Network Working Group
Internet-Draft
Intended status: Standards Track
Expires: April 7, 2013

D. Kutscher
NEC
S. Farrell
E. Davies
Trinity College Dublin

October 4, 2012

The NetInf Protocol
draft-kutscher-icnrg-netinf-proto-00

Abstract

This document defines a conceptual protocol and corresponding node requirements for NetInf nodes in a NetInf network. A NetInf network offers an information-centric paradigm that supports the creation, location, exchange and storage of Named Data Objects (NDOs). NetInf nodes can provide different services to other NetInf nodes, e.g., forwarding requests for information objects, delivering corresponding response messages, name resolution services etc. This (abstract) protocol is intended to be run over some "convergence layer" that handles transport issues. Two "wire" formats are defined, one that uses HTTP for message transfer and one layered on UDP.

Status of this Memo

Copyright Notice

Table of Contents

1. Introduction

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. [RFC2119]

Syntax definitions in this memo are specified according to ABNF [RFC5234].

There is an open-source implementation available that implements (most of) this. See http://sourceforge.net/projects/netinf/ for code and http://village.n4c.eu/getputform.html for access to a test server.

2. Principles and Assumptions

A NetInf network provides an information-centric networking (ICN) environment in which units of data content can be identified and accessed using a URI-based naming scheme. NetInf nodes in a NetInf network support the creation, location, exchange and storage of these units of content. In order to support interoperable implementation of the NetInf design, [ref.netinf-db2] the following assumptions are made here:

- all nodes can take on all NetInf roles (but do not have to);

- as necessary, nodes may access a Name Resolution System (NRS) and/or a (possibly name based) message routing infrastructure for NetInf messages; and

- the NetInf protocol can be used directly to access content.

The NetInf protocol operates on Named Data Objects (see [ref.netinf-db2]) referred to as NDOs. An NDO is an ordered collection of octets associated with a name. The NetInf protocol is designed to cache, locate and transmit complete NDOs.

The NetInf protocol is specified so that NDOs can in principle be retrieved from nodes anywhere in the network to which messages can be routed. This routing is intended to be driven by the names of the NDOs, with the option to use an NRS, but this specification does not discuss how routing, nor calling to an NRS, is carried out. Routing will also depend on the underlying Convergence Layer protocol (see Section 3) in use at that node.

Nodes offering NetInf services may return locators in some cases. These locators designate network locations where an NDO might

potentially be available for retrieval, but locators may not be usable outside of some (possibly hard-to-characterise) domain, or for more than a limited period of time, due to mobility of nodes or time limited access through "pinholes" in middleboxes such as firewalls and Network Address Translators (NATs). Accordingly, a design goal is to enable preferential use of names, with locators mostly as hints to improve efficiency. For this reason one can argue that locators ought not be made available to applications using the NetInf protocol in a form that would allow them to try to use the locator outside the NetInf protocol. NDOs may have multiple locators in order to indicate specific interfaces or to reflect attachment to multiple addressing domains. Locators also typically map to a specific instance (copy) of an NDO residing at a given host.

Locators are an example of NDO associated data that may be stored in association with the data content of the NDO. Other types of such data include "metadata" relating to the data content of the NDO, information describing the history of the copy of the NDO in the node where it is stored and search terms that are applicable to the NDO content. The term "affiliated data" will be used to describe the overall set of data, other than the actual octets of the content, stored or transmitted in association with an NDO. This affiliated data could be described as metadata of the NDO but we will reserve that term for a subset of the affiliated data that is usually constructed by the publisher of the NDO describing the content data of the NDO that is sent out in tandem with the data content. The NetInf protocol allows this affiliated data to be transmitted, in whole or in part, in association with NetInf messages.

NDO names will often be based on hash-function output values, and since the preferred hash-function will change over time and may change depending on location, this implies that NDOs can also have more than one name. There may also be cases where truncated hash values are desired (e.g., in cases where packets must be kept below some small size and fitting an entire request into one packet is required), and in such cases collisions will occur, or can easily be generated by bad actors. There are also cases where it is desirable to use a name to refer to some "dynamic" NDO, whose octets change (e.g., perhaps the current weather report) and there are cryptographic methods for doing this. This all means that there is no strict 1:1 mapping between names and NDOs, however, we do expect that for most objects, for most ICN deployments, there will in practice be one NDO that is named by each name. That is, each name usually does refer to just one object, and this protocol is designed to work best for that case.

The following NetInf services are assumed to be implemented on nodes through the NetInf protocol:

- caching of NDOs, both locally originated and acquired through network operations with the NetInf protocol;
- requesting the fetching of an NDO using its name, possibly with the addition of a locator, from elsewhere in the network;
- responding to NetInf protocol NDO fetch operations using a name referring to one of its locally known NDOs, which may have been locally generated or acquired from another NetInf node and cached here, by returning either or both of the data named in the operation or affiliated data including locator(s) referring to a node where that NDO is (assumed to be) available;
- initiating a search for NDOs matching specified search criteria;
- responding to search requests received by determining if any locally known NDOs meet the search criteria according to locally determined algorithms;
- NDO publication via sending out the name and, optionally, either or both of the content data and some affiliated data, such as locators, to other nodes;
- according to locally determined policy, the ability to accept or reject NDO publication requests that are delivered to the node, and to cache either or both of the objects and/or information about those that are accepted;
- according to locally determined policy, after carrying out local processing, the ability to forward NetInf messages to other nodes or discard them;
- managing the data affiliated with the NDO as well as the content data; and
- local cache management, driven by local policy and (optionally) whatever cache directives are carried in NetInf messages.

3. Convergence Layer Architecture

The idea of the Convergence Layer (CL) is to provide a means to transport NetInf messages between pairs of nodes that offer NetInf services. Any protocol that allows NetInf messages to be passed without loss of information can be used as a NetInf Convergence Layer (NetInf-CL) protocol.

This document does not cover the bit-level specification of any CL

protocol. The individual CL protocols will provide their own specification regarding their bit-level format.

Different CLs can be used in the various regions forming a global NetInf network. Where a message has to pass through several intermediate NetInf-capable nodes from source to destination, the NetInf protocol layer at each node is responsible for selecting the appropriate link and CL to forward messages.

Each CL has to offer the following minimal set of capabilities:

- unidirectional point-to-point transport of NetInf messages from source to destination,
- preservation of message boundaries,
- reliable transmission of message octets, and
- in-order delivery of message octets to the destination node.

If an underlying protocol used by a particular CL cannot offer these capabilities natively, then the CL is responsible for synthesising these capabilities by appropriate means, e.g., use of retransmission or insertion of sequence numbers. However, this does not prevent a CL that uses a more capable underlying protocol from implementing additional capabilities, e.g., bidirectional connections that allow a single connection to send NDOs in both directions.

The CL itself does not specify the properties of the messages, how they are interpreted, and the way nodes should interact with them, as that is what is specified in the present document.

The CL architecture is inspired by, and similar to, the concept used in Delay-Tolerant Networking. [RFC4838][RFC5050].

However, in contrast to DTN-CLs, the NetInf-CL concept does not include the handling of fragments of an NDO "above" the CL. This is the main difference between the CL concept as used in DTNs and ICNs. Put another way, a DTN-CL may result in a bundle being fragmented, and those fragments are only re-assembled at the final bundle destination. In the case of an NetInf-CL, if an NDO is fragmented or chunked within the CL, then those fragments or chunks are reassembled at the next ICN node and that fragmentation or chunking is not visible to the ICN protocol. One can also consider that the DTN Bundle Protocol (BP)[RFC5050], which runs over a DTN-CL, can itself, with an appropriate extension such as the "BPQ" extension, [I-D.farrell-dtnrg-bpq] be an NetInf-CL. That is, a concrete instance of this protocol could use the BP with the BPQ extension as

an NetInf-CL.

4. The NetInf Protocol - Overview

This protocol assumes that NDOs are named using URIs, and in particular via the "ni" URI scheme [I-D.farrell-decade-ni] which MUST be supported. There are a set of extensions to the "ni" URI scheme [I-D.hallambaker-decade-ni-params] that MAY be supported by nodes. However, other URI forms MAY also be used in the NetInf protocol, in particular as locators, and nodes SHOULD support at least fetching of "http" URLs.

Nodes are assumed to be capable of discriminating between names and locators, based on the URI scheme or otherwise.

The most common operations for a NetInf node will be fetching (using a GET message) an NDO or responding to such queries. The response to the GET message will, if possible, contain the octets making up the specified NDO and MAY contain

- o one or more URIs (typically locators) that could subsequently be used to retrieve the octets of the NDO either via this NetInf protocol or by alternative, locator-specific, means, and/or

- o other affiliated data such as metadata relevant to the NDO.

There are some circumstances in which it MAY be appropriate for the response to the GET message to contain only one or more locators and, optionally, other affiliated data. Examples of this situation occur if the responding node is aware that the object content can be returned more effectively using an alternative protocol or from an alternative source because of bandwidth limitations on the links connecting the responding node.

In addition to GET, there is the analagous PUBLISH operation where one node sends URIs and/or NDO octets to another. There is also a SEARCH operation, where one node submits a search query and receives a set of URIs and optional meta-data in response.

GET, PUBLISH and SEARCH messages MAY be forwarded by any node that receives them if there is good reason and local policy indicates that this would not result in excessive usage of network resources.

If a request message is forwarded, then a response message MUST NOT be sent for that request while the overall "transaction" is still in progress. That is, a node that forwards a request does not answer that request itself until it gets an answer from elsewhere.

Response messages MUST be forwarded by routers to the node from which the corresponding request message was received. The routing mechanisms that are used to ensure responses are correctly forwarded in this way are not specified here.

Since this specification does not determine how message routing, nor use of an NRS is done, we do not otherwise specify how or when messages are to be forwarded.

Nodes that want to make a locally stored NDO available with a specific name can use the PUBLISH message to announce that data to the network. This message MAY "push" the octets of the NDO into other nodes' caches. (If those nodes are willing to take them.) The reasoning behind this is that in many circumstances pushing just a name or a locator will not be helpful because the node with the NDO may be located behind a middlebox that will not allow access to the data from "outside." Pushing the complete NDO to a node that is accessible from the originating node but is also accessible from outside the middlebox "interior," can allow global access, e.g., by caching the NDO on a server in the DMZ ("DeMilitarized Zone") of an enterprise network or in a server provided by a home user's ISP.(Internet Service Provider). The publisher MAY also push affiliated data for the NDO, including additional locators and content metadata that can be stored in a node's NDO cache. The caching node MAY choose to store just the affiliated data without the content data depending on local policy.

As in the case of routing messages generally, this specification does not determine the node(s) to which an NDO can be "pushed."

Finally, NetInf nodes can send a SEARCH message to other NetInf nodes. In response, a NetInf node can perform a local search (i.e., of its local cache) As a response, any of the NetInf nodes that receives the SEARCH message returns a set of "ni" URIs of objects matching the search query. It may also return other types of URI such as "http" URIs. Searching of a node's local cache is the main goal for the SEARCH operation, but if a set of nodes were to forward SEARCH messages, then a global search (e.g., a Google-like service) service could be offered.

NDOs together with any affiliated data are represented using MIME objects. [RFC2045]. Placing as much of the affiliated data linked to the NDO in a multipart MIME object along with the octets of the actual object allows for significant specification and code re-use. For example, we do not need to invent a new typing scheme nor any associated registration rules nor registries.

As an example we might have a MIME object of that is multipart/mixed

and contains image/jpeg and application/json body parts, with the named image in the former and loosely structured associated data in the latter. The "ni" scheme parameters draft discusses such examples. This means that the details of the verification of name-data integrity supported by the ni name scheme also depend on the MIME type(s) used.

MIME also simplifies the specification of schemes that make use of digital signatures, reusing techniques from existing systems including Secure MIME (S/MIME) [RFC5751]and the Cryptographic Message Syntax (CMS) [RFC5652].

Note that (as specified in [I-D.farrell-decade-ni]) two "ni" URIs refer to the same object when the digest algorithm and values are the same, and other fields within the URI (e.g., the authority) are not relevant. Two ni names are identical when they refer to the same object. This means that a comparison function for ni names MUST only compare the digest algorithms and values.

5. Protocol Details

We define the GET, PUBLISH and SEARCH messages in line with the above. GET and PUBLISH MUST be supported. SEARCH SHOULD be supported. Each message has an associated response.

This means that GET and PUBLISH MUST be implemented and SEARCH SHOULD be implemented. In terms of services, GET and PUBLISH SHOULD be operational but SEARCH MAY be turned off.

5.1. GET/GET-RESP

The GET message is used to request an NDO from the NetInf network. A node responding to the GET message would send a GET-RESP that is linked to the GET request using the msg-id from the GET message as the msg-id for corresponding GET-RESP messages if it has an instance of the requested NDO.

The "ni" form or URI MUST be supported. Other forms of URI MAY be supported.

The msg-id SHOULD be chosen so as to be highly unlikely to collide with any other msg-id and MUST NOT contain information that might be personally identifying, e.g., an IP address or username. A sufficiently long random string SHOULD be used for this.

The ext field is to handle future extensibility (e.g., for message authenticators) and allows for the inclusion of a sequence of type,

length value tuples. No extensions for GET messages are defined at this point in time.

```
get-req = GET msg-id URI [ ext ]
get-resp =  status msg-id [ 1*URI ] [ ext ] [ object ]

ext = json-coded-string
```

Figure 1: GET/GET-RESP Message Format

Any node that receives a GET message and does not have an instance of the NDO referenced in the message MUST either

o forward the message to another node, or

o generate a GET response message with an appropriate status code and the msg-id from the GET message as the response msg-id.

If the message is forwarded, the node SHOULD maintain state that will allow it to generate the GET response message if a matching response message is not received for forwarding within a reasonable period of time after the GET message was forwarded.

If the node has an instance of the NDO, the response MAY contain zero or more URIs that MUST be either locators for the specified object or else alternative names for that object. If the receiving node has a copy of the relevant object in its cache it SHOULD include the object in the response. Possible reasons for not including the object would include situations where the GET message was received via a low-bandwidth interface but where the node "knows" that returning a locator will allow the requestor faster access to the object octets. Alternatively, the node may only be maintaining the affiliated data for the NDO and not the content data if it has not yet received the content data or has discarded it due to cache size limitations.

The object MUST be encoded as a MIME object. If there is affiliated data linked to the object this MUST also be encoded using MIME and integrated with the object in a multipart/mixed MIME object.

If the receiving node does not have a cached copy of the object it MAY choose to forward the message depending on local policy. Such forwarding could be based on name-based routing, on an NRS lookup or other mechanisms (e.g. a node might have a default route).

If an get-resp is received with an object that is not MIME encoded or of an unknown MIME type then that MUST be treated as an application/

octet-stream for the purposes of name-data integrity verification.

get-resp messages MAY include extensions as with all others.

5.2. PUBLISH/PUBLISH-RESP

The PUBLISH message allows a node to push the name, and optionally, alternative names, locators, a copy of the object octets and/or object meta-data. Ignoring extensions, only a status code is expected in return.

A msg-id MUST be included as in a GET message.

A URI containing a name MUST be included. The "ni" URI scheme SHOULD be used for this name.

The message MAY also contain additional URIs that represent either alternative names or locators where the identical object can be found and metadata relating to the published content. As mentioned in Section 4 it is the responsibility of the receiving node to discriminate between those URIs used as names and those used as locators.

The object octets MAY be included. This is intended to handle the case where the publishing node is not able to receive GET messages for objects. An implementation SHOULD test (or "know") its local network context sufficiently well to decide if the object octets ought to be included or not. Methods for checking this are out of scope of this specification.

A node receiving a PUBLISH message chooses what information from the message, if any, to cache according to local policy and availability of resources. It is RECOMMENDED that a node that receives a PUBLISH message containing the object octets verify that the digest in the name under which the content is published matches with the digest of the received data.

One way to "fill a cache" if the object octets are not included in the PUBLISH would be for the recipient of the PUBLISH to simply request the object octets using GET and cache those. (There is no point in sending a PUBLISH without the octets and without any locator.) This behaviour is, of course, an implementation issue.

In some cases it may make sense for a (contactable) node to only publish the name and metadata about the object. The idea here is that the metadata could help with routing or name resolution or search. Since we are representing both NDO octets and affiliated data such as the metadata as MIME objects, we need to tell the

receiver of the PUBLISH message whether or not that message contains the full object. We do this via the "full-ndo-flag" which, if present, indicates that the PUBLISH message contains enough data so the receiver of the PUBLISH message has sufficient data to provide a complete answer a subsequent GET message for that name, i.e., data content and affiliated data.

If a node receives a PUBLISH message for an NDO which already exists in its cache, the received information SHOULD be used to complete or update the node's cached information for the NDO:

o If the object octets are included and the node currently does not have the octets cached, the data content MAY be added to the cache. Again it is RECOMMENDED that the received data has the correct digest as specified in the NDO name, and

o Items in the affiliated data MAY be merged into cached affiliated data, including adding additional locators to the list of known locators for the NDO and merging any content metadata with previously received metadata. If there is a conflict, the choice of metadata to be stored is a matter of policy.

It is RECOMMENDED that a timestamp be recorded whenever the cached information for an NDO is updated and that this timestamp be stored in the affiliated data and the most recent timestamp returned with any subsequent GET or SEARCH request that references the NDO.

Extensions ("ext") MAY be included as in a GET request. One such HTTP CL-specific extension ("meta") is defined in Section 6.1 below.

```
pub-req = PUBLISH msg-id 1*URI [ ext ] [ [ full-ndo-flag ] object ]
pub-resp = status msg-id [ ext ]
```

Figure 2: PUBLISH/PUBLISH-RESP Message Format

The response to a PUBLISH message is a status code and the msg-id from the PUBLISH message and optional extensions.

A node receiving a PUBLISH message MAY choose to forward the message to other nodes whether or not it chooses to cache any information. If this node does not cache the information but does forward the PUBLISH message, it should postpone sending a response message until a reasonable period of time has elapsed during which no other responses to the PUBLISH message are received for forwarding. However, the node MAY send an extra response message, even if it forwards the PUBLISH message, if the sender of the PUBLISH message

would have expected the receiving node to cache the object (e.g., because of a contractual relationship) but it was unable to do so for some reason.

5.3. SEARCH/SEARCH-RESP

The SEARCH message allows the requestor to send a set of query tokens containing search keywords. The response is either a status code or a multipart MIME object containing a set of metadata body parts, each of which MUST include a name for an NDO that is considered to match the query keywords.

```
search-req = SEARCH msg-id [ 1*token ] [ ext ]
search-resp = status msg-id [ results ] [ ext ]
```

Figure 3: SEARCH/SEARCH-RESP Message Format

In the case where the response contains results, these MUST take the form of an application/json MIME object containing an array of results. Each result MUST have a "name" field with a URI as the value of that field. Any other fields in array elements SHOULD contain metadata that is intended to allow the requestor to select which, if any, of the names offered to retrieve.

The URIs included in a search-resp SHOULD be names, but MAY be locators, to be distinguished by the requestor as in the case of GET responses.

The intent of the SEARCH message is to allow nodes to search one another's caches, but without requiring us to fix the details (ontology) for NDO content metadata. While this main intended use-case does not involve forwarding of SEARCH messages that is not precluded.

As with PUBLISH messages, if a SEARCH message is forwarded, the forwarding node postpones sending an empty SEARCH response until a reasonable time is elapsed to see if alternative node responds to the SEARCH.

If a SEARCH at a node identifies an NDO that is included in the results of a search, the tokens that were used for the search MAY be recorded in the affiliated data cached with the NDO. Each set of search tokens for which a "match" is obtained should be recorded separately resulting in an array of set of tokens. If the search mechanisms used provides a reliability measure, this MAY also be recorded and the measure may be used to limit the size of the search

tokens array by discarding (or never inserting) sets of tokens with low reliability scores.

SEARCH messages MAY include extensions as for other messages.

6. Convergence Layer Specifications

This section specifies two convergence layers that represent instantiations of the NetInf protocol. The first, based on HTTP, is intended for using NetInf in existing web infrastructures, whereas the second, based on UDP, provides an efficient datagram-based hop-by-hop message transport that can be used to query for GET requests sent to an NRS node or for multicasting such requests in a local network.

6.1. HTTP CL

The basic idea with the HTTP CL is to use forms for NetInf protocol requests and Multipart MIME HTTP response bodies for Netinf protocol responses. This has be done to allow web browsers to be able to easily interact with NetInf and because there are many tools available that make implementation relatively easy. Note thought that the NetInf HTTP CL is also intended for use between NetInf infrastructure nodes.

The HTTP CL assumes that the client knows the address of the HTTP server to which it will send requests. Clients MAY use the authority part of an ni URI, if one is present to select the HTTP responder. NetInf HTTP responders MUST accept requests sent to the following paths:

/netinfproto/get for NetInf GET requests

/netinfproto/publish for NetInf PUB requests

/netinfproto/search for NetInf SEARCH requests

So for example a client would send an HTTP request containing a NetInf GET to http://example.com/netinfproto/get

NetInf HTTP responders SHOULD also make ni URIs available at the relevant well-known URL [RFC5785] for the ni URI. [I-D.farrell-decade-ni]

NetInf protocol requests use forms. The mapping of the fields from the abstract protocol is as shown in Figure 4. [[NOTE: this is a bit inconsistent now, and just reflects SF's code.]]

Abstract Protocol Field	Form field	Comments (field type in form)
URI	urival, URI loc1,loc2	usually an ni URI (text) or locator
msg-id	msgid	a message identifier (text)
ext	ext	extension(s) (JSON encoded string)
full-ndo-flag	fullPut	true if object supplied (checkbox)
object	octets	object octets (file specification)
n/a	rform	response format required, can be "html" or "json" (radio)
token	tokens	one text field with all search keywords (text)

Figure 4: Form fields used in NetInf requests

Notes for Figure 4:

For GET messages:	"URI" and "msgid" are mandatory. "loc1" and "loc2" are optional. "ext" may be used in future but no values currently defined.
For PUBLISH messages:	"URI" and "msgid" are mandatory. "loc1", "loc2", "ext", "rform" and "fullPut" are optional. If "rform" is absent, the "json" value is assumed. If "fullPut" is absent, a "false" value is assumed. If "fullPut" is present and set to "true", "octets" must be present. If present, "octets" contains a file specification and the object octets. If present, "ext" may contain a "meta" item. The value of "ext" MUST be a JSON object string and the value of the "meta" item MUST be a (subsidiary) object, e.g., the "ext"

string might be
{ "meta": { "mi1": 5, "mi2": { ...}, "mi3": "abcd". "mi4": [...] }}

For SEARCH messages: "msgid" and "tokens" are mandatory. "rform" is optional. If "rform" is absent, the "json" value is assumed. "ext" may be used in future but no values currently defined.

HTTP responses for each command can differ.

For GET, the a successful HTTP response (HTTP response code 2xx) will contain either an application/json (if no object is returned) or else a multipart/mixed with two (and exactly two:-) body parts, the first being an application/json and the second containing the object octets, with whatever MIME type is appropriate.

The application/json component will consist of a JSON object that SHOULD contain the following named fields:

NetInf	A string describing the version of the NetInf protocol in use (e.g., "V0.1a").
ni	The "canonicalized" form of the NDO as a URI in the ni scheme: "canonicalized" means that the URI has empty netloc and query string fields. For example: "ni:///sha-256-64;gf2yhPY9Mu0" or "nih:/ sha256-32;81fdb284;d".
msgid	The value of the msgid field in the GET message that resulted in this response.
ts	The timestamp of the last update of the cached information in the cache from which the NDO is being sent.
status	A code, taken from the HTTP 2xx response codes indicating what has been returned (200 if both affiliated data and content has been returned and 203 if only affiliated data is returned).
ct	The MIME content type of the NDO content data, if known. Empty string if not yet known.

loclist	Array of locator names (strings) from where the NDO might potentially be retrieved.
metadata	A JSON object containing any named items copied in from "meta" object(s) supplied by any PUBLISH messages received at the node that sent the response plus an entry named "publish" which contains a string indicating the class of node and software that generated the cache entry.
searches	A JSON array of objects each containing a set of strings representing search tokens and information about the search mechanism that resulted in a match with the NDO during a previous search.

For PUBLISH, the HTTP response will contain an application/json or text/html response, depending on the value of the rform form field. (If rform is missing json is the default.) The application/json structure is as for a GET response. The text/html document will provide a report of the successful publication of the NDO and whatever other relevant information form the affiliated information seems appropriate for inspection by a human user.

For SEARCH, the HTTP response will contain an application/json or text/html response, depending on the value of the rform form field. (If rform is missing json is the default.) The application/json structure is similar to the previous structures, but has a "results" object that contains an array of object details.

6.2. UDP CL

The UDP CL implements the NetInf protocol with a UDP datagram services, i.e., all NetInf messages are mapped to individual UDP messages. The purpose is to provide a light-weight datagram-based CL that can be used to implement NetInf transport protocols on top and that can provide efficient communication for querying NRSs, and request broadcasting/multicasting. The UDP CL provides no hop-by-hop flow control, retransmission and fragmentation/re-assembly.

The UDP CL has two sending modes: 1) send to specified destination IP address and 2) send to the well-known IPv4 multicast address 225.4.5.6. For both unicast and multicast the UDP port number is 2345. All request and response messages are JSON objects, i.e., unordered sets of name/value pairs.

For UDP CL messages, the following JSON names for name/value pairs are defined (not all objects have to be present in all messages):

```
    version  # the NetInf UDP CL protocol version -- currently
             # "NetInfUDP/1.0"

    msgType  # the message type (e.g., GET)

    uri      # the NI URI

    msgId    # the message ID (must be unique per CL hop and
             #request/response pair)

    locators # an array of locators

    instance # an UDP CL speaker identifier (must be unique per IP host,
             # e.g., process ID and per process ID
```

Figure 5: UDP CL JSON request structure

This version of the specification defines the GET request and the corresponding GET response only.

GET request A GET request provides the following objects:

version:	"NetInfUDP/1.0"
msgType:	"GET"
uri:	name of the requested NDO
msgId:	message ID (see above)

GET reponse A GET response provides the following objects:

version:	"NetInfUDP/1.0"
msgType:	"GET-RESP"
uri:	name of the requested NDO
msgId:	message ID (see above)
locators:	a list of locator strings

7. Security Considerations

For privacy preserving reasons requestors SHOULD attempt to limit the personally identifying information (PII) included with search requests. Including fine-grained search keywords can expose requestor PII. For this reason, we RECOMMEND that requestors include more coarse grained keywords and that responders include sufficient meta-data to allow the requestor to refine their search based on the meta-data in the response.

Similarly, search responders SHOULD consider whether or not they respond to all or some search requests as exposing one's cached content can also be a form of PII if the cached content is generated at the behest of the responder.

Name-data integrity validation details are TBD for some common MIME types.

Users need to be aware that the affiliated data is NOT protected by the name-data integrity as this applies only to the data content octets.

[[More TBD no doubt.]]

8. Acknowledgements

This work has been supported by the EU FP7 project SAIL.

Claudio Imbrenda and Christian Dannewitz contributed to early versions of this document whilst working at NEC and the University of Paderborn respectively.

9. References

9.1. Normative References

[I-D.farrell-decade-ni]
Farrell, S., Kutscher, D., Dannewitz, C., Ohlman, B., Keranen, A., and P. Hallam-Baker, "Naming Things with Hashes", draft-farrell-decade-ni-10 (work in progress), August 2012.

[I-D.hallambaker-decade-ni-params]
Hallam-Baker, P., Stradling, R., Farrell, S., Kutscher, D., and B. Ohlman, "The Named Information (ni) URI Scheme: Optional Features", draft-hallambaker-decade-ni-params-03 (work in progress), June 2012.

[RFC2045] Freed, N. and N. Borenstein, "Multipurpose Internet Mail Extensions (MIME) Part One: Format of Internet Message Bodies", RFC 2045, November 1996.

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

[RFC5234] Crocker, D. and P. Overell, "Augmented BNF for Syntax Specifications: ABNF", STD 68, RFC 5234, January 2008.

[RFC5652] Housley, R., "Cryptographic Message Syntax (CMS)", STD 70, RFC 5652, September 2009.

[RFC5785] Nottingham, M. and E. Hammer-Lahav, "Defining Well-Known Uniform Resource Identifiers (URIs)", RFC 5785, April 2010.

9.2. Informative References

[I-D.farrell-dtnrg-bpq]
Farrell, S., Lynch, A., Kutscher, D., and A. Lindgren, "Bundle Protocol Query Extension Block", draft-farrell-dtnrg-bpq-01 (work in progress), March 2012.

[RFC4838] Cerf, V., Burleigh, S., Hooke, A., Torgerson, L., Durst, R., Scott, K., Fall, K., and H. Weiss, "Delay-Tolerant Networking Architecture", RFC 4838, April 2007.

[RFC5050] Scott, K. and S. Burleigh, "Bundle Protocol Specification", RFC 5050, November 2007.

[RFC5751] Ramsdell, B. and S. Turner, "Secure/Multipurpose Internet Mail Extensions (S/MIME) Version 3.2 Message Specification", RFC 5751, January 2010.

[ref.netinf-db2]
SAIL, "NetInf Content Delivery and Operations", SAIL Project Deliverable D-3.2 , May 2012.

Authors' Addresses

Dirk Kutscher
NEC
Kurfuersten-Anlage 36
Heidelberg,
Germany

Phone:
Email: kutscher@neclab.eu

Stephen Farrell
Trinity College Dublin
Dublin, 2
Ireland

Phone: +353-1-896-2354
Email: stephen.farrell@cs.tcd.ie

Elwyn Davies
Trinity College Dublin
Dublin, 2
Ireland

Phone: +44 1353 624 579
Fax:
Email: davieseb@scss.tcd.ie
URI: