Link Design

Spyros Mastorakis

March 31, 2015

1 Link Structure

Link is a data packet, whose content contains multiple pairs in the form of (alias, preference). It represents one or more namespace delegations. The link is defined as a new ContentType. This ContentType currently exists in the ndn-cxx library (i.e., link type from src/encoding/tlv.hpp).

Let's assume that we have files that are published under /net/ndnsim, but are hosted under /att/user/alex/net/ndnsim and /verizon/user/alex/net/ndnsim. The structure of the Link would be the following:

- **Name**: Name of the link (/net/ndnsim/LINK)
- **MetaInfo**: ContentType = LINK
- **Content**: Content in the form of (alias, preference) pairs

  (/att/user/alex/net/ndnsim,100)
  (/verizon/user/alex/net/ndnsim, 10)
- **Signature**: Signed by the publisher of the Link

2 Link Implementation

As far as the implementation is concerned, a Link class can be defined, which would inherit the Data class of ndn-cxx and offer some further functionalities as well. The pairs of (alias, preference) would be stored as a private variable named m_delegations and defined as a std::multiset <std::pair <name : Name, preference : int>>.

The basic methods of this class would be following:

- **addDelegation**: This method would accept a pair (delegation names, preference) and store this pair to the m_pairs multiset based on its preference value.
- **removeDelegation**: Remove a delegation pair based on a given name.
- **getDelegations**: Get the pairs of ¡Name, Preference¿ as a DelegationSet.
- **encodeContent**: This method would Encode Link into the content. Then, the LINK will be encoded as a Data packet.
- **wireDecode**: This method would first call parent's wireDecode method and then decode the payload part.

3 Modifications in the Interest class

Regarding the encapsulation of the link in an interest packet, our approach is to encapsulate it as a Block instance, not as a Link instance directly. In this way, the Link would be encoded once by the data consumer, who would then attach it to the interest packet. Thereafter, each router will decode this Block instance in order to get the Link instance and perform further processing.

The new methods that we are planning to introduce to the Interest class are the following:

- **getLink**: This method would convert the Block variable to an actual Link instance and it is about to be called by the routers.
- **setLink**: This method would attach the link as a Block to the interest packet. This method is supposed to be called by the consumer.
- **unsetLink**: This method would de-attach the existing link from the interest packet.

The wireEncode and wireDecode methods also need to be modified to account for the introduced link block.

4 LINK TLV wire format

The TLV wire format for the LINK object would be the following:

```
LinkContent ::= CONTENT-TYPE TLV-LENGTH
                  Delegation+

Delegation ::= LINK-DELEGATION-TYPE TLV-LENGTH
                  Preference
                  Name

Preference ::= LINK-PREFERENCE-TYPE TLV-LENGTH
                  nonNegativeInteger

                  where:

               LinkPreference = 30,
               LinkDelegation = 31,
```

5 Interest wire format

The wire format of the Interest would be the following:

```
Interest ::= INTEREST-TYPE TLV-LENGTH
                 Name
               Selectors?
                 Nonce
                Scope?
            InterestLifetime?
                 Link?
          SelectedDelegation?
```

where

The SelectedDelegation field will be used to carry the delegation namespace (its index inside the link object) chosen by a previous hop.

```
SelectedDelegation ::= SELECTED-DELEGATION-TYPE TLV-LENGTH
                              nonNegativeInteger
```

and

SelectedDelegation = 32

The value of the SelectedDelegation field will be 0 up to (link.getDelegations().size() - 1).