                        P2P Overlay Diagnostics
                      draft-ietf-p2psip-diagnostics-19

Abstract

   This document describes mechanisms for P2P overlay diagnostics. It
   defines extensions to the RELOAD P2PSIP base protocol to collect
   diagnostic information, and details the protocol specifications for
   these extensions. Useful diagnostic information for connection and
   node status monitoring is also defined. The document also describes
   the usage scenarios and provides examples of how these methods are
   used to perform diagnostics in P2PSIP overlay networks.

Status of This Memo

Copyright Notice

Table of Contents

1. Introduction

   In the last few years, overlay networks have rapidly evolved and
   emerged as a promising platform for deployment of new applications
   and services in the Internet. One of the reasons overlay networks
   are seen as an excellent platform for large scale distributed systems
   is their resilience in the presence of failures. This resilience has
   three aspects: data replication, routing recovery, and static
   resilience. Routing recovery algorithms are used to repopulate the
   routing table with live nodes when failures are detected. Static
   resilience measures the extent to which an overlay can route around
   failures even before the recovery algorithm repairs the routing
   table. Both routing recovery and static resilience rely on accurate
   and timely detection of failures.

   There are a number of situations in which some nodes in a Peer-to-
   Peer (P2P) overlay may malfunction or behave badly. For example,
   these nodes may be disabled, congested, or may be misrouting
   messages. The impact of these malfunctions on the overlay network
   may be a degradation of quality of service provided collectively by
   the peers in the overlay network or an interruption of the overlay
   services. It is desirable to identify malfunctioning or badly
   behaving peers through diagnostic tools, and exclude or reject them
   from the P2P system. Node failures may also be caused by failures of
   underlying layers. For example, recovery from an incorrect overlay
   topology may be slow when the speed at which IP routing recovers

after link failures is very slow. Moreover, if a backbone link fails
and the failover is slow, the network may be partitioned, leading to
partitions of overlay topologies and inconsistent routing results
between different partitioned components.

Some keep-alive algorithms based on periodic probe and acknowledge
mechanisms enable accurate and timely detection of failures of one
node's neighbors [Overlay-Failure-Detection], but these algorithms by
themselves can only detect the disabled neighbors using the periodic
method. This may not be sufficient for the service provider
operating the overlay network.

For Peer-to-Peer SIP (P2PSIP), a single, general P2PSIP overlay
diagnostic framework supporting periodic and on-demand methods for
detecting node failures and network failures is desirable. This
document describes a general P2PSIP overlay diagnostic extension to
the P2PSIP base protocol RELOAD [RFC6940] and is intended as a
complement to keep-alive algorithms in the P2PSIP overlay itself.
Readers are advised to consult [I-D.ietf-p2psip-concepts] for further
background on the problem domain.

2. Terminology

This document uses the concepts defined in RELOAD [RFC6940].

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 [RFC2119].

3. Diagnostic Scenarios

P2P systems are self-organizing and ideally setup and configuration
of individual P2P nodes requires no network management in the
traditional sense. However, users of an overlay, as well as P2P
service providers may contemplate usage scenarios where some
monitoring and diagnostics are required. We present a simple
connectivity test and some useful diagnostic information that may be
used in such diagnostics.

The common usage scenarios for P2P diagnostics can be broadly
categorized in three classes:

a. Automatic diagnostics built into the P2P overlay routing
    protocol. Nodes perform periodic checks of known neighbors and
    remove those nodes from the routing tables that fail to respond
    to connectivity checks [Handling_Churn_in_a_DHT]. Unresponsive
    nodes may only be temporarily disabled, for example due to a
    local cryptographic processing overload, disk processing overload

or link overload. It is therefore useful to repeat the
connectivity checks to see nodes have recovered and can be again
placed in the routing tables. This process is known as 'failed
node recovery' and can be optimized as described in the paper
"Handling Churn in a DHT" [Handling_Churn_in_a_DHT].

b. Diagnostics used by a particular node to follow up on an
individual user complaint or failure. For example, a technical
support staff member may use a desktop sharing application (with
the permission of the user) to remotely determine the health of,
and possible problems with, the malfunctioning node. Part of the
remote diagnostics may consist of simple connectivity tests with
other nodes in the P2PSIP overlay and retrieval of statistics
from nodes in the overlay. The simple connectivity tests are not
dependent on the type of P2PSIP overlay. Note that other tests
may be required as well, including checking the health and
performance of the user's computer or mobile device and checking
the bandwidth of the link connecting the user to the Internet.

c. P2P system-wide diagnostics used to check the overall health of
the P2P overlay network. These include checking the consumption
of network bandwidth, checking for the presence of problem links
and checking for abusive or malicious nodes. This is not a
trivial problem and has been studied in detail for content and
streaming P2P overlays [Diagnostic_Framework], and has not been
addressed in earlier P2PSIP documents
[Diagnostics_and_NAT_traversal_in_P2PP]. While this is a
difficult problem, a great deal of information that can help in
diagnosing these problems can be obtained by obtaining basic
diagnostic information for peers and the network. This document
provides a framework for obtaining this information.

4. Data Collection Mechanisms

4.1. Overview of Operations

The diagnostic mechanisms described in this document are primarily
intended to detect and locate failures or monitor performance in
P2PSIP overlay networks. It provides mechanisms to detect and locate
malfunctioning or badly behaving nodes including disabled nodes,
congested nodes and misrouting peers. It provides a mechanism to
detect direct connectivity or connectivity to a specified node, a
mechanism to detect the availability of specified resource records
and a mechanism to discover P2PSIP overlay topology and the underlay
topology failures.

The P2PSIP diagnostics extensions define two mechanisms to collect
data. The first is an extension to the RELOAD Ping mechanism,

allowing diagnostic data to be queried from a node, as well as to
diagnose the path to that node. The second is a new method and
response, PathTrack, for collecting diagnostic information
iteratively. Payloads for these mechanisms allowing diagnostic data
to be collected and represented are presented, and additional error
codes are introduced. Essentially, this document reuses RELOAD
[RFC6940]specification and extends them to introduce the new
diagnostics methods. The extensions strictly follow how RELOAD
specifies message routing, transport, NAT traversal, and other RELOAD
protocol features. The diagnostic methods are however P2PSIP
protocol independent.

This document primarily describes how to detect and locate failures
including disabled nodes, congested nodes, misrouting behaviors and
underlying network faults in P2PSIP overlay networks through a simple
and efficient mechanism. This mechanism is modeled after the ping/
traceroute paradigm: ping [RFC0792] is used for connectivity checks,
and traceroute is used for hop-by-hop fault localization as well as
path tracing. This document specifies a "ping-like" mode (by
extending the RELOAD Ping method to gather diagnostics) and a
"traceroute-like" mode (by defining the new PathTrack method) for
diagnosing P2PSIP overlay networks.

One way these tools can be used is to detect the connectivity to the
specified node or the availability of the specified resource-record
through the extended P2PSIP Ping operation. Once the overlay network
receives some alarms about overlay service degradation or
interruption, a Ping is sent. If the Ping fails, one can then send a
PathTrack to determine where the fault lies.

The diagnostic information can only be provided to authorized nodes.
Some diagnostic information can be provided to all the participants
in the P2PSIP overlay, and some other diagnostic information can only
be provided to the nodes authorized by the local or overlay policy.
The authorization depends on the type of the diagnostic information
and the administrative considerations, and is application specific.

This document considers the general administrative scenario based on
diagnostic kind type, where a whole overlay can authorize a certain
type of diagnostic information to a small list of particular nodes
(e.g. administrative nodes). That means, if a node gets the
authorization to access a diagnostic kind type, it can access that
information from all nodes in the overlay network. It leaves the
scenario where a particular node authorizes its diagnostic
information to a particular list of nodes out of scope. This could
be achieved by extension of this document if there is requirement in
the near future. The default policy or access rule for a type of
diagnostic information is "permit" unless specified in the

diagnostics extension document. As the RELOAD protocol already
requires that each message carries the message signature of the
sender, the receiver of the diagnostics requests can use the
signature to identify the sender. It can then use the overlay
configuration file with this signature to determine which types of
diagnostic information that node is authorized for.

In the remainder of this section we define mechanisms for collecting
data, as well as the specific protocol extensions (message
extensions, new methods, and error codes) required to collect this
information. In Section 5 we discuss the format of the data
collected, and in Section 6 we discuss detailed message processing.

4.2. "Ping-like" Behavior: Extending Ping

To provide "ping-like" behavior, the RELOAD Ping method is extended
to collect diagnostic data along the path. The request message is
forwarded by the intermediate peers along the path and then
terminated by the responsible peer. After optional local
diagnostics, the responsible peer returns a response message. If an
error is found when routing, an Error response is sent to the
initiator node by the intermediate peer.

The message flow of a Ping message (with diagnostic extensions) is as
follows:

```
 Peer A                 Peer B                 Peer C                 Peer D
   |                      |                      |                      |
   |(1). PingReq          |                      |                      |
   |--------------------->|(2). PingReq          |                      |
   |                      |--------------------->|(3). PingReq          |
   |                      |                      |--------------------->|
   |                      |                      |                      |
   |                      |                      |<---------------------|
   |                      |<---------------------|(4). PingAns          |
   |<---------------------|(5). PingAns          |                      |
   |(6). PingAns          |                      |                      |
   |                      |                      |                      |
```
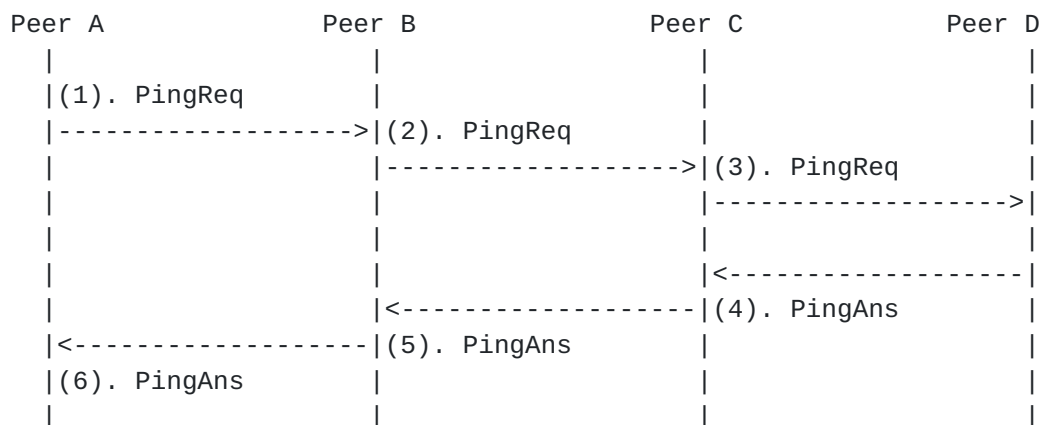
Figure 1: Ping Diagnostic Message Flow

4.2.1. RELOAD Request Extension: Ping

To extend the ping request for use in diagnostics, a new extension of
RELOAD is defined. The structure for a MessageExtension in RELOAD is
defined as:

```
          struct {
            MessageExtensionType  type;
            Boolean               critical;
            opaque                extension_contents<0..2^32-1>;
          } MessageExtension;
```

For the Ping request extension, we define a new MessageExtensionType,
extension 0x0002 named Diagnostic_Ping, as specified in Table 4. The
extension contents consists of a DiagnosticsRequest structure,
defined later in this document in Section 5.1. This extension MAY be
used for new requests of the Ping method and MUST NOT be included in
requests using any other method.

This extension is not critical. If a peer does not support the
extension, they will simply ignore the diagnostic portion of the
message, and will treat the message as if it was a normal ping.
Senders MUST accept a response that lacks diagnostic information and
SHOULD NOT resend the message expecting a reply. Receivers who
receive a method other than Ping including this extension MUST ignore
the extension.

4.3. "Traceroute-like" Behavior: The Path_Track Method

We define a simple PathTrack method for retrieving diagnostic
information iteratively.

The operation of this request is shown below in Figure 2. The
initiator node A asks its neighbor B which is the next hop peer to
the destination ID, and B returns a message with the next hop peer C
information, along with optional diagnostic information for B to the
initiator node. Then the initiator node A asks the next hop peer C
(directly or via symmetric routing) to return next hop peer D
information and diagnostic information of C. Unless a failure
prevents the message from being forwarded, this step can be
iteratively repeated until the request reaches responsible peer D for
the destination ID, and retrieves diagnostic information of peer D.

The message flow of a PathTrack message (with diagnostic extensions)
is as follows:

```
   Peer-A                Peer-B                Peer-C                Peer-D
     |                     |                     |                     |
     |(1).PathTrackReq     |                     |                     |
     |-------------------->|                     |                     |
     |(2).PathTrackAns     |                     |                     |
     |<--------------------|                     |                     |
     |                     |(3).PathTrackReq     |                     |
     |-------------------|-------------------->|                     |
     |                     |(4).PathTrackAns     |                     |
     |<------------------|-------------------|                     |
     |                     |                     |(5).PathTrackReq     |
     |------------------|-------------------|-------------------->|
     |                     |                     |(6).PathTrackAns     |
     |<------------------|-------------------|-------------------|
     |                     |                     |                     |
```
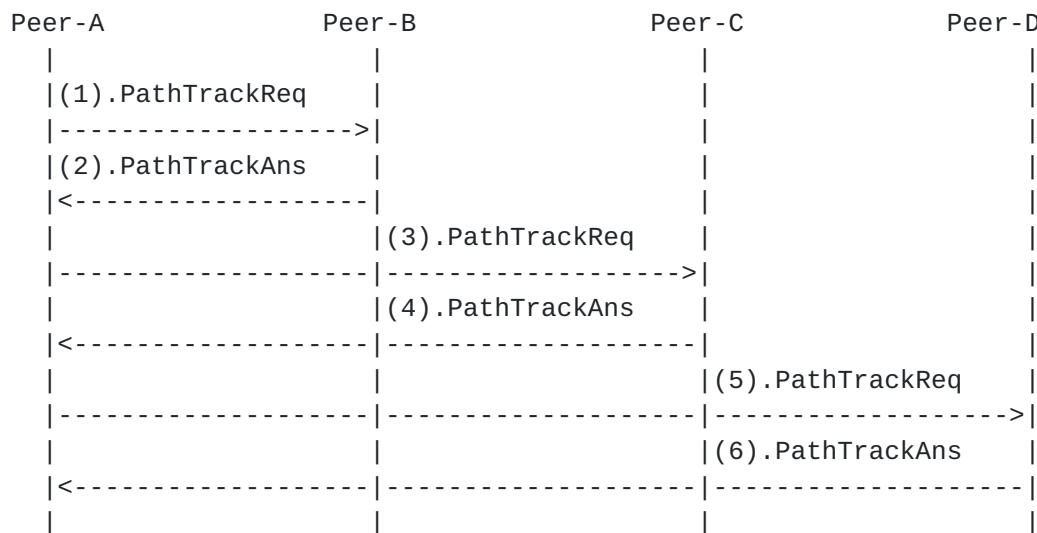
Figure 2: PathTrack Diagnostic Message Flow

There have been proposals that RouteQuery and a series of Fetch
requests can be used to replace the PathTrack mechanism, but in the
presence of high rates of churn, such an operation would not,
strictly speaking, provide identical results, as the path may change
between RouteQuery and Fetch operations. While obviously the path
could change between steps of PathTrack as well, with a single
message rather than two messages for query and fetch, less
inconsistency is likely, and thus the use of a single message is
preferred.

Given that in a typical diagnostic scenario the peer sending the
PathTrack request desires to obtain information about the current
path to the destination, in the event that succesive calls to
PathTrack return different paths, the results should be discarded and
the request resent, ensuring that the second request traverses the
appropriate path.

4.3.1. New RELOAD Request: PathTrack

This document defines a new RELOAD method, PathTrack, to retrieve the
diagnostic information from the intermediate peers along the routing
path. At each step of the PathTrack request, the responsible peer
responds to the initiator node with requested status information.
Status information can include a peer's congestion state, processing
power, available bandwidth, the number of entries in its neighbor
table, uptime, identity, network address information, and next hop
peer information.

A PathTrack request specifies which diagnostic information is
requested using a DiagnosticsRequest data structure, defined and

discussed in detail later in this document in Section 5.1. Base
information is requested by setting the appropriate flags in the data
structure in the request. If all flags are clear (no bits are set),
then the PathTrack request is only used for requesting the next hop
information. In this case the iterative mode of PathTrack is
degraded to a RouteQuery method which is only used for checking the
liveness of the peers along the routing path. The PathTrack request
can be routed directly or through the overlay based on the routing
mode chosen by the initiator node.

A response to a successful PathTrackReq is a PathTrackAns message.
The PathTrackAns contains general diagnostic information in the
payload, returned using a DiagnosticResponse data structure. This
data structure is defined and discussed in detail later in this
document in Section 5.2. The information returned is determined
based on the information requested in the flags in the corresponding
request.

4.3.1.1. PathTrack Request

The structure of the PathTrack request is as follows:

```
                    struct{
                        Destination destination;
                        DiagnosticsRequest request;
                    }PathTrackReq;
```

The fields of the PathTrackReq are as follows:

   destination : The destination which the initiator node is
   interested in. This may be any valid destination object,
   including a NodeID, opaque ids, or ResourceID. One example should
   be noted that, for debugging purpose, the initiator will use the
   destination ID as it was used when failure happened.

   request : A DiagnosticsRequest, as discussed in Section 5.1.

4.3.1.2. PathTrack Response

The structure of the PathTrack response is as follows:

```
                    struct{
                        Destination next_hop;
                        DiagnosticsResponse response;
                     }PathTrackAns;
```

The fields of the PathTrackAns are as follows:

next_hop : The information of the next hop node from the
responding intermediate peer to the destination. If the
responding peer is the responsible peer for the destination ID,
then the next_hop node ID equals the responding node ID, and after
receiving a PathTrackAns where the next_hop node ID equals the
responding node ID the initiator MUST stop the iterative process.

response : A DiagnosticsResponse, as discussed in Section 5.2.

4.4. Error Code Extensions

This document extends the Error response method defined in the RELOAD
specification to support error cases resulting from diagnostic
queries. When an error is encountered in RELOAD, the Message Code
0xFFFF is returned. The ErrorResponse structure includes an error
code. We define new error codes to report possible error conditions
detected while performing diagnostics:

```
   Code Value           Error Code Name
      [TBD1]               Underlay Destination Unreachable
      [TBD2]               Underlay Time exceeded
      [TBD3]               Message Expired
      [TBD4]               Upstream Misrouting
      [TBD5]               Loop detected
      [TBD6]               TTL hops exceeded
```

The final error codes will be assigned by IANA as specified in RELOAD
protocol [RFC6940]. The error code is returned by the upstreaming
node before the failure node. And the upstreaming node uses the
normal ping to detect the failure type and return it to the initiator
node, which will help the user (initiator node) to understand where
the failure happened and what kind of error happened, as the failure
may happen at the same location and for the same reason when sending
the normal message and the diagnostics message.

As defined in RELOAD, additional information may be stored (in an
implementation-specific way) in the optional error_info byte string.
While the specifics are obviously left to the implementation, as an
example, in the case of [TBD1], the error_field could be used to
provide additional information as to why the underlay destination is
unreachable (net unreachable, host unreachable, fragmentation needed,
etc.)

5. Diagnostic Data Structures

Both the extended Ping method and PathTrack method use the following
common diagnostics data structures to collect data. Two common

structures are defined: DiagnosticsRequest for requesting data, and
DiagnosticsResponse for returning the information.

5.1. DiagnosticsRequest Data Structure

The DiagnosticsRequest data structure is used to request diagnostic
information and has the following form:

```
enum{ (2^16-1) } DiagnosticKindId;

struct{
    DiagnosticKindId kind;
    opaque  diagnostic_extension_contents<0..2^32-1>;
}DiagnosticExtension;

struct{
    uint64 expiration;
    uint64 timestamp_initiated;
    uint64 dMFlags;
    uint32 ext_length;
    DiagnosticExtension diagnostic_extensions_list<0..2^32-1>;
 }DiagnosticsRequest;
```

The fields in the DiagnosticsRequest are as follows:

   expiration : The time when the request will expire represented as
   the number of milliseconds elapsed since midnight Jan 1, 1970 UTC
   not counting leap seconds. This will have the same values for
   seconds as standard UNIX time or POSIX time. More information can
   be found at UnixTime [UnixTime]. This value MUST have a value of
   between 1 and 600 seconds in the future. This value is used to
   prevent replay attacks.

   timestamp_initiated : The time when the P2PSIP diagnostics request
   was initiated represented as the number of milliseconds elapsed
   since midnight Jan 1, 1970 UTC not counting leap seconds. This
   will have the same values for seconds as standard UNIX time or
   POSIX time.

   dMFlags : A mandatory field which is an unsigned 64-bit integer
   indicating which base diagnostic information the request initiator
   node is interested in. The initiator sets different bits to
   retrieve different kinds of diagnostic information. If dMFlags is
   set to zero, then no base diagnostic information is conveyed in
   the PathTrack response. If dMFlag is set to all '1's, then all
   base diagnostic information values are requested. A request may
   set any number of the flags to request the corresponding
   diagnostic information.

Note this memo specifies the initial set of flags, the flags can
be extended. The dMflags indicate general diagnostic information
The mapping between the bits in the dMFlags and the diagnostic
information kind presented is as described in Section 9.1.

ext_length : the length of the extended diagnostic request
information in bytes. If the value is greater than or equal to 1,
then some extended diagnostic information is being requested, on
the assumption this information will be included in the response
if the recipient understands the extended request and is willing
to provide it. The specific diagnostic information requested is
defined in the diagnostic_extensions_list below. A value of zero
indicates no extended diagnostic information is being requested.
The value of ext_length MUST NOT be negative. Note that it is not
the length of the entire DiagnosticsRequest data structure, but of
the data making up the diagnostic_extensions_list.

diagnostic_extensions_list : consists of one or more
DiagnosticExtension structures (see below) documenting additional
diagnostic information being requested. Each DiagnosticExtension
consists of the following fields:

   kind : a numerical code indicating the type of extension
   diagnostic information (see Section 9.2). Note that kinds
   0xF000 - 0xFFFE are reserved for overlay specific diagnostics
   and may be used without IANA registration for local diagnostic
   information. Kinds from 0x0000 to 0x003F MUST NOT be indicated
   in the diagnostic_extensions_list in the message request, as
   they may be represented using the dMFlags in a much simpler
   (and more space efficient) way.

   diagnostic_extension_contents : the opaque data containing the
   request for this particular extension. This data is extension
   dependent.

5.2. DiagnosticsResponse Data Structure

```
            enum { (2^16-1) } DiagnosticKindId;
            struct{
                DiagnosticKindId kind;
                opaque diagnostic_info_contents<0..2^16-1>;
            }DiagnosticInfo;

            struct{
                uint64 expiration;
                uint64 timestamp_initiated;
                uint64 timestamp_received;
                uint8 hop_counter;
                uint32 ext_length;
                DiagnosticInfo diagnostic_info_list<0..2^32-1>;
            }DiagnosticsResponse;
```

The fields in the DiagnosticsResponse are as follows:

   expiration : The time when the response will expire represented as
   the number of milliseconds elapsed since midnight Jan 1, 1970 UTC
   not counting leap seconds. This will have the same values for
   seconds as standard UNIX time or POSIX time. This value MUST have
   a value of between 1 and 600 seconds in the future.

   timestamp_initiated: This value is copied from the diagnostics
   request message. The benefit of containing such a value in the
   response message is that the initiator node does not have to
   maintain the state.

   timestamp_received : The time when P2PSIP Overlay diagnostic
   request was received represented as the number of milliseconds
   elapsed since midnight Jan 1, 1970 UTC not counting leap seconds.
   This will have the same values for seconds as standard UNIX time
   or POSIX time.

   hop_counter : This field only appears in diagnostic responses. It
   MUST be exactly copied from the TTL field of the forwarding header
   in the received request. This information is sent back to the
   request initiator, allowing it to compute the number of hops that
   the message traversed in the overlay.

   ext_length : the length of the returned DiagnosticInfo information
   in bytes. If the value is greater than or equal to 1, then some
   extended diagnostic information (as specified in the
   DiagnosticsRequest) was available and is being returned. In that
   case, this value indicates the length of the returned information.
   A value of zero indicates no extended diagnostic information is
   included, either because none was requested or the request could
   not be accommodated. The value of ext_length MUST NOT be

negative. Note that it is not the length of the entire
DiagnosticsRequest data structure, but of the data making up the
diagnostic_info_list.

diagnostic_info_list : consists of one or more DiagnosticInfo
structures containing the requested diagnostic_info_contents. The
fields in the DiagnosticInfo structure are as follows:

kind : A numeric code indicating the type of information being
returned. For base data requested using the dMFlags, this code
corresponds to the dMFlag set, and is described in Section 5.1.
For diagnostic extensions, this code will be identical to the
value of the DiagnosticKindId set in the "kind" field of the
DiagnosticExtension of the request. See Section 9.2.

diagnostic_info_contents : Data containing the value for the
diagnostic information being reported. Various kinds of
diagnostic information can be retrieved, Please refer to
Section 5.3 for details of the diagnostic kind ID for the base
diagnostic information that may be reported.

5.3. dMFlags and Diagnostic Kind ID Types

The dMFlags field described above is a 64 bit field that allows
initiator nodes to identify up to 62 items of base information to
request in a request message (the first and last flags being
reserved). When the requested base information is returned in the
response, the value of the diagnostic kind ID will correspond to the
numeric field marked in the dMFlags in the request. The values for
the dMFlags are defined in Section 9.1 and the diagnostic kind IDs
are defined in Section 9.2. The information contained for each value
is described in this section. Access to each kind of diagnostic
information MUST NOT be allowed unless compliant to the rules defined
in Section 7.

STATUS_INFO (8 bits):A single value element containing an unsigned
byte representing whether or not the node is in congestion status.
An example usage of STATUS_INFO is for congestion-aware routing.
In this scenario, each peer has to update its congestion status
periodically. An intermediate peer in the distributed hash table
(DHT) network will choose its next hop according to both the DHT
routing algorithm and the status information. This is done to
avoid increasing load on congested peers. The rightmost 4 bits
are used and other bits MUST be cleared to "0"s for future use.
There are 16 levels of congestion status, with "0x00" represent
zero load and "0x0F" represent congested. This document does not
provide a specific method for congestion, leaving this decision to
each overlay implementation. One possible option for an overlay

implementation would be to take node's CPU/memory/bandwidth usage
percentage in the past 600 seconds and normalize the highest value
to the range [0x00, 0x0F]. And an overlay implementation can also
decide to not use all that 16 values from 0x00 to 0x0F. A future
draft may define an objective measure or specific algorithm for
this.

ROUTING_TABLE_SIZE (32 bits): A single value element containing an
unsigned 32-bit integer representing the number of peers in the
peer's routing table. The administrator of the overlay may be
interested in statistics of this value for reasons such as routing
efficiency.

PROCESS_POWER (64 bits): A single value element containing an
unsigned 64-bit integer specifying the processing power of the
node in unit of MIPS. Fractional values are rounded up.

UPSTREAM_BANDWIDTH (64 bits): A single value element containing an
unsigned 64-bit integer specifying the upstream network bandwidth
(provisioned or maximum, not available) of the node in unit of
Kbps. Fractional values are rounded up. For multihomed hosts,
this should be the link used to send the response.

DOWNSTREAM_BANDWIDTH (64 bits): A single value element containing
an unsigned 64-bit integer specifying the downstream network
bandwidth (provisioned or maximum, not available) of the node in
unit of Kbps. Fractional values are rounded up. For multihomed
hosts, this should be the link the request was received from.

SOFTWARE_VERSION: A single value element containing a US-ASCII
string that identifies the manufacture, model, operating system
information and the version of the software. Given that there are
very large number of peers in some networks, and no peer is likely
to know all other peer's software, this information may be very
useful to help determine if the cause of certain groups of
misbehaving peers is related to specific software versions. While
the format is peer-defined, a suggested format is as follows:
"ApplicationProductToken (Platform; OS-or-CPU) VendorProductToken
(VendorComment)". For example: "MyReloadApp/1.0 (Unix; Linux
x86_64) libreload-java/0.7.0 (Stonyfish Inc.)". The string is a
C-style string, and MUST be delimited by "\0"."\0" MUST NOT be
included in the string itself to prevent confusion with the
delimiter.

MACHINE_UPTIME (64 bits): A single value element containing an
unsigned 64-bit integer specifying the time the node's underlying
system has been up in seconds.

APP_UPTIME (64 bits): A single value element containing an
unsigned 64-bit integer specifying the time the P2P application
has been up in seconds.

MEMORY_FOOTPRINT (64 bits): A single value element containing an
unsigned 64-bit integer representing the memory footprint of the
peer program in kilobytes (1024 bytes). Fractional values are
rounded up.

DATASIZE_STORED (64 bits): An unsigned 64-bit integer representing
the number of bytes of data being stored by this node.

INSTANCES_STORED: An array element containing the number of
instances of each kind stored. The array is indexed by Kind-ID.
Each entry is an unsigned 64-bit integer.

MESSAGES_SENT_RCVD: An array element containing the number of
messages sent and received. The array is indexed by method code.
Each entry in the array is a pair of unsigned 64-bit integers
(packed end to end) representing sent and received.

EWMA_BYTES_SENT (32 bits): A single value element containing an
unsigned 32-bit integer representing an exponential weighted
average of bytes sent per second by this peer. sent = alpha x
sent_present + (1 - alpha) x sent where sent_present represents
the bytes sent per second since the last calculation and sent
represents the last calculation of bytes sent per second. A
suitable value for alpha is 0.8. This value is calculated every
five seconds.

EWMA_BYTES_RCVD (32 bits): A single value element containing an
unsigned 32-bit integer representing an exponential weighted
average of bytes received per second by this peer. rcvd = alpha x
rcvd_present + (1 - alpha) x rcvd where rcvd_present represents
the bytes received per second since the last calculation and rcvd
represents the last calculation of bytes received per second. A
suitable value for alpha is 0.8. This value is calculated every
five seconds.

UNDERLAY_HOP (8 bits): Indicates the IP layer hops from the
intermediate peer which receives the diagnostics message to the
next hop peer for this message. (Note: RELOAD does not require
the intermediate peers to look into the message body. So here we
use PathTrack to gather underlay hops for diagnostics purpose).

BATTERY_STATUS (8 bits): The left-most bit is used to indicate
whether this peer is using a battery or not. If this bit is clear

(set to '0'), then the peer is using a battery for power. The
other 7 bits are to be determined by specific applications.

6. Message Processing

6.1. Message Creation and Transmission

When constructing either a Ping message with diagnostic extensions or
a PathTrack message, the sender first creates and populates a
DiagnosticsRequest data structure. The timestamp_initiated field is
set to the current time, and the expiration field is constructed
based on this time. The sender includes the dMFlags field in the
structure, setting any number (including all) of the flags to request
particular diagnostic information. The sender MAY leave all the bits
unset, requesting no particular diagnostic information.

The sender MAY also include diagnostic extensions in the
DiagnosticsRequest data structure to request additional information.
If the sender includes any extensions, it MUST calculate the length
of these extensions and set the ext_length field to this value. If
no extensions are included, the sender MUST set ext_length to zero.

The format of the DiagnosticRequest data structure and its fields
MUST follow the restrictions defined in Section 5.1.

When constructing a Ping message with diagnostic extensions, the
sender MUST create an MessageExtension structure as defined in RELOAD
[RFC6940], setting the value of type to 0x0002, and the value of
critical to FALSE. The value of extension_contents MUST be a
DiagnosticsRequest structure as defined above. The message MAY be
directed to a particular NodeId or ResourceID, but MUST NOT be sent
to the broadcast NodeID.

When constructing a PathTrack message, the sender MUST set the
message_code for the RELOAD MessageContents structure to
path_track_req ([TBD7]). The request field of the PathTrackReq MUST
be set to the DiagnosticsRequest data structure defined above. The
destination field MUST be set to the desired destination, which MAY
be either a NodeId or ResourceID but SHOULD NOT be the broadcast
NodeID.

6.2. Message Processing: Intermediate Peers

When a request arrives at a peer, if the peer's responsible ID space
does not cover the destination ID of the request, then the peer MUST
continue processing this request according to the overlay specified
routing mode from RELOAD protocol.

In P2PSIP overlay, error responses to a message can be generated by
either an intermediate peer or the responsible peer. When a request
is received at a peer, the peer may find connectivity failures or
malfunctioning peers through the pre-defined rules of the overlay
network, e.g. by analyzing via list or underlay error messages. In
this case, the intermediate peer SHOULD return an error response to
the initiator node, reporting any malfunction node information
available in the error message payload. All error responses
generated MUST contain the appropriate error code.

Each intermediate peer receiving a Ping message with extensions (and
which understands the extension) or receiving a PathTrack request/
response SHOULD check the expiration value (Unix time format) to
determine if the message is expired. If the message expired, the
intermediate peer SHOULD generate a response with Error Code [TBD3]
"Message Expired", return the response to the initiator node, and
discard the message.

The intermediate peer SHOULD return an error response with the Error
Code [TBD1] "Underlay Destination Unreachable" when it receives an
ICMP message with "Destination Unreachable" information after
forwarding the received request to the destination peer.

The intermediate peer SHOULD return an error response with the Error
Code [TBD2] "Underlay Time Exceeded" when it receives an ICMP message
with "Time Exceeded" information after forwarding the received
request.

The peer SHOULD return an Error response with Error Code [TBD4]
"Upstream Misrouting" when it finds its upstream peer disobeys the
routing rules defined in the overlay. The immediate upstream peer
information SHOULD also be conveyed to the initiator node.

The peer SHOULD return an Error response with Error Code [TBD5] "Loop
detected" when it finds a loop through the analysis of via list.

The peer SHOULD return an Error response with Error Code [TBD6] "TTL
hops exceeded" when it finds that the TTL field value is no more than
0 when forwarding.

6.3. Message Response Creation

When a diagnostic request message arrives at a peer, it is
responsible for the destination ID specified in the forwarding
header, and assuming it understands the extension (in the case of
Ping) or the new request type PathTrack, it MUST follow the
specifications defined in RELOAD to form the response header, and
perform the following operations:

When constructing a PathTrack response, the sender MUST set the
message_code for the RELOAD MessageContents structure to
path_track_ans ([TBD8]).

The receiver MUST check the expiration value (Unix time format) in
the DiagnosticsRequest to determine if the message is expired. If
the message is expired, the peer MUST generate a response with the
Error Code [TBD3] "Message Expired", return the response to the
initiator node, and discard the message.

If the message is not expired, the receiver MUST construct a
DiagnosticsResponse structure, as follows: The TTL value from the
forwarding header is copied to the hop_counter field of the
DiagnosticsResponse structure. Note that the default value for TTL
at the beginning represents 100-hops unless overlay configuration has
overridden the value. The receiver generates an Unix time format
timestamp for the current time of day and places it in the
timestamp_received field, and constructs a new expiration time and
places it in the expiration field of the DiagnosticsResponse.

The destination peer MUST check if the initiator node has the
authority to request specific types of diagnostic information, and if
appropriate, append the diagnostic information requested in the
dMFlags and diagnostic_extensions (if any) using the
diagnostic_info_list field to the DiagnosticsResponse structure. If
any information returned, the receiver MUST calculate the length of
the response and set ext_length appropriately. If no diagnostic
information is returned, ext_length MUST be set to zero.

The format of the DiagnosticResponse data structure and its fields
MUST follow the restrictions defined in Section 5.2.

In the event of an error, an error response containing the error code
followed by the description (if they exist) MUST be created and sent
to the sender. If the initiator node asks for diagnostic information
that they are not authorized to query, the receiving peer MUST return
an Error response with the Error Code 2 "Error_Forbidden".

6.4. Interpreting Results

The initiator node, as well as the responding peer, MAY compute the
overlay One-Way-Delay time through the value in timestamp_received
and the timestamp_initiated field. However, for a single hop
measurement, the traditional measurement methods (IP layer ping) MUST
be used instead of the overlay layer diagnostics methods.

The P2P overlay network using the diagnostics methods specified in
this document MUST enforce time synchronization with a central time

server. Network Time Protocol [RFC5905] can usually maintain time to
within tens of milliseconds over the public Internet, and can achieve
better than one millisecond accuracy in local area networks under
ideal conditions. However, this document does not specify the choice
for time synchronization, leaving it to the implementation.

The initiator node receiving the Ping response MAY check the
hop_counter field and compute the overlay hops to the destination
peer for the statistics of connectivity quality from the perspective
of overlay hops.

7. Authorization through Overlay Configuration

Different level of access control can be made for different users/
nodes. For example, diagnostic information A can be accessed by node
1 and 2, but diagnostic information B can only be accessed by node 2.

The overlay configuration file MUST contain the following XML
elements for authorizing a node to access the relative diagnostic
kinds.

diagnostic-kind: This has the attribute "kind" with the hexadecimal
number indicating the diagnostic Kind Type, this attribute has the
same value with Section 9.2, and at least one sub element "access-
node".

access-node: This element contains one hexadecimal number indicating
a NodeID, and the node with this NodeID is allowed to access the
diagnostic "kind" under the same diagnostic-kind element.

8. Security Considerations

The authorization for diagnostic information must be designed with
care to prevent it becoming a method to retrieve information for bot
attacks. It should also be noted that attackers can use diagnostics
to analyze overlay information to attack certain key peers. For
example, diagnostic information might be used to fingerprint a peer
where the peer will loose its anonymity characteristics, but
anonymity might be very important for some P2P overlay networks, and
defenses against such fingerprinting are probably very hard. As
such, networks where anonymity is of very high importance may find
implementation of diagnostics problematic or even undesirable,
despite the many advantages it offers. As this document is a RELOAD
extension, it follows RELOAD message header and routing
specifications, the common security considerations described in the
base document [RFC6940] are also applicable to this document.
Overlays may define their own requirements on who can collect/share
diagnostic information.

9. IANA Considerations

9.1. Diagnostics Flag

   IANA SHALL create a "RELOAD Diagnostics Flag" Registry. Entries in
   this registry are 1-bit flags contained in a 64-bits long integer
   dMFlags denoting diagnostic information to be retrieved as described
   in Section 4.3.1. New entries SHALL be defined via [RFC5226]
   Standards Action. The initial contents of this registry are:

```
   +------------------------+-----------------------------+----------+
   |  diagnostic information |diagnostic flag in dMFlags   | RFC      |
   |------------------------+-----------------------------+----------|
   |Reserved                | 0x 0000 0000 0000 0000       |RFC-[TBDX]|
   |STATUS_INFO             | 0x 0000 0000 0000 0001       |RFC-[TBDX]|
   |ROUTING_TABLE_SIZE      | 0x 0000 0000 0000 0002       |RFC-[TBDX]|
   |PROCESS_POWER           | 0x 0000 0000 0000 0004       |RFC-[TBDX]|
   |UPSTREAM_BANDWIDTH      | 0x 0000 0000 0000 0008       |RFC-[TBDX]|
   |DOWNSTREAM_ BANDWIDTH   | 0x 0000 0000 0000 0010       |RFC-[TBDX]|
   |SOFTWARE_VERSION        | 0x 0000 0000 0000 0020       |RFC-[TBDX]|
   |MACHINE_UPTIME          | 0x 0000 0000 0000 0040       |RFC-[TBDX]|
   |APP_UPTIME              | 0x 0000 0000 0000 0080       |RFC-[TBDX]|
   |MEMORY_FOOTPRINT        | 0x 0000 0000 0000 0100       |RFC-[TBDX]|
   |DATASIZE_STORED         | 0x 0000 0000 0000 0200       |RFC-[TBDX]|
   |INSTANCES_STORED        | 0x 0000 0000 0000 0400       |RFC-[TBDX]|
   |MESSAGES_SENT_RCVD      | 0x 0000 0000 0000 0800       |RFC-[TBDX]|
   |EWMA_BYTES_SENT         | 0x 0000 0000 0000 1000       |RFC-[TBDX]|
   |EWMA_BYTES_RCVD         | 0x 0000 0000 0000 2000       |RFC-[TBDX]|
   |UNDERLAY_HOP            | 0x 0000 0000 0000 4000       |RFC-[TBDX]|
   |BATTERY_STATUS          | 0x 0000 0000 0000 8000       |RFC-[TBDX]|
   |Reserved                | 0x FFFF FFFF FFFF FFFF       |RFC-[TBDX]|
   +------------------------+-----------------------------+--------+
```

   [To RFC editor: Please replace all RFC-[TBDX] in this document with
   the RFC number of this document.]

9.2. Diagnostic Kind ID Types

   IANA SHALL create a "RELOAD Diagnostic Kind ID Types" Registry.
   Entries in this registry are 16-bit integers denoting diagnostics
   extension data kind types carried in the diagnostic request and
   response message, as described in Section 5.2. Code points from
   0x0000 to 0x003F SHALL be assigned together with flags within "RELOAD
   Diagnostics Flag" registry via RFC 5226 [RFC5226] standards action.
   Code points in the range 0x0040 to 0xEFFF SHALL be registered via RFC
   5226 standards action.

```
+---------------------------+--------------+---------------+
|    Diagnostic Kind Type   |     Code     | Specification |
+---------------------------+--------------+---------------+
|          reserved         |    0x0000    |   RFC-[TBDX]  |
|        STATUS_INFO        |    0x0001    |   RFC-[TBDX]  |
|     ROUTING_TABLE_SIZE    |    0x0002    |   RFC-[TBDX]  |
|       PROCESS_POWER       |    0x0003    |   RFC-[TBDX]  |
|     UPSTREAM_BANDWIDTH    |    0x0004    |   RFC-[TBDX]  |
|    DOWNSTREAM_BANDWIDTH   |    0x0005    |   RFC-[TBDX]  |
|      SOFTWARE_VERSION     |    0x0006    |   RFC-[TBDX]  |
|       MACHINE_UPTIME      |    0x0007    |   RFC-[TBDX]  |
|         APP_UPTIME        |    0x0008    |   RFC-[TBDX]  |
|      MEMORY_FOOTPRINT     |    0x0009    |   RFC-[TBDX]  |
|      DATASIZE_STORED      |    0x000A    |   RFC-[TBDX]  |
|      INSTANCES_STORED     |    0x000B    |   RFC-[TBDX]  |
|     MESSAGES_SENT_RCVD    |    0x000C    |   RFC-[TBDX]  |
|      EWMA_BYTES_SENT      |    0x000D    |   RFC-[TBDX]  |
|      EWMA_BYTES_RCVD      |    0x000E    |   RFC-[TBDX]  |
|        UNDERLAY_HOP       |    0x000F    |   RFC-[TBDX]  |
|       BATTERY_STATUS      |    0x0010    |   RFC-[TBDX]  |
| reserved for future flags |   0x0011-40  |   RFC-[TBDX]  |
|    local use (reserved)   | 0xF000-0xFFFE |  RFC-[TBDX]  |
|          reserved         |    0xFFFF    |   RFC-[TBDX]  |
+---------------------------+--------------+---------------+
```

Table 1: Diagnostic Kind Types

9.3. Message Codes

This document introduces two new types of messages and their
responses, requiring the following additions to the "RELOAD Message
Code" Registry defined in RELOAD [RFC6940]. These additions are:

```
+-------------------+------------+----------+
| Message Code Name | Code Value |   RFC    |
+-------------------+------------+----------+
|   path_track_req  |   [TBD7]   | RFC-AAAA |
|   path_track_ans  |   [TBD8]   | RFC-AAAA |
+-------------------+------------+----------+
```

Table 2: Extensions to RELOAD Message Codes

[To RFC editor: Values starting at [TBD1] were used to prevent
collisions with RELOAD base values and other extensions. Please
replace with the next highest available values. The final message
codes will be assigned by IANA. And all RFC-AAAA should be replaced
with the RFC number of RELOAD when publication.]

9.4. Error Code

   This document introduces the following new error codes, extending the
   "RELOAD Message Code" registry as described below:

```
+----------------------------------------+------------+----------+
|          Message Code Name             | Code Value |   RFC    |
+----------------------------------------+------------+----------+
| Error_Underlay_Destination_Unreachable |   [TBD1]   | RFC-AAAA |
|      Error_Underlay_Time_Exceeded      |   [TBD2]   | RFC-AAAA |
|         Error_Message_Expired          |   [TBD3]   | RFC-AAAA |
|       Error_Upstream_Misrouting        |   [TBD4]   | RFC-AAAA |
|          Error_Loop_Detected           |   [TBD5]   | RFC-AAAA |
|        Error_TTL_Hops_Exceeded         |   [TBD6]   | RFC-AAAA |
+----------------------------------------+------------+----------+
```

                 Table 3: Extensions to RELOAD Error Codes

   [To RFC editor: Values starting at [TBD1] were used to prevent
   collisions with RELOAD base values and other extensions. Please
   replace with the next highest available values. The final message
   codes will be assigned by IANA. And all RFC-AAAA should be replaced
   with the RFC number of RELOAD when publication.]

9.5. Message Extension

   This document introduces the following new RELOAD extension code:

```
+-----------------+------------+----------+
| Extension Name  | Code Value |   RFC    |
+-----------------+------------+----------+
| Diagnostic_Ping |   0x0002   | RFC-AAAA |
+-----------------+------------+----------+
```

                   Table 4: New RELOAD Extension Code

   [To RFC editor: The value 0x0002 was used to prevent collisions with
   other extensions. Please replace with the next highest available
   value. The final codes will be assigned by IANA. And all RFC-AAAA
   should be replaced with the RFC number of RELOAD when publication.]

9.6. XML Name Space Registration

   This document registers a URI for the config-diagnostics XML
   namespaces in the IETF XML registry defined in [RFC3688]. All the
   elements defined in this document belong to this namespace.

   URI: urn:ietf:params:xml:ns:p2p:config-diagnostics
   Registrant Contact: The IESG.
   XML: N/A, the requested URIs are XML namespaces

   And the overlay configuration file MUST contain the following xml
   language declaring P2PSIP diagnostics as a mandatory extension to
   RELOAD.

   <mandatory-extension>
                urn:ietf:params:xml:ns:p2p:config-diagnostics
   </mandatory-extension>

10. Acknowledgments

   We would like to thank Zheng Hewen for the contribution of the
   initial version of this document. We would also like to thank Bruce
   Lowekamp, Salman Baset, Henning Schulzrinne, Jiang Haifeng and Marc
   Petit-Huguenin for the email discussion and their valued comments,
   and special thanks to Henry Sinnreich for contributing to the usage
   scenarios text. We would like to thank the authors of the RELOAD
   protocol for transferring text about diagnostics to this document.

11. References

11.1. Normative References

   [RFC0792] Postel, J., "Internet Control Message Protocol", STD 5,
              RFC 792, DOI 10.17487/RFC0792, September 1981,
              <http://www.rfc-editor.org/info/rfc792>.

   [RFC2119] Bradner, S., "Key words for use in RFCs to Indicate
              Requirement Levels", BCP 14, RFC 2119,
              DOI 10.17487/RFC2119, March 1997,
              <http://www.rfc-editor.org/info/rfc2119>.

   [RFC3688] Mealling, M., "The IETF XML Registry", BCP 81, RFC 3688,
              DOI 10.17487/RFC3688, January 2004,
              <http://www.rfc-editor.org/info/rfc3688>.

   [RFC5905] Mills, D., Martin, J., Ed., Burbank, J., and W. Kasch,
              "Network Time Protocol Version 4: Protocol and Algorithms
              Specification", RFC 5905, DOI 10.17487/RFC5905, June 2010,
              <http://www.rfc-editor.org/info/rfc5905>.

   [RFC6940] Jennings, C., Lowekamp, B., Ed., Rescorla, E., Baset, S.,
              and H. Schulzrinne, "REsource LOcation And Discovery
              (RELOAD) Base Protocol", RFC 6940, DOI 10.17487/RFC6940,
              January 2014, <http://www.rfc-editor.org/info/rfc6940>.

11.2. Informative References

   [UnixTime]
               "UnixTime", <Wikipedia, "Unix Time",
               <http:/wikipedia.org/wiki/Unix_time>.>.

   [I-D.ietf-p2psip-concepts]
               Bryan, D., Matthews, P., Shim, E., Willis, D., and S.
               Dawkins, "Concepts and Terminology for Peer to Peer SIP",
               draft-ietf-p2psip-concepts-07 (work in progress), May
               2015.

   [Overlay-Failure-Detection]
               Zhuang, S., "On failure detection algorithms in overlay
               networks", Proc. IEEE Infocomm, Mar 2005.

   [Handling_Churn_in_a_DHT]
               Rhea, S., "Handling Churn in a DHT", USENIX Annual
               Conference, June 2004.

   [Diagnostic_Framework]
               Jin, X., "A Diagnostic Framework for Peer-to-Peer
               Streaming", 2005.

   [Diagnostics_and_NAT_traversal_in_P2PP]
               Gupta, G., "Diagnostics and NAT Traversal in P2PP - Design
               and Implementation", Columbia University Report , June
               2008.

   [RFC5226]   Narten, T. and H. Alvestrand, "Guidelines for Writing an
               IANA Considerations Section in RFCs", BCP 26, RFC 5226,
               DOI 10.17487/RFC5226, May 2008,
               <http://www.rfc-editor.org/info/rfc5226>.

Appendix A. Examples

   Below, we sketch how these metrics can be used.

A.1. Example 1

   A peer may set EWMA_BYTES_SENT and EWMA_BYTES_RCVD flags in the
   PathTrackReq to its direct neighbors. A peer can use EWMA_BYTES_SENT
   and EWMA_BYTES_RCVD of another peer to infer whether it is acting as
   a media relay. It may then choose not to forward any requests for
   media relay to this peer. Similarly, among the various candidates
   for filling up routing table, a peer may prefer a peer with a large
   UPTIME value, small RTT, and small LAST_CONTACT value.

A.2. Example 2

   A peer may set the STATUS_INFO Flag in the PathTrackReq to a remote
   destination peer. The overlay has its own threshold definition for
   congestion. The peer can obtain knowledge of all the status
   information of the intermediate peers along the path. Then it can
   choose other paths to that node for the subsequent requests.

A.3. Example 3

   A peer may use Ping to evaluate the average overlay hops to other
   peers by sending PingReq to a set of random resource or node IDs in
   the overlay. A peer may adjust its timeout value according to the
   change of average overlay hops.

Appendix B. Problems with Generating Multiple Responses on Path

   An earlier version of this document considered an approach where a
   response was generated by each intermediate peer as the message
   traversed the overlay. This approach was discarded. One reason this
   approach was discarded was that it could provide a DoS mechanism,
   whereby an attacker could send an arbitrary message claiming to be
   from a spoofed "sender" the real sender wished to attack. As a
   result of sending this one message, many messages would be generated
   and sent back to the spoofed "sender" - one from each intermediate
   peer on the message path. While authentication mechanisms could
   reduce some risk of this attack, it still resulted in a fundamental
   break from the request-response nature of the RELOAD protocol, as
   multiple responses are generated to a single request. Although one
   request with responses from all the peers in the route will be more
   efficient, it was determined to be too great a security risk and
   deviation from the RELOAD architecture.

Appendix C. Changes to the Draft

   To RFC editor: This section is to track the changes. Please remove
   this section before publication.

C.1. Changes since -00 version

   1. Changed title from "Diagnose P2PSIP Overlay Network" to "P2PSIP
       Overlay Diagnostics".

   2. Changed the table of contents. Add a section about message
       processing and a section of examples.

   3. Merge diagnostics text from the p2psip base draft -01.

   4. Removed ECHO method for security reasons.

C.2. **Changes since -01 version**

      Added BATTERY_STATUS as diagnostic information.

      Removed UnderlayTTL test from the Ping method, instead adding an
      UNDERLAY_HOP diagnostic information for PathTrack method.

      Give some examples for diagnostic information, and give some
      editor's notes for further work.

C.3. **Changes since -02 version**

   Provided further explanation as to why the base draft Ping in the
   current form cannot be used to replace Ping, and why some combination
   of methods cannot replace PathTrack.

C.4. **Changes since -03 version**

   Modified structure used to share information collected. Both
   mechanisms now use a common data structure to convey information.

C.5. **Changes since -04 version**

   Updated the authors' addresses and modified the last sentence in .
   (Section 4.3.1.2)

C.6. **Changes since -05 version**

   Resolve Marc's comments from the mailing list. And define the
   details of STATUS_INO.

C.7. **Changes in version -10**

   Resolve the authorization issue and other comments (e.g. define
   diagnostics as a mandatory extension) from WGLC. And check for the
   languages.

C.8. **Changes in version -15**

   Changed several diagnostic kind return values to be 64 bit vs. 32 bit
   to provide headroom. Split bandwidth into upstream and downstream.
   Renamed length in diagnostic request object to ext_length, added
   ext_length to response object, and clarified that ext_length is
   length of diagnostic info/extensions being returned, not the length
   of the object.

   Aligned many flags/values with RELOAD by using hex vs decimal values.

   Significant reorganization and edit for readability.

Authors' Addresses

   Haibin Song
   Huawei

   Email: haibin.song@huawei.com

   Jiang Xingfeng
   Huawei

   Email: jiangxingfeng@huawei.com

   Roni Even
   Huawei
   14 David Hamelech
   Tel Aviv 64953
   Israel

   Email: roni.even@mail101.huawei.com

   David A. Bryan
   ethernot.org
   Cedar Park, Texas
   United States of America

   Email: dbryan@ethernot.org

   Yi Sun
   ICT

   Email: sunyi@ict.ac.cn