@huawei.com

Andres Lindgren
SICS Swedish ICT
Box 1263
Kista SE-164 29
SE

Email: andersl@sics.se

Bengt Ahlgren
SICS Swedish ICT
Box 1263
Kista, CA SE-164 29
SE

Email: bengta@sics.se

Olov Schelen
Lulea University of Technology
Lulea SE-971 87
SE

Email: lov.schelen@ltu.se