July 29, 2015
PARC

Encapsulated CCN

Marc Mosko[1]*

Abstract
We describe several alternatives to encapsulating CCN messages within other CCN messages. Encapsulation enables features such as IP-IP tunnels, where an outer address redirects a message over a different route than it would normally take. The first method, called *Link Concatenation* concatenates a Link message with a ContentObject message, essentially forming a redirect and the redirect data. The second method, called *Encapsulation*, wraps one message inside another message. We discuss the security implications of each method.

Keywords
Content Centric Networks

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

Contents

Introduction

Encapsulation is a well-known method to send a message via a different route than it would normally follow. For Internet Protocol, there are several options, such as IP-in-IP, GRE, or VPNs. In CCNx, there have been several informal proposals, and we seek here to consolidate the options and their implications to networking and security.

In CCNx 1.0, an Interest message carries a Name and an optional ContentObjectHash. A Content Object matches an Interest if the names are exactly equal. If the Interest also specified the ContentObjectHash restriction, then the SHA-256 hash of the entire Content Object must be computed and matched against the restriction in the Interest.

In CCNx, the Name is also used for routing. Therefore, the name not only identifies the piece of content, but also indicates where the content lives, or at least the name of the authority to say where it lives. The authority could respond with a Link to a different location, such as a different replica. Because the link it signed by the authority, the client may trust the transfer of security context. The link could be to just a name, but we believe for a transfer of security context the link should enumerate the (Name, KeyIdRestriction, ContentObjectHashRestriction) tuple.

The original CCNx documentation on Links and encapsulation is sparse. An unpublished, undated NDN document [1] goes in to much more detail on using Links. The document covers redirection – the basic function of a link – and what they call "delegation". Delegation assigns a namespace from A to B and includes a confirmation from B to A (an acceptance of the delegation). It also includes ideas on "LINK for Data Retrieval", which have multiple data type embedding, which has some similarity to what we call Link Concatenation, but is substantially different in how it is realized. The NDN document is concerned with doing arbitrary suffix matching on the embedded content, so the mechanisms become more complicated than presented here.

We have specifically not included Delegation. Delegation is the assignment of one namespace to another namespace. For example, Alice says that Bob is allowed to answer any name under /alice with a name /bob/alice. The NDN document allows prefix delegation in all cases, even for a simple Link message. We believe that a Link should only point from one concrete ContentObject (a Link) to another concrete ContentObject (Link or Data), not delegate authority for a second namespace to publish anything on behalf of the first namespace. In this document, all Links and encapsulations are between concrete objects with strong trust association via a signature or hash chain.

Section 1 describes the first method, called *Link Concatenation*, which concatenates a Link message with a ContentObject message, essentially forming a redirect and the redirect data. Sec. 2 describes the second method, called *Encapsulation*, which wraps one message inside another message.

1. Link Concatenation

As defined in the CCNx Semantics and Messages documents, a Link is a CCN ContentObject whose PayloadType is LINK and whose Payload contains a well-known message format pointing to the target of the link. The Link ContentObject has its own name, called the Link Name. It must match the corresponding Interest that solicited the ContentObject. The payload contains the tuple {TargetName, [TargetKeyId], [TargetHash] }, where the last two parameters are optional. The recipient of the Link may then issue a second Interest using the TargetName, TargetKeyId, and TargetHash. The link target may, itself, be another link, and by recursively following the links eventually arrive at the desired data. At each step, the recipient must make a trust decision based on the signer of each link.

To avoid multiple round trips, Link Concatenation provides both the Link ContentObject and the Data ContentObject in one response. If multiple Link traversals are needed, the response would concatenate the chain.

Because the embedded ContentObjects do not follow normal routing, an intermediate system *must not* cache anything except the first Link. It may be possible to cache the inner messages, but only to answer Interests that request content by ContentObjectHash restriction or by KeyId restrictions where the intermediate node has cryptographically verified the signature.

1.1 Simple Concatenation

The Simple Concatenation approach sends a series of links followed by one data item. Each link has an individual signature, which should be signed with a key trusted for the Link Name namespace. The final Data object should be signed by a key trusted for its namespace, or follow a hash chain from the terminal Link.

Fig. 1 shows the ABNF for a Link Concatenation. The FixedHeader covers all messages in the packet, so the size of all links and the target Data ContentObject must fit within the 64KB packet limit. The chain of links must be verifiable. That is, in the serial order given, the first LinkName must match the InterestName, and its {TagetName, [TargetKeyId], [TargetHash] } must match the next element in the list. An intermediate node is required to verify the chain, as it would for normal Interest processing: it matches the name and keyid by equality and the hash by computation.

An end system receiving a Link Concatenation should verify all cryptographic signatures and make a trust decision on each one. For example, if the first Link is for `/apple` and the first TargetName is for `/banana` and the second target name is for `/cherry`, then the end system must determine if the first signing key is trusted for `/apple` and the second signing key is trusted for `/banana` and the final Data object's signing key is trusted for `/cherry`.

1. Benefits
   (a) Each encapsulation only needs to sign the size of a Link. If the Link includes a TargetHash, then there is a strong trust chain.
   (b) Normally, we except only one Link followed by the Data.

```
LinkConcat := FixedHeader [OptionalHeaders]
              Link *ConCat
ConCat :=  *Link Data
Link := LinkMessage [Validation]
Data := DataMessage [Validation]
LinkMessage := <PayloadType = LINK>
DataMessage := <PayloadType = DATA>
Validation := ValidationAlg ValidationPayload
FixedHeader := <PacketType = ContentObject>
OptionalHeaders := <as per spec>
ValidationAlg := <as per spec>
ValidationPayload := <as per spec>
```

Figure 1. Simple Link Concatenation

```
LinkConcat := FixedHeader [OptionalHeaders]
              1*LinkMessage [Validation]
              Data
```

Figure 2. Aggregated Link Concatenation

   (c) The outer Link message could have its own TLVs to control caching or other behaviors.
2. Drawbacks
   (a) Sending a long link chain may exceed size restrictions.
   (b) For hash-based chains, each link will be at least 48 bytes plus the name length plus the Validation length. For a 2048-bit RSA signature, that's a 256 byte signature for each link. Each link could easily exceed 348 bytes.

1.2 Aggregated Signing Concatenation

The Aggregated Signing Concatenation approach sends a chain of links, but one one Validation section for all links. It must be signed by a key trusted for the initial Link Name namespace. The final Data object should be signed by a key trusted for its namespace, or follow a hash chain from the terminal Link.

1. Benefits
   (a) The trusted authority answering the first Interest aggregates all links to one signature to reduce size and verification complexity.
   (b) As in Simple Concatenation, it could reduce the Link chain to a single Link followed by Data.
2. Drawbacks
   (a) One loses the security context chain that lead one to trust the final object. One must trust the answering system has decided the Data object is the right object.

1.3 Separate Messages

The Separate Messages approach sends each Link and the final Data object as independent CCNx messages, each with their own FixedHeader. The leading messages are marked to indicate the LinkTarget will arrive next. We do not think this is a practical solution, as it requires inter-packet state and would be subject to many of the perils of fragmentation, such as reassembly timeout and injection attacks.

2. Encapsulation

The Encapsulation approach wraps one CCNx message inside another message. Unlike Link methods, it may be applied to both Interest and ContentObject messages.

Fig. 3 shows the format for Encapsulation. The FixedHeader carries a PacketType that corresponds to the wrapped message, so one could encapsulate either an Interest or a ContentObject. The wrapper carries the TLV type T_ENCAP and inside has its own Wrap Name. The Wrapper Name is the CCNx name used by routing. There may also be a set of TLVs associated with the wrapper, such as cache control directives, which would be processed as normal by intermediate nodes. Inside the wrapper is the wrapped message: an Interest or ContentObject.

An intermediate node forwarding a encapsulated message only needs to operate on the outer Wrapper Name and associated outer TLVs, such as cache control. It does not need to inspect the inner wrapped message.

As with Link encapsulation, one should not cache the inner message separately from the outer wrapper because the outer wrapper does not follow normal routing, with the same possible exceptions for strongly named objects.

An end system receiving an Encapsulation message should verify the outer validation is trusted for the WrapperName namespace. If the wrapped message is signed, it should verify its signature and trust for the inner wrapped namespace.

Because the Wrapped signature is trusted for the wrapper namespace and covers the wrapped message, the wrapped message does not necessarily need its own signature and trust assertion. The trusted wrapper signer is asserting that the wrapped message satisfies an Interest or requests a wrapped response. A Link concatenation, unless using TargetHash, has no strong association between the initial Link Name and the trailing Data object. One must traverse either signature chains or hash chains in the Links and make a trust assertion at each step.

```
Encap := FixedHeader [OptionalHeaders] Wrapper
FixedHeader := <PayloadType = As Wrapped>
Wrapper := WrapperName [tlvs] Wrapped
          [Validation]
tlvs := <other TLVs in Wrapper context>
Wrapped := Interest / ContentObject
```

Figure 3. Encapsulation

1. Benefits
   (a) Allows encapsulation of different message types.
   (b) Because the Wrapper signature covers the encapsulated message, there is a strong trust assertion to the wrapped object.
2. Drawbacks
   (a) The wrapper Validation is over the entire contents, so there is a larger resigning than with Links.
   (b) As with Link concatenation, the message will expand by the size of the Wrapper Name and Wrapper signature. Because the wrapper signature covers the Wrapped message, in some cases one could strip the inner signature to reduce message bloat.

3. Conclusion

We have presented two general types of encapsulation: Link Concatenation and Encapsulation. Within Link Concatenation, we identified three methods. Simple Concatenation chains one or more Links, each with their own Validation, to a terminal Data object. Aggregated Concatenation only signs the chain of Links rather than having all links individually singed. Separate Message Concatenation uses independent serial CCNx messages to realize concatenation by adding some extra state to the leading messages and using inter-packet state at intermediate systems. The Encapsulation method uses an outer Wrapper with its own Name and Wrapper TLVs (such as cache control) to delivery an inner Wrapped message.

References

[1] NDN Project Team, "(PRELIMINARY DRAFT) NDN Technical Memo: LINK." http://named-data.net/downloads/memos/link.pdf.