NDNLPv2

NDNLPv2 is a link protocol for Named Data Networking.

Goals

NDNLPv2 provides the following **features**:

- fragmentation and reassembly: fragment a network layer packet to fit in link MTU
- reliability: reduce packet loss
- failure detection: rapidly detect link failure and recovery
- integrity: prevent packet injection
- forwarding instruction: NACK, nexthop choice, cache control, etc
- packet information: for management and monitoring

NDNLPv2 is designed to be a **unified protocol** that can be used on all kinds of links, including but not limited to: UNIX sockets, Ethernet unicast/multicast, UDP unicast/multicast, TCP connections, WebSockets, etc.

NDNLPv2 protocol operates as a **link adaptation layer**; it is above link layer and below network layer. Please, do not call this "layer 2.5": there is no such notion in RFC protocols.

Different links need different features, or different designs of a feature. NDNLPv2 ensures **all features are optional** and can be turned on or off per-link. NDNLPv2 also allows different designs of a feature to be adopted per-link.

NDNLPv2 deprecates and replaces: original NDNLP (aka NDNLPv1), NDNLPv1 multicast extension, NDNLPv1-TLV, NDNLP-BFD, NFD LocalControlHeader.

NDNLP Packet Format

NDNLPv2 packet adopts a Type-Length-Value (TLV) structure similar to NDN Packet Format.

```
NdnlpPacket ::= NDNLP-PACKET-TYPE TLV-LENGTH
                  NdnlpHeader
                  NdnlpFragment?

NdnlpHeader ::= NDNLP-HEADER-TYPE TLV-LENGTH
                  NdnlpSequence?
                  NdnlpHeaderExtension*

NdnlpSequence ::= NDNLP-SEQUENCE-TYPE TLV-LENGTH
                    fixed-width unsigned integer

NdnlpFragment ::= NDNLP-FRAGMENT-TYPE TLV-LENGTH
                    byte+
```

Outermost packet transmitted on a NDNLPv2 link is NdnlpPacket. In addition, a host MUST also accept bare network packets (Interest and Data) on a NDNLPv2 link, which SHOULD be interpreted as a NdnlpPacket with empty header, and have the bare network packet as its NdnlpFragment. However, such packets could be dropped later in processing if the link configured to require a certain NDNLPv2 feature but a field is missing.

NdnlpSequence contains a sequence number that is useful to multiple features. This field is REQUIRED if any enabled feature is using sequence numbers, otherwise it's OPTIONAL. Bit width of the sequence is determined on a per-link basis; 8-octet is recommended for today's links. A host MUST generate consecutive sequence numbers for outgoing packets on the same face.

NdnlpFragment contains a fragment of one or more network layer packets. The fragmentation and reassembly feature defines how NdnlpFragment field is constructed and interpreted. When fragmentation and reassembly feature is disabled, the NdnlpFragment field contains a whole network layer packet. NdnlpFragment is OPTIONAL; NdnlpPacket without NdnlpFragment is an **IDLE packet**.

NdnlpHeaderExtension is a repeatable optional structure in NdnlpHeader. NDNLPv2 features MAY add new header field by extending the definition of NdnlpHeaderExtension. Header extension fields can appear in any order. Unless otherwise specified, the same extension field shall appear at most once at an NdnlpHeader.

If an incoming NdnlpPacket contains unknown fields, the receiver MUST drop the packet, but SHOULD NOT consider the link has an error. Note: if a field is recognized but the relevant feature is disabled, it's not an "unknown field".

Indexed Fragmentation

Indexed fragmentation provides fragmentation and reassembly feature on datagram links that does not guarantee in-order delivery.

This feature defines two header fields:

```
NdnlpHeaderExtension ::= .. | NdnlpFragIndex | NdnlpFragCount

NdnlpFragIndex ::= NDNLP-FRAG-INDEX-TYPE TLV-LENGTH
                   nonNegativeInteger

NdnlpFragCount ::= NDNLP-FRAG-COUNT-TYPE TLV-LENGTH
                   nonNegativeInteger
```

Sender slices a network layer packet into one or more fragments. The size of each fragment MUST be small enough so that the NdnlpPacket carrying every fragment is below link MTU. It is RECOMMENDED that all except the last fragments have the same size.

NdnlpFragCount field indicates the number of fragments belonging to the same network layer packet. It MUST be the same in all fragments belonging to the same network layer packet.

NdnlpFragIndex field indicates the zero-based index of the current packet. It MUST be assigned consecutively for fragments belonging to the same network layer packet, starting from zero. The feature is named "indexed fragmentation" because every fragment is given an index in this field.

NdnlpSequence field is REQUIRED when this feature is enabled. Fragments belonging to the same network layer packet MUST be assigned consecutive sequence numbers, in the same order with NdnlpFragIndex.

For example, a 5000-octet network layer packet may be sliced as illustrated:

```
+-------------+-------------+    +-------------+-------------+
| NDNLPv2     | Fragment    |    | NDNLPv2     | Fragment    |
| seq=8801    |             |    | seq=8802    |             |
| FragIndex=0 | [0:1400]    |    | FragIndex=1 | [1400:2800] |
```

```
| FragCount=4 |             |    | FragCount=4 |             |
+-------------+-------------+    +-------------+-------------+

+-------------+-------------+    +-------------+-------------+
| NDNLPv2     | Fragment    |    | NDNLPv2     | Fragment    |
| seq=8803    |             |    | seq=8804    |             |
| FragIndex=2 | [2800:4200] |    | FragIndex=3 | [4200:5000] |
| FragCount=4 |             |    | FragCount=4 |             |
+-------------+-------------+    +-------------+-------------+
```

Receiver stores fragments in a *PartialMessageStore* data structure, which is a collection of PartialMessages, indexed by MessageIdentifier=NdnlpSequence-NdnlpFragIndex. Since both NdnlpSequence and NdnlpFragIndex are assigned consecutively, MessageIdentifier should be the sequence number of the first fragment of a network layer packet. After collecting all fragments belonging to a network layer packet, the receiver joins them together, and delivers the complete network layer packet to upper layer.

The receiver SHOULD maintain a reassembly timer in each PartialMessage, which is reset each time a new fragment is received. If this timer expires, the PartialMessage is dropped. Default duration for this timer is 500ms.

If this feature is enabled but NdnlpFragIndex is missing, it is implied as zero. If this feature is enabled but NdnlpFragCount is missing, it is implied as one. If this feature is disabled but either header field is received, the packet MUST be dropped.

Unless otherwise specified, header extension fields from other features shall only appear on the first fragment. If a field appear on a non-first fragment, it MUST be ignored.

Network NACK

A network NACK is a forwarding instruction from upstream to downstream that indicates the upstream is unable to satisfy an Interest.

This feature defines a header field:

```
NdnlpHeaderExtension ::= .. | NdnlpNack

NdnlpNack ::= NDNLP-NACK-TYPE TLV-LENGTH
                Nack?

Nack ::= DuplicateNack | GiveUpNack

DuplicateNack ::= DUPLICATE-NACK-TYPE TLV-LENGTH(=0)

GiveUpNack ::= GIVE-UP-NACK-TYPE TLV-LENGTH(=0)
```

NdnlpNack header field indicates an Interest is a NACK, and is not a normal Interest. The receiver MUST NOT process the packet as an Interest.

A **reason element**, such as DuplicateNack or GiveUpNack, MAY appear under NdnlpNack to indicate why the NACK is transmitted. Although all defined reason elements are empty, a reason element to be defined in the future may be non-empty and carry an optional suggestion on what the downstream should do. A receiver MUST be prepared to process a NACK without a reason element. If a NACK contains an unknown reason element, the receiver MUST treat this NACK as a NACK without reason, and MUST NOT drop the packet.

```
+-------------------+---------------+
| NDNLPv2           | Interest      |
|                   | Name=/example |
| +-NdnlpNack-----+ | Nonce=35      |
| | DuplicateNack | |               |
| +---------------+ |               |
+-------------------+---------------+
```

DuplicateNack indicates the upstream has detected a duplicate Nonce in the Interest sent by the downstream.

GiveUpNack indicates the upstream has attempted to forward the Interest, but no Data can be retrieved after exhausting all available routes.

It's RECOMMENDED to enable this feature on every link. If this feature is disabled but NdnlpNack is received, the packet MUST be dropped.

NdnlpNack header field is permitted only on a NdnlpPacket carrying an Interest. When NdnlpNack appears on a NdnlpPacket carrying a network layer packet other than an Interest, the packet MUST be dropped.

Consumer Controlled Forwarding

Consumer controlled forwarding allows a local consumer application to explicitly specify the nexthop face to forward an Interest.

This feature defines a header field:

```
NdnlpHeaderExtension ::= .. | NextHopFaceId

NextHopFaceId ::= NEXT-HOP-FACE-ID-TYPE TLV-LENGTH
                  nonNegativeInteger
```

NextHopFaceId indicates the nexthop FaceId to which an Interest should be forwarded. A local consumer application MAY add this field to an NdnlpPacket carrying an Interest. The local forwarder SHOULD follow this instruction and forward the Interest to the specified nexthop. ContentStore lookup SHOULD be bypassed unless NextHopFaceId equals a special FaceId that represent the ContentStore.

This feature is designed to be used on local faces only. It SHOULD NOT be enabled on non-local faces. If this feature is enabled but NextHopFaceId refers to a non-existent face, the Interest SHOULD be processed as if there is no available route. If this feature is disabled but NextHopFaceId is received, the packet SHOULD be dropped, or this field MUST be ignored.

NextHopFaceId header field is permitted only on a NdnlpPacket carrying an Interest, from an application to the forwarder. When NextHopFaceId appears on a NdnlpPacket carrying a network layer packet other than an Interest, the packet MUST be dropped. When NextHopFaceId is received by an application from a forwarder, this field MUST be ignored.

Implementation note: Currently, NFD honors NextHopFaceId only if client-control strategy is chosen for the namespace. In addition, ContentStore lookup will not be bypassed. This limitation may be lifted in a future version of NFD.

Local Cache Policy

Local cache policy feature allows a local producer application to instruct ContentStore on whether and how to cache a Data packet.

This feature defines a header field:

```
NdnlpHeaderExtension ::= .. | CachingPolicy

CachingPolicy ::= CACHING-POLICY-TYPE TLV-LENGTH
                    NoCache

NoCache ::= NO-CACHE-TYPE TLV-LENGTH(=0)
```

CachingPolicy contains a sub-element that gives a suggestion to the ContentStore. The ContentStore MAY follow this suggestion.

A **policy element**, such as NoCache, MUST appear under CachingPolicy to give a suggestion to the ContentStore. Although all defined policy elements are empty, a policy element to be defined in the future may be non-empty and carry additional arguments. If CachingPolicy field contains an unknown policy element, the forwarder SHOULD drop the packet.

```
+-------------------+---------------+
| NDNLPv2           | Data          |
|                   | Name=/example |
| +-CachingPolicy-+ | Content=xxxx  |
| | NoCache       | | Signature=xx  |
| +---------------+ |               |
+-------------------+---------------+
```

NoCache indicates the ContentStore SHOULD NOT admit the Data packet.

This feature is designed to be used on local faces only. It SHOULD NOT be enabled on non-local faces. If this feature is disabled but CachingPolicy is received, this field MUST be ignored.

CachingPolicy header field is permitted only on a NdnlpPacket carrying a Data packet, from an application to the forwarder. When CachingPolicy header field appears on a NdnlpPacket carrying a network layer packet other than a Data packet, the packet MUST be dropped. When CachingPolicy is received by an application from a forwarder, this field MUST be ignored.

Incoming Face Indication

Incoming face indication feature allows the forwarder to inform local applications about the face on which a packet is received.

This feature defines a header field:

```
NdnlpHeaderExtension ::= .. | IncomingFaceId

IncomingFaceId ::= INCOMING-FACE-ID-TYPE TLV-LENGTH
                     nonNegativeInteger
```

IncomingFaceId contains the FaceId from which the network layer packet is received. When this feature is enabled,

the forwarder SHOULD attach this field to every network layer packet going to a local application, and indicate the FaceId on which this network layer packet is received by the forwarder. If a Data packet comes from the ContentStore, IncomingFaceId SHOULD contain a special FaceId that represents the ContentStore, rather than the FaceId on which this Data packet was originally received. Even if this feature is enabled, the application MUST be prepared to receive a packet without IncomingFaceId field.

This feature is designed to be used on local faces only. It SHOULD NOT be enabled on non-local faces.

IncomingFaceId header field is permitted only on a NdnlpPacket from the forwarder to an application. When IncomingFaceId is received by the forwarder from an application, this field MUST be ignored.

Type Code Assignments

type	code (decimal)	code (hexadecimal)
NdnlpPacket	100	0x64
NdnlpHeader	80	0x50
NdnlpSequence	81	0x51
NdnlpFragment	82	0x52
NdnlpFragIndex	83	0x53
NdnlpFragCount	84	0x54
NdnlpNack	85	0x55
DuplicateNack	86	0x56
GiveUpNack	87	0x57
NextHopFaceId	88	0x58
CachingPolicy	89	0x59
NoCache	90	0x5a
IncomingFaceId	91	0x5b