

CCNx Selector Based Discovery

draft-mosko-icnrg-selectors-01

Abstract

CCNx selector based discovery uses exclusions and interest name suffix matching to discover content in the network. Participating nodes may respond with matching Content Objects from cache using an encapsulation protocol. This document specifies the available selectors, their encoding in a name path segment, and the encapsulation protocol.

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 February 4, 2017.

Copyright Notice

Copyright (c) 2016 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust's Legal Provisions Relating to IETF Documents (<http://trustee.ietf.org/license-info>) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.

Table of Contents

1. Introduction
 - 1.1. Requirements Language
2. Protocol Description
3. Name Labels and TLV types
 - 3.1. Child Selector
 - 3.2. Interest Min(Max)SuffixComponents
 - 3.3. Name Excludes

- 3.3.1. Exclude Singleton
- 3.3.2. Exclude Range
- 3.3.3. Examples
- 4. Content Store and Caching
- 5. Annex A: Examples
- 6. IANA Considerations
- 7. Security Considerations
- 8. References
 - 8.1. Normative References
 - 8.2. Informative References
- § Author's Address

TOC

1. Introduction

Content Discovery is an important feature of CCNx. This document specifies a discovery mechanism that uses a name path segment to encode a discovery query in an Interest. Participating nodes may reply with a Content Object if it matches the encoded query. The query uses exclusions to work around incorrect responses.

This document specifies a new name TLV type, T_SELECTOR, for selector query. It also specifies a new Content Object PayloadType that encapsulates another Content Object. The inner Content Object is used to return a Content Object with a longer name than in an interest. The inner object's signature should verify.

Not all nodes along the Interest path need to participate in the discovery process. A non-participating node should forward the Interest and encapsulating Content Object as normal per [CCN semantics] (Mosko, M., Solis, I., and C. Wood, "CCNx Semantics (Internet draft)," 2016.). A participating node should verify that the inner Content Object matches the selector query in the PIT entry.

Note that Selector discovery is not needed when asking for a Content Object by its ContentObjectHash, as there should only ever be one match for that.

Selector discovery in CCNx 1.0 differs in three ways from the current NDN and prior CCNx 0.x selector discovery. First, CCNx 1.0 uses a distinguished field for the ContentObjectHashRestriction. It is not appended to the name to form the so-called "full name." This means that there is no implicit digest name segment. Thus, using a MinSuffixComponents and MaxSuffixComponents of 0 will match the exact name in the Interest without needing to add one extra component to account for the implicit digest. Second, there is a HashExcludes field that lists ContentObjectHashRestrictions to exclude instead of appending them as an implicit name component. Third, the encoding of Excludes differs from prior encodings and uses a simpler formulation with the same expressiveness that also takes in to consideration that name segments in CCNx 1.0 have TLV types associated with them.

CCNx 1.0 allows Content Objects to have no name and be retrieved by hash only. As they have no name, they are not discoverable via name-based selector discovery.

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.

TOC

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 (Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," March 1997.) [RFC2119].

TOC

2. Protocol Description

Selector based discovery uses seven query variables to discover content. These selectors are encoded as a single name path segment affixed to an Interest name. The selectors operate on the prefix up to, but not including the selector name path segment. The selector name path segment should be the last path segment.

The selectors are:

- **MinSuffixComponents**: the minimum number of additional name path segments a matching Content Object must have in its name. The default value is 0.
- **MaxSuffixComponents**: The maximum number of additional name path segments a matching Content Object may have in its name. The default value is unlimited.
- **ChildSelector**: Answer with the left-most or right-most child.
- **NameExcludes**: A set of range and singleton exclusions to eliminate Content Objects. The exclusions match against the name path segment that would immediately follow the Interest name prefix up to but not including the Selector path segment.
- **InnerKeyId**: Matches the KeyId of the encapsulated object.
- **HashExcludes**: A list of ContentObjectHashRestrictions to exclude.
- **SelectorNonce**: A number to make the query unique.

A node using Selector discovery appends a Selector name path segment to the end of the Interest name. Even if no selectors are used, the Selector path segment is added to the end, which indicates to a participating node that it should apply Selector based matching to the Interest. In this case, the default values -- if any -- of each selector are used.

A node receiving a Selector Interest should match against the Content Store using the selector rules. Based on the sort order, it should pick the appropriate Content Object, if any, and return it in an Encapsulation Object. If no Content Objects match, the Interest should be forwarded or NACKed as normal.

An Encapsulation Object is a Content Object that matches the Selector Interest and whose payload is the discovered Content Object. The ContentType of an Encapsulation Object is "ENCAP". The outer name matches the Selector Interest name. The inner Content Object name matches the Selector discovery.

The `KeyIdRestriction` of the Interest matches the outer `KeyId` of the outer Content Object, as normal. This allows a responding cache or producer to sign (or MAC or MIC) the response. The `InnerKeyId` of the Selector matches the inner ContentObject in the same way. This allows the selector to discriminate discovery including the inner `KeyId`.

The `HashExcludes` eliminate any Content Objects whose `ContentObjectHash` matches any of the listed values. It should not matter if matching objects are discarded before name prefix selector matching or after. A Content Object must always pass both the `HashExcludes` filter and the name prefix selector filters, whether it is done first or last does not matter. `HashExcludes` are encoded the same way as a `ContentObjectHashRestriction` value in an Interest. Note that this Selector does not exist in NDN or CCNx 0.x. We use an explicit set of `HashExcludes` rather than constructing a full name with the implicit digest component at the end.

A consumer MAY include a `SelectorNonce`. This nonce is to make the query unique to bypass cached responses to the same Selectors at non-participating nodes. A consumer SHOULD use this field if it receives a non-conforming response in an encapsulated ContentObject and cannot further exclude that response. If an attacker were able to inject an incorrect response into a non-participating cache then that non-participating node cannot determine that the response it is serving from cache is correct or not. Therefore, a consumer can use the `SelectorNonce` to make its request name different from the cached name. Note that if all nodes are participating, then this field has no effect as it is not processed by them. The `SelectorNone` is not used for loop detection and may be as few bytes as needed to avoid a cached response.

If an authoritative producer receives a Selector discovery, it SHOULD generate the inner Content Object as normal and encapsulate it as normal. It MAY also respond with an Interest Return or not respond at all. At the present, responding directly to the Selector Interest with data without encapsulating it is not supported. Note that an application is NOT REQUIRED to implement Selector discovery; if the application wishes to make use of this mechanism, then it must implement it, if it does not use this mechanism then it does not need to implement it.

Normally, the outer Content Object does not have a Validation section. A responding node MAY include a CRC32C or other integrity check. Signing or MACing an outer Content Object is possible, but should only be used in environments where that degree of trust is necessary. Signing the outer Content Object in no way replaces the signature (if any) of the inner Content Object. The outer signature only identifies the responding cache (or producer).

TOC

3. Name Labels and TLV types

The Selector name segment type `T_SELECTOR` has type `%x0010`.

The `PayloadType` of `T_PAYLOADTYPE_ENCAP` has the value 8.

Type	Symbol	Name	Description
1	<code>T_MINSUFFIX</code>	Selectors: Min Suffix	Minimum number of additional name components after given name to match (0 default if missing).

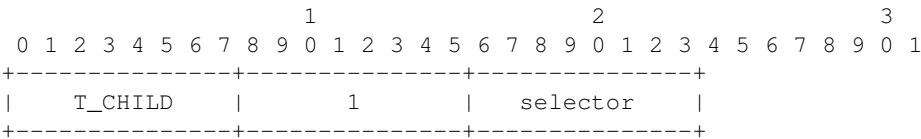
Components			
2	T_MAXSUFFIX	Selectors: Max Suffix Components	Maximum number of additional name components after given name to match (unlimited default is missing).
3	T_CHILD	Selectors: Child Selector	0 = left, 1 = right (default)
4	T_NAME_EXCLUDES	Name Excludes	Encloses ExcludeComponents
1	T_EX_SINGLE	Exclude Singleton	Exclude a single name path segment.
2	T_EX_RANGE	Exclude Range	Exclude an half-open range, beginning at this value and continuing up to the next Singleton, or to infinity if omitted on the last entry.
5	T_INNER_KEYID	Inner KeyId	A restriction on the inner keyid. If present, it must match the KeyId of the inner Content Object in the encapsulated response.
6	T_HASH_EXCLUDES	Hash Excludes	Excludes a set of ContentObjectHash from the allowed responses. Each restriction is encoded using its Hash Function Type Registry type (e.g. T_SHA-256) from [CCNMessages] (Mosko, M., Solis, I., and C. Wood, "CCNx Messages in TLV Format (Internet draft)," 2016.).

Table 1: Selector Types

TOC

3.1. Child Selector

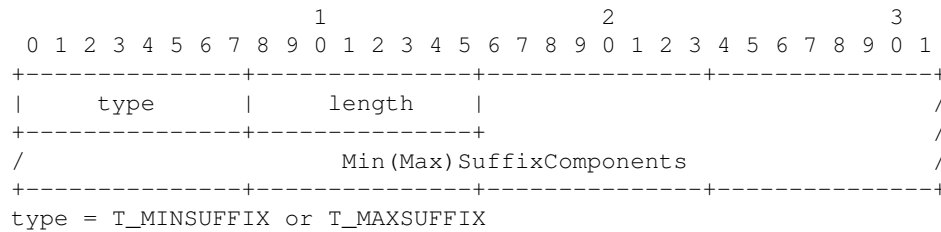
If there are multiple choices to answer an Interest, the Child Selector specifies the desired ordering of responses. %x00 = leftmost, %x01 = rightmost.



TOC

3.2. Interest Min(Max)SuffixComponents

The Min and Max suffix components are encoded as a minimum-length unsigned integer in network byte order number inside the value. A "0" is represented as a single byte %0x00. A length 0 value is interpreted the same as the type not being present.



3.3. Name Excludes

Interest Excludes specify a set of singletons and ranges to exclude when matching Content Object names to an Interest. They match the name component immediately following the last component of the Interest name (not including the Selector TLV). The excludes must be sorted in ascending order, using the Exclude sorting rules below.

A name exclusion is the full TLV expression of a name component, not just it's value.

Exclude Sorting: An exclusion value A is less than B iff the TLV type of A is less than the TLV type of B, or being equal, the TLV value of A is shortlex less than the TLV value of B. A shortlex comparison means that X is less than Y is X is shorter than Y or the lengths being equal, X lexicographically sorts before Y.

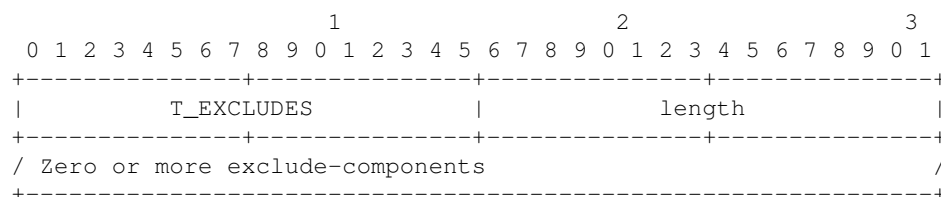
Using the normal 2+2 TLV encoding of [CCNMessages] (Mosko, M., Solis, I., and C. Wood, "CCNx Messages in TLV Format (Internet draft)," 2016.), the Exclude Sorting can be done by a byte-by-byte memcmp() of two TLVs. This is because the fixed length Type ensures correct type sorting and fixed length Length ensures correct shortlex length sorting. This will not necessarily be true of other encodings.

A zero-length exclusion is the minimum exclusion and must appear before any other exclusion. Note that a zero-length exclusion has no TLV type for the name component, so it will match any name segment TLV type. It is equivalent to minus infinity.

The zero-length name component is the minimum name component of that name component type (e.g. T_NAMESEGMENT).

An exclude may contain either an Exclude Range type or an Exclude Singleton type. An Exclude Range type means the given value starts a half-open exclusion range that begins inclusive of the Range value and ends open at the next Singleton or at infinity if it is the last exclude component. An Exclude Singleton means to exclude the exact value given.

Note that this syntax does not require the "ANY" exclude component that is part of the NDN and CCNx 0.x syntax.



```
exclude-components = *component [start-range-tlv]
component = (start-range-tlv singleton-tlv) / singleton-tlv
```

The ABNF of the exclude-component allows for zero or more components followed by an optional start-range-tlv. A component is either a half-open range (start-range-tlv singleton-tlv) or a singleton-tlv.

The optional final start-range-tlv has no terminating singleton-tlv. This means it extends out to plus infinity.

Note that to exclude from negative infinity to some value "foo", we do not need to include an ANY element because the zero-length name component is, by definition, the minimum element and we use inclusive range start. Therefore, beginning an exclusion with the zero-length range effectively excludes from minus infinity.

TOC

3.3.1. Exclude Singleton

A singleton exclude component means to exclude a name path segment exactly matching the given value.

```

      1              2              3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+
|           T_EX_SINGLE           |           length           |
+-----+-----+-----+
/                   TLV name segment                   /
+-----+-----+-----+
```

TOC

3.3.2. Exclude Range

A Range exclude means to exclude the from the given value up to but not including the next Singleton. If the Range is the last component in the Exclude, it means to exclude to infinity.

```

      1              2              3
0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-----+-----+-----+
|           T_EX_RANGE           |           length           |
+-----+-----+-----+
/                   TLV name segment                   /
+-----+-----+-----+
```

TOC

3.3.3. Examples

In these examples, we will use the notation S[foo] to represent a singleton exclusion "foo" and R[foo] to represent a range exclusion beginning at "foo." In the column Range, we use standard open (parenthesis) and closed (square bracket) interval notation. We assume all TLV name types of T_NAMESEGMENT if there is no explicit name segment type given. In our notation, something like S[VER=bar] would exclude a TLV type Version and value "bar".

Exclude Pattern	Range
S[ace]	NAME=ace
S[ace] R[bat]	NAME=ace, [NAME=bat, infity)
R[ace] S[bat]	[NAME=ace, NAME=bat)
R[CHUNK=0] S[CHUNK=20]	[CHUNK=0, CHUNK=20)
R[] S[ace]	(-infity, NAME=ace), matches any preceeding TLV types using a zero-length Range exclude
R[NAME=] S[ace]	[NAME=, NAME=ace)
R[]	(-infity, +infity)
S[zoo] S[ape]	Invalid range, not sorted
R[NAME=ace] S[CHUNK=0]	[NAME=ace, CHUNK=0), this will span TLV ranges type between T_NAMESEGMENT and T_CHUNK
R[CHUNK=] S[CHUNK+1=]	[CHUNK=, CHUNK+1=), excludes all CHUNK TLV possibilities

Table 2: CCNx Name Types

TOC

4. Content Store and Caching

The encapsulated responses to discovery are cachable, like all Content Objects. A participating forwarder MAY cache the inner Content Object separately from the outer Content Object assuming it passes the selector tests. A non-participating forwarder MAY only cache the outer Content Object (encapsulating the inner).

A participating content store MUST obey both the outer and inner cache control directives: ExpiryTime and RecommendedCacheTime. At a participating node, the outer and inner Content Objects are independent and cached independently. This is allowed because a participating node has verified that the inner ContentObject comes from an on-path direction of the routing prefix, so it cannot be an off-path injection of bad content.

A non-participating content store must obey the outer cache control directives, as normal. The inner content object is opaque data to it.

It is RECOMMENDED that a participating node creating the encapsulated response set a short ExpiryTime and MAY set a 0 ExpiryTime (to prevent all caching). This is desirable because non-participating nodes only look at the outer ExpiryTime and cannot determine if the inner ContentObject actually satisfies the Selector query. Note that a consumer can also use a SelectorNonce to avoid bad cache entries at non-participating nodes, so it is not necessary for correctness to use a 0 ExpiryTime.

Note that cached responses are, in general, not a problem for the discovery process. Participating nodes will always do a full selector match, so a consumer can work around incorrect responses as normal. Because Selector interests with different Exclude blocks will result in different names, prior responses will not match in the caches of non-participating nodes, especially if the ExpiryTime is set to 0.

TOC

5. Annex A: Examples

TODO: Provide complete examples of Interests and responses.

TOC

6. IANA Considerations

This memo includes no request to IANA. TODO: If this document is submitted as an official draft, this section must be updated to reflect the IANA registries described in [CCNMessages] (Mosko, M., Solis, I., and C. Wood, "CCNx Messages in TLV Format (Internet draft)," 2016.)

TOC

7. Security Considerations

Because responses use encapsulation, there is size expansion in the response from the original Content Object. The expansion will be the length of the encapsulating Selector name plus the size of any validation uses on the outer Content Object (e.g. a CRC32C), plus framing overhead. This means that one cannot respond with a Content Object that is too close to the maximum packet size.

Participating nodes should be able to filter incorrect responses just as they do in NDN or CCNx 0.x. If all nodes participate, then one has equivalent in-network filtering behavior as those other protocols.

If the outer Content Object is signed, the consumer should, as normal, verify the signature for accuracy. However, the trust of the outer signature is normally not important and usually reflects operation in a specific environment. An outer Validation section is usually used only for integrity checks.

8. References

8.1. Normative References

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels," BCP 14, RFC 2119, March 1997 (TXT, HTML, XML).

8.2. Informative References

- [CCNMessages] Mosko, M., Solis, I., and C. Wood, "CCNx Messages in TLV Format (Internet draft)," 2016.
- [CCNSemantics] Mosko, M., Solis, I., and C. Wood, "CCNx Semantics (Internet draft)," 2016.
- [CCNx] PARC, Inc., "CCNx Open Source," 2007.
- [RFC3552] Rescorla, E. and B. Korver, "Guidelines for Writing RFC Text on Security Considerations," BCP 72, RFC 3552, July 2003 (TXT).
- [RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs," BCP 26, RFC 5226, May 2008 (TXT).

Author's Address

Marc Mosko
PARC, Inc.
Palo Alto, California 94304
USA
Phone: +01 650-812-4405
Email: marc.mosko@parc.com