ccnd FIB and PIT data structure

Junxiao Shi, 2013-02-10, updated 2013-03-30 2014-01-18

This document attempts to find out how FIB and PIT are organized and accessed in ccnd. The investigation is based on CCNx 0.7.2.

# Structure

FIB and PIT are stored in two hashtables h->interest\_tab and h->nameprefix\_tab (h is ccnd\_handle).

## Interest hashtable

h->interest\_tab stores pending interests.

Key type is the interest message in ccnb format up to end of selectors, but excludes InterestLifetime and Nonce.

Mapped type is struct interest\_entry.

### Interest entry

An Interest entry represents one or more similar pending Interests. Two Interests are "similar" if their Name and all selectors are identical; they may differ in InterestLifetime and/or Nonce. Multiple similar Interests share the same Interest entry.

An Interest entry contains:

* time of first Interest (strategy.birth)
* time of most recent similar Interest (strategy.renewed)
* number of similar Interests (strategy.renewals)
* per-face PIT information (singly linked list of struct pit\_face\_item)
* strategy timer (strategy.ev): invoke strategy\_callout(CCNST\_TIMER) once at the predicted response time of the "best" upstream if it does not respond, so that the name prefix entry's predicted response time is adjusted up; this event is cancelled (in finalize\_interest) when this Interest is satisfied
* do\_propagate timer (ev): invoke do\_propagate each time a downstream or upstream times out, or every 600 seconds
* the Interest message, up to end of selectors

### Per-Face PIT information for downstream Face

A downstream Face (where Interest comes in) has a pit\_face\_item with CCND\_PFI\_DNSTREAM flag. It contains:

* time of receiving last Interest with a unique Nonce (renewed)
* Interest expiry time (expiry): renewed + InterestLifetime
* Nonce of the last similar Interest received from this Face
* PFI\_PENDING flag: an Interest is pending (related to face->pending\_interests)
* PFI\_SUPDATA flag: do not forward, because the Interest has the same Nonce as another downstream Face

### Per-Face PIT information for upstream Face

An upstream Face (where Interest propagates to) has a pit\_face\_item with CCND\_PFI\_UPSTREAM flag. It contains:

* time of sending last Interest (renewed)
* Interest expiry time (expiry): renewed + InterestLifetime
* Nonce of the last similar Interest sent through this Face
* PFI\_UPENDING flag: an Interest is sent, waiting for response
* PFI\_SENDUPST flag: (no permanent meaning; local usage in strategy\_callout to indicate this face has no expiry)
* PFI\_UPHUNGRY flag: if there're n unexpired downstreams, do\_propagate sends the Interest to at most n "expired" upstreams ("expired" means "no pending Interest"), and assign this flag to all other "expired" upstreams

## Name Prefix hashtable

h->nameprefix\_tab stores the forwarding table, and states of forwarding strategy.

Key type is one or more components in ccnb format. The XML representation of a hash key looks like:  
<Component ccnbencoding="text">PARC</Component><Component ccnbencoding="base64Binary">AAEC</Component>  
The actual key is the ccnb encoded blob of the above.

Mapped type is struct nameprefix\_entry.

### Name Prefix entry

struct nameprefix\_entry contains all information about a name prefix.

A tree structure is maintained based on Name hierarchy. npe->parent points to the parent node (Name prefix with one less component); npe->children is the number of children, but there's no way to list the children.

Fields:

* doubly linked list of Interest entries with this Name (ie\_head)  
  This is a list, rather than a single entry, because Interests may have same Name but different selectors
* singly linked list of Forwarding entries (forwarding)
* Face list (forward\_to, tap) and namespace-wide flags (flags) collected from FIB entries
* Face through which last matching ContentObject is received (src)
* previous src value (osrc)
* response time prediction (usec)

## Forwarding entry

struct ccn\_forwarding represents a forwarding entry.

Fields:

* FaceID
* flags, as described in *CCNx Face Management and Registration Protocol*
* expiry time, and whether expired (expired entry does not have FORW\_REFRESHED flag, and will be deleted during next run of age\_forwarding function)

## Forwarding list version

h->forward\_to\_gen indicates the version number of Name Prefix hashtable. It is incremented every time a prefix is registered or unregistered, and every time old forwarding entries are removed. npe->forwarding and npe->tap must have a correct version number in npe->fgen before usage.

update\_forward\_to function updates npe->forwarding and npe->tap according to forwarding entries on the npe and all its ancestors.

# Operations

## Name Prefix entry insert

A node and all its ancestors are created (nameprefix\_seek function) when registering a prefix (Forwarding entry is inserted into npe->forwarding), or when processing an Interest (Interest entry is linked on npe->ie\_head). A new node will inherit flags, src, osrc, and usec from its parent.

## Name Prefix hashtable cleanup

Periodically (check\_nameprefix\_entries function):

* nodes with no Interest entry, no Forwarding entry, and no children are deleted
* dead faces are removed from npe->forward\_to and npe->tap
* npe->src is cleared to CCN\_NOFACEID, and its previous value (might be CCN\_NOFACEID) is kept in npe->osrc

## Name Prefix entry lookup (longest prefix match)

Given a Name, to perform the longest prefix match, we do a lookup the Name (in CCNB) in the Name Prefix hashtable. If a node does not exist, we delete the last component in the Name, and repeat the lookup with the Name which is now one component shorter, until a node is found.

On incoming Interest processing path, nameprefix\_seek function combines longest prefix match and new node insertion. On incoming ContentObject processing path, match\_interests function does longest prefix match as part of Interest hashtable match.

## Interest entry insert

Interest entry is created when a non-similar Interest is received (in propagate\_interest function).

The actual creation is the hashtb\_seek function call, which performs an exact match internally, and only creates a new entry if there isn’t an Interest entry with same Name and selectors.

After that, link\_interest\_entry\_to\_nameprefix function is invoked to add pointers between Interest entry and Name Prefix entry of the same Name.

## Interest entry delete

Interest entry is destroyed (in consume\_interest function) when the set of similar Interests is consumed (either satisfied, or every upstream and downstream timed out).

## Interest entries match

Given a ContentObject (either incoming, or outgoing to a specific face), match\_interests function performs a longest prefix match on the Name Prefix hashtable. From the matched entry, it walks up the Name Prefix tree via npe->parent pointer, calls consume\_matching\_interests function on each npe along the way.

consume\_matching\_interests function visits each Interest entry associated with one npe, and determines whether each Interest entry could match the ContentObject, by comparing Interest selectors against the ContentObject with ccn\_content\_matches\_interest function. If a matching Interest is found, the ContentObject is appended to the sending queue of every pending downstream face listed as pit\_face\_item, if the same ContentObject is not already in that sending queue; in case we are processing an outgoing ContentObject (which happens in incoming Interest processing path with a hit in ContentStore; the matched ContentObject would go to the source of that Interest), only those Interest entries with a downstream face which is the outgoing face of current ContentObject would be considered.

If match\_interests function sees CCN\_FORW\_LOCAL flag along the way up, and the incoming face of the current ContentObject is not local, it returns an error. Otherwise, match\_interests function will continue up, and count the total number of Interest entries matched; if the total number of Interest entries matched it zero, the ContentObject is unsolicited, and forwarding will take appropriate action.