P2PSIP Working Group
Internet-Draft
Intended status: Standards Track
Expires: July 1, 2012

H. Song
X. Jiang
R. Even
Huawei
D. Bryan
Polycom

December 29, 2011

# P2PSIP Overlay Diagnostics
# draft-ietf-p2psip-diagnostics-08

Abstract

This document describes mechanisms for P2PSIP diagnostics. It defines extensions to the RELOAD P2PSIP base protocol RELOAD [I-D.ietf-p2psip-base] 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 a P2PSIP overlay networks.

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 relies on accurate and timely detection of failures.

There are a number of situations in which some peers in a P2PSIP overlay may malfunction or behave badly. For example, these peers 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 P2PSIP system. Node failures may be also caused by underlying failures, for example the recovery from an incorrect overlay topology may be slow when the IP layer routing failover speed 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 peer's neighbors [Overlay-Failure-Detection], but these algorithms by themselves can only detect the disabled neighbors using the periodic method, it may not be enough for service providers operating the overlay network.

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 and is intended as a complement to keep-alive algorithms in the P2PSIP overlay itself.

## 2. Terminology

The concepts used in this document are compatible with "Concepts and Terminology for Peer to Peer SIP" [I-D.ietf-p2psip-concepts] and the P2PSIP base protocol RELOAD [I-D.ietf-p2psip-base].

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 require no network management in the traditional sense to set up and to configure individual P2P nodes. 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]. However, the unresponsive nodes may only be temporarily disabled, for example due to some local cryptographic processing overload, disk processing overload or link overload. It is therefore useful to repeat the connectivity checks to see if such 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 for a particular node to follow up an individual user complaint or failure. For example, in this case a technical support person may use a desktop sharing application with the permission of the user to determine remotely the health 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 statistics of nodes from the overlay. The simple connectivity tests are not dependent on the type of P2PSIP overlay. Note that other tests may be required as well, such as checking the health and performance of the user's computer or mobile device and also

checking the bandwidth of the link connecting the user to the Internet.

c. P2P system diagnostics to check the overall health of the P2P overlay network, the consumption of network bandwidth, for the presence of problem links and also to check 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] as well as in earlier P2PSIP documents [Diagnostics_and_NAT_traversal_in_P2PP]. While this is a difficult problem, a great deal of information can be obtained in helping these diagnostics by sending messages to diagnose the network. This document provides a framework for obtaining this information.

## 4. Overview of operations

The diagnostic mechanisms described in this document are mainly intended to detect and localize failures or monitor performance in P2PSIP overlay networks. It provides mechanisms to detect and localize malfunctioning or badly behaving peers including disabled peers, congested peers and misrouting peers. It provides a mechanism to detect direct connectivity or connectivity to a specified peer, 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 peer, as well as to diagnose the path to that peer. The second is a new method and response, Path_Track, 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 draft reuses P2PSIP base protocol specification and extends them to introduce the new diagnostics methods. The extensions strictly follow the P2PSIP base protocol specification on the messages routing, transport, NAT traversal etc. The diagnostic methods are however P2PSIP protocol independent.

This document primarily describes how to detect and localize failures including disabled peers, congested peers, 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 (RFC792 ICMP echo request [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 Path_Track method) for diagnosing P2PSIP overlay networks.

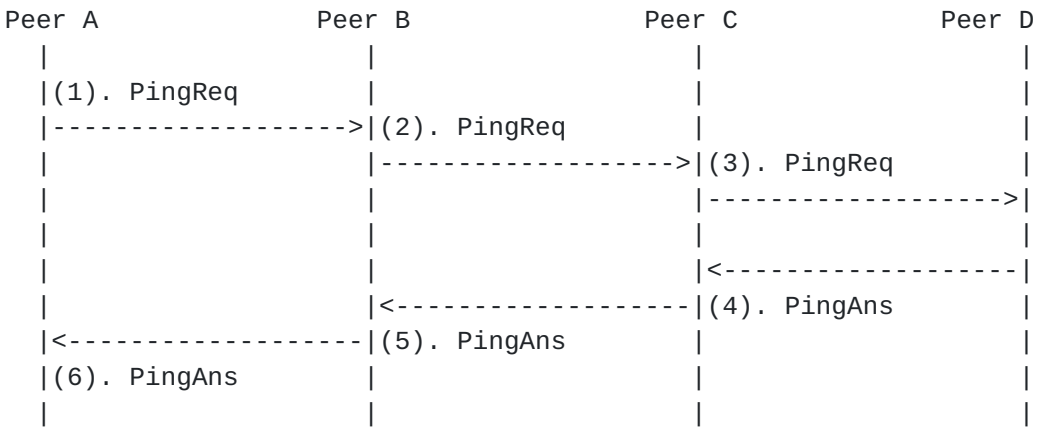
One approach these tools can be used is to detect the connectivity to the specified peer or the availability of the specified resource-record through the extended P2PSIP Ping operation once the overlay network receives some alarms about overlay service degradation or interruption, if the Ping fails, one can then send a Path_Track to determine where the fault lies.

The diagnostic information must be only provided to authorized peers. Some diagnostic information can be authorized to all the participants in the P2PSIP overlay, and some other diagnostic information can only be provided to the authorization peer list of each diagnostic information according to the local or overlay policy. The authorization depends on the kinds of the diagnostic information and the administrative considerations, and is application specific.

## 4.1. "Ping-like" Behavior: Extending Ping

To provide "ping-like" behavior, the RELOAD Ping method has been 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, and 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:

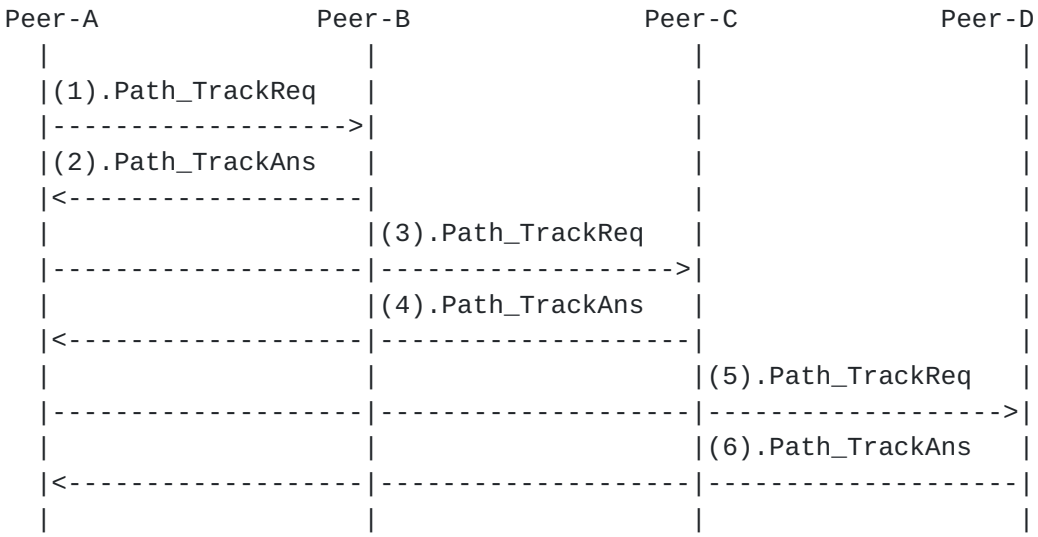



Ping Diagnostic Message Flow

### 4.2. "Traceroute-like" Behavior: The Path_Track Method

We define a simple Path_Track method for retrieving diagnostics information iteratively. First, the initiating peer asks its neighbor A which is the next hop peer to the destination ID, and then retrieve the next hop peer B information, along with optional diagnostic information of A, to the initiator peer. Then the initiator peer asks the next hop peer B (directly or symmetric routing) to get the further next hop peer C information and diagnostic information of B. 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 Path_Track message (with diagnostic extensions) is as follows:




Path_Track Diagnostic Message Flow

There have been proposals made on list that RouteQuery and a series of Fetch requests can be used to replace the Path_Track mechanism, but in the presence of churn such an operation would not, strictly speaking, provide identical results, as the path may change between RouteQuery and Fetch operations. (although obviously the path could change between steps of Path_Track as well).

### 4.3. 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 as a result of working group discussion. 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 of the 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.

## 5. RELOAD diagnostic extensions

This document extends RELOAD to carry diagnostics information. Considering the special usage of diagnostics, this document defines extensions for a payload to Ping, as well as the new method Path_Track and its response. Additionally, new Error codes, message bodies for conveying diagnostics, and some suggested common diagnostic values are defined. Processing of the Path_Track message and the diagnostic bodies is discussed.

The mechanism defined in this document follows the RELOAD specification, the new request and response message use the message format specified in P2PSIP base protocol messages. Please refer to the RELOAD [I-D.ietf-p2psip-base] for details of the protocol.

### 5.1. Diagnostic Data Structures

The diagnostics use the following common diagnostics data structures. Two common structures are defined, DiagnosticsRequest for requesting data, and DiagnosticsResponse for returning information.

#### 5.1.1. DiagnosticRequest Data Structure

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

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

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

struct{
    uint64 expiration;
    uint64 timestamp_initiated;
    uint32 length;
    DiagnosticExtension diagnostic_extensions[length];
 }DiagnosticsRequest;
```

The fields in the DiagnosticsRequest are as follows:

expiration : The time when the request will be expired 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]

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.

length : the length of the extended diagnostic request information in bytes. If the value is greater than or equal to 1, then some extended diagnostics information is requested. The value of length must not be negative.

dMFlags : A mandatory field which is an unsigned 64-bit integer indicating which base diagnostic information the initiator is interested in. The initiator sets different bits to retrieve different kinds of diagnostic information. If dMFlags is clear, then no base diagnostic information is conveyed in the Path_Track 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 by standard action. 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 9.6.

diagnostic_extensions : consists of one or more DiagnosticExtension structures (see below) documenting additional diagnostic information being requested.

Each DiagnosticExtension has the following fields:

type : the extension type (see Section 9.1) Note that type 0xFFFE is reserved for overlay specific diagnostics and may be used without IANA registration for local diagnostic information.

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

### 5.1.2. DiagnosticResponse Data Structure

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

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

The fields in the DiagnosticsResponse are as follows:

expiration : The time when the response will be expired 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.

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 hops that the message traversed in the overlay.

diagnostic_info_list : consists of one or more DiagnosticInfo values containing the requested diagnostic information.

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 listed in Section 5.1.1 Paragraph 5. For diagnostic extensions, this code will be identical to the value of the DiagnosticExtensionRequestType set in the type field of the DiagnosticExtension of the request, and these two values will be assigned together. See Section 9.2.

diagnostic_information : Data containing the value for the diagnostic information being reported. Various kinds of diagnostic information can be retrieved, Please refer to Section 5.1.3 for details of the types and Diagnostic Kind-ID for the base diagnostic information that may be reported.

### 5.1.3. dMFlags and Diagnostic Kind ID Types

The dMFlags field described above is a 64 bit field that allows requesters to identify up to 62 items of base information to request (the first and last flags being reserved) when sending a request. 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 9.6 and the diagnostic Kind-IDs are defined in Section 9.2. The information contained for each value is described in this section.

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 DHT network will choose its next hop according to both the DHT routing algorithm and the status information, and then forward requests to the chosen next hop, so as 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" respresent zero load and "0x0F" representing congested.

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 the consideration such

as routing efficiency.

PROCESS_POWER (32 bits): A single value element containing an unsigned 32-bit integer specifying the processing power of the node in unit of MIPS.

BANDWIDTH (32 bits): A single value element containing an unsigned 32-bit integer specifying the bandwidth of the node in unit of Kbps.

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. The format is like: ApplicationProductToken (Platform; OS-or-CPU) * VendorProductToken (VendorComment). One example is: MyReloadApp/1.0 (Unix; Linux x86_64) libreload-java/0.7.0 (Stonyfish Inc.).

MACHINE_UPTIME (64 bits): A single value element containing an unsigned 64-bit integer specifying the time the nodes 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 (32 bits): A single value element containing an unsigned 32- bit integer representing the memory footprint of the peer program in kibibytes(1024 bytes).

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 index 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. Same calculation as above.

UNDERLAY_HOP (8 bits): It indicates the IP layer hops from the intermediate peer which receives the diagnostics message to its next hop peer for this message. (Note: this is from the underlayTTL in the previous version. However, RELOAD does not require the intermediate peers to look into the message body. So here we use Path_Track to gather underlay hops for diagnostics purpose.

BATTERY_STATUS (8 bits): The left-most bit is used to indicate whether this peer is using battery or not. If this bit is clear ('0'), then the peer is using battery power. The other 7 bits are to be determined by specific applications.

### 5.1.4. Extending Diagnostic Information

The DiagnosticsExtension structure may be used to extend the diagnostic information collected.

Editor's Note: The self-tuning draft [I-D.ietf-p2psip-self-tuning] could extend the diagnostics information here to collect related information for calculating self-tuning parameters.

## 5.2. Request Extension: Ping

To extend the ping request for use in diagnostics, a new extension as defined in Section 5.3.3 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 5 and specified in the RELOAD draft section 13.14. The extension contents consists of a DiagnosticsRequest structure as defined in Section 5.1.1. This extension MAY be used for new requests of the the Ping method and MUST NOT be included in requests using any other method.

This extension is NOT critical. If a peer does not extend the extension, they will simply ignore the diagnostic portion of the message, and will treat the message 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.

### 5.3. New Request: Path_Track

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

A Path_Track request specifies which diagnostic information is requested using a DiagnosticsRequest Data structure. Base information is requested by setting the appropriate flags in the dMFlags field of the DiagnosticsRequest. If the flag is clear (no bits are set), then the Path_Track request is only used for requesting the next hop information. In this case the iterative mode of Path_Track is degraded to a Route_Query method which is only used for checking the liveness of the peers along the routing path. The Path_Track 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. There is a general diagnostic information portion of the payload, the contents of which are based on the flags in the request. Please refer to Section 5.1.3 for the definitions of the base diagnostic information, and Section 9.3 for the numeric message code for the new request.

#### 5.3.1. Path_track Request

The structure of the Path_track request is as follows:

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

The fields of the PathTrackReq are as follows:

destination : The destination which the requester is interested in. This may be any valid destination object, including a Node-ID, compressed ids, or Resource-ID.

request : A DiagnosticsRequest, as discussed in Section 5.1.

### 5.3.2. Path_track Response

The structure of the Path_Track 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 node. 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 that the initiator must stop the iterative process.

response : A DiagnosticsResponse, as discussed in Section 5.1.

## 5.4. Error Codes

This document extends the Error response method defined in the P2PSIP base protocol specification to describe the result of diagnostics. When an error is encountered in RELOAD, the Message Code 0xFFFF is returned. The ErrorResponse structure includes an error code, and we define new error codes to report on possible error conditions detected while performing diagnostics:

| Code Value | Error Code Name |
|---|---|
| 101 | Underlay Destination Unreachable |
| 102 | Underlay Time exceeded |
| 103 | Message Expired |
| 104 | Upstream Misrouting |
| 105 | Loop detected |
| 106 | TTL hops exceeded |

The final error codes will be assigned by IANA. as specified in section 13.9 of the p2psip base protocol [I-D.ietf-p2psip-base].

This document introduces several types of error information in the error_info field in the case of Code 101. These are represented as an opaque UTF-8 text string. Here are some examples for the error info.

error_info:

net unreachable
host unreachable
protocol unreachable
port unreachable
fragmentation needed
source route failed

The error_info field of the Code 102 to 106 are to be specified by the specific overlay.

## 5.5. Message Processing

### 5.5.1. Message Creation and Transmission

When constructing either a Ping message with diagnostic extensions or a Path_Track message, the sender MUST create a DiagnosticsRequest data structure. The sender MUST set the expiration field of this structure in Unix time timestamp format. The value MUST be at least 10 seconds in the future, and MUST NOT be more than 600 seconds in the future. The timestamp_initiated field MUST be set to the current time in Unix time timestamp format. The sender MUST include the dMFlags field in the structure, and MAY send any number (or all) of the flags to request the desired diagnostic information. The sender MAY leave all the bits unset, requesting no diagnostic information, but MUST include the field. The sender MAY also include diagnostic extensions for additional information. If the sender includes any extensions, they MUST calculate the length of these extensions and set the length field to the correct length. If no extensions are included, the sender MUST set length to zero.

When constructing a diagnostic ping message, the sender MUST create an MessageExtension structure as defined in RELOAD 5.3.3. The value of type MUST be 0x0002. The value of critical must be FALSE. The value of extension_contents MUST be the DiagnosticsRequest structure defined above. The sender MUST place the MessageExtension structure in the extensions field of the MessageContents structure. The message MAY be directed to a particular NodeId or ResourceID, but SHOULD NOT be sent to the broadcast NodeID.

When constructing a Path_Track message, the sender MUST set the message_code for the RELOAD MessageContents structure for Path_track. 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.

### 5.5.2. 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 process this request according to the overlay specified routing mode from the base draft.

In p2psip overlay, the error response can be generated by the intermediate peer or responsible peer, to a diagnostic message or other messages. When a request is received at a peer, the peer may find some connectivity failures or malfunction peers through the pre-defined rules of the overlay network, e.g. by analyzing via list or underlay error messages. The peer SHOULD report the error responses to the initiator node. The malfunction node information should also be reported to the initiator node in the error message payload. All error responses contain the Error code followed by the subcode and descriptions if existed.

Each intermediate peer receiving a Ping message with extensions (and which understands the extension) or receiving a Path_Track request/ response SHOULD check the expiration value (Unix time format) to determine if the message expired. If the message expired, the intermediate peer SHOULD generate a message with Error Code 103 "Message Expired" and return it to the initiator node, and discard the message.

The peer should return an Error response with the Error Code 101 "Underlay Destination Unreachable" when it receives an ICMP message with "Destination Unreachable" information after forwarding the received request to the destination peer.

The peer should return an Error response with the Error Code 102 "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 104 "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 105 "Loop detected" when it finds a loop through the analysis of via list.

The peer should return an Error response with Error Code 106 "TTL hops exceeded" when it finds that the TTL field value is no more than 0 when forwarding.

### 5.5.3. Message Response Creation

When a diagnostic request message arrives at a peer, it understands the extension (in the case of ping) or the new request type path_track, and it is responsible for the destination ID specified in the forwarding header, it MUST follow the specifications defined in 5.1.3 of the base draft to form the response header, and perform the following operations:

The receiver MUST check the expiration value (Unix time format) in the DiagnosticsRequest to determine if the message expired. If the message expired, the peer MUST generate a message with the Error Code 103 "Message Expired" and return it to the initiator node, and discard the message.

If the message is not expired, the receiver MUST construct a DiagnosticsResponse structure. The destination peer MUST copy the TTL value from the forwarding header to the hop_counter field of the DiagnosticsResponse structure. Note that this value will represent 100-hops unless overlay configuration has overridden the value. The receiver MUST generate an Unix time format timestamp for the current time of day and place it in the timestamp_received field. The receiver MUST construct a new expiration time and place it in the expiration field of the DiagnosticsResponse. This expiration MUST be at least 10 seconds in the future and MUST NOT be more than 600 seconds in the future.

The destination peer MUST check if the initiator node has the authority to get certain kinds of diagnostic information, and if appropriate, appends the diagnostic information requested in the dMFlags and diagnostic_extensions (if any) in the diagnostic_info_list field of the DiagnosticsResponse structure. If there is any information returned, the receiver MUST calculate the length of the response and set length appropriately. If there is no diagnostic information returned, length MUST be set to zero.

In the event of an error, an error response containing the error code followed by the subcode and description (if they exist) MUST be created and sent to the sender. If the requester asks for diagnostic information that they are not authorized to query, the receiving peer MUST return an Error response with the Error Code 1 "Error_Unauthorized".

### 5.5.4. 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 MUST be used instead of the overlay layer diagnostics methods.

Editor note: We need more discussion and careful consideration on how to use the timestamp here because time synchronization is a barrier in open Internet environment, while in the operator's network, it may be less of a problem.

The initiator node receiving the Inspect 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.

## 6. Examples

Below, we sketch how these metrics can be used.

### 6.1. Example 1

A peer may set EWMA_BYTES_SENT and WEMA_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.

### 6.2. Example 2

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

### 6.3. Example 3

A peer may use Inspect to evaluate the average overlay hops to other peers by sending InspectReq 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.

## 7. Mandatory Extension

This document defines the following XML namespace name for the purpose of adding it to a <mandatory-extension> element in the

overlay configuration file, to force all nodes of an overlay to support diagnostics.

xmlns:diagnostics="urn:ietf:params:xml:ns:p2p:diagnostics"

## 8. Security Considerations

The authorization for diagnostics information must be designed with care to prevent it becoming a resort to retrieve information for bot attacks. It should also be careful that attackers can use diagnostics to analyze overlay information to attack certain key peers if there are. As this draft is a RELOAD extension, it follows RELOAD message header and routing specifications, the common security considerations described in the base draft [I-D.ietf-p2psip-base] are also applicable to this draft. Overlays may define their own requirements on who can collect/share diagnostic information.

## 9. IANA Considerations

### 9.1. Diagnostic Extension Types

IANA SHALL create a "RELOAD Diagnostic Extension Types" Registry. Entries in this registry are 16-bit integers denoting diagnostics extension data types in the diagnostic request message, as described in Section 5.1.1. Code points in the range 0x0040 to 0xFFFD SHALL be registered via [RFC5226] Standards Action. The code SHALL be assigned together with "RELOAD Diagnostic Kind ID Types" with the same value.

| Diagnostic Extension Name | Code | Specification |
|---|---|---|
| reserved (identifiers used for built in types) | 0 - 0x003F | RFC-BBBB |
| local use (reserved) | 0xFFFE | RFC-BBBB |
| reserved | 0xFFFF | RFC-BBBB |

Table 1: Diagnostic Extension Request Types

### 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 response message, as described in Section 5.1.2. Code points from 0x0000 to 0x003F SHALL be assigned together with flags within "RELOAD Diagnostics

Flag" registry via RFC 5226 Standards Action. Code points in the range 0x0040 to 0xFFFD SHALL be registered via RFC 5226 Standards Action and be assigned together with same value within "RELOAD Diagnostic Extension Types" registry.

| Diagnostic Kind Type | Code | Specification |
|---|---|---|
| reserved | 0x0000 | RFC-BBBB |
| STATUS_INFO | 0x0001 | RFC-BBBB |
| ROUTING_TABLE_SIZE | 0x0002 | RFC-BBBB |
| PROCESS_POWER | 0x0003 | RFC-BBBB |
| BANDWIDTH | 0x0004 | RFC-BBBB |
| SOFTWARE_VERSION | 0x0005 | RFC-BBBB |
| MACHINE_UPTIME | 0x0006 | RFC-BBBB |
| APP_UPTIME | 0x0007 | RFC-BBBB |
| MEMORY_FOOTPRINT | 0x0008 | RFC-BBBB |
| DATASIZE_STORED | 0x0009 | RFC-BBBB |
| INSTANCES_STORED | 0x000A | RFC-BBBB |
| MESSAGES_SENT_RCVD | 0x000B | RFC-BBBB |
| EWMA_BYTES_SENT | 0x000C | RFC-BBBB |
| EWMA_BYTES_RCVD | 0x000D | RFC-BBBB |
| UNDERLAY_HOP | 0x000E | RFC-BBBB |
| BATTERY_STATUS | 0x000F | RFC-BBBB |
| reserved | 0x003F | RFC-BBBB |
| local use (reserved) | 0xFFFE | RFC-BBBB |
| reserved | 0xFFFF | RFC-BBBB |

Table 2: 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 [I-D.ietf-p2psip-base]. These additions are:

| Message Code Name | Code Value | RFC |
|---|---|---|
| path_track_req | 101 | RFC-AAAA |
| path_track_ans | 102 | RFC-AAAA |

Table 3: Extensions to RELOAD Message Codes

Note: Values starting at 101 were used to prevent collisions with

RELOAD base values. Once RELOAD moves to RFC, these values may start at the next higher value after the RELOAD base values. The final message code will be assigned by IANA.

## 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 | 101 | RFC-AAAA |
| Error_Underlay_Time_Exceeded | 102 | RFC-AAAA |
| Error_Message_Expired | 103 | RFC-AAAA |
| Error_Upstream_Misrouting | 104 | RFC-AAAA |
| Error_Loop_Detected | 105 | RFC-AAAA |
| Error_TTL_Hops_Exceeded | 106 | RFC-AAAA |

Table 4: Extensions to RELOAD Error Codes

## 9.5. Message Extension

This document introduces the following new RELOAD extension code:

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

Table 5: New RELOAD Extension Code

## 9.6. 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 5.3. 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-BBBB |
| STATUS_INFO | 0x 0000 0000 0000 0001 | RFC-BBBB |
| ROUTING_TABLE_SIZE | 0x 0000 0000 0000 0002 | RFC-BBBB |
| PROCESS_POWER | 0x 0000 0000 0000 0004 | RFC-BBBB |
| BANDWIDTH | 0x 0000 0000 0000 0008 | RFC-BBBB |
| SOFTWARE_VERSION | 0x 0000 0000 0000 0010 | RFC-BBBB |
| MACHINE_UPTIME | 0x 0000 0000 0000 0020 | RFC-BBBB |
| APP_UPTIME | 0x 0000 0000 0000 0040 | RFC-BBBB |
| MEMORY_FOOTPRINT | 0x 0000 0000 0000 0080 | RFC-BBBB |
| DATASIZE_STORED | 0x 0000 0000 0000 0100 | RFC-BBBB |
| INSTANCES_STORED | 0x 0000 0000 0000 0200 | RFC-BBBB |
| MESSAGES_SENT_RCVD | 0x 0000 0000 0000 0400 | RFC-BBBB |
| EWMA_BYTES_SENT | 0x 0000 0000 0000 0800 | RFC-BBBB |
| EWMA_BYTES_RCVD | 0x 0000 0000 0000 1000 | RFC-BBBB |
| UNDERLAY_HOP | 0x 0000 0000 0000 2000 | RFC-BBBB |
| BATTERY_STATUS | 0x 0000 0000 0000 4000 | RFC-BBBB |
| Reserved | 0x FFFF FFFF FFFF FFFF | RFC-BBBB |

## 10. Open Questions

## 11. Acknowledgments

We would like to thank Zheng Hewen for the contribution of the initial version of this draft. 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 p2psip base draft for transferring text about diagnostics to this document.

## 12. Appendix: Changes to the Draft

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

### 12.1. 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.

### 12.2. Changes since -01 version

Added BATTERY_STATUS as diagnostic information.

Removed UnderlayTTL test from the Inspect 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.

### 12.3. Changes since -02 version

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

### 12.4. Changes since -03 version

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

### 12.5. Changes since -04 version

Updated the authors' addresses and modified the last sentence in . (Section 5.3.2)

### 12.6. Changes since -05 version

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

## 13. References

### 13.1. Normative References

[RFC0792] Postel, J., "Internet Control Message Protocol", STD 5, RFC 792, September 1981.

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

[RFC3261] Rosenberg, J., Schulzrinne, H., Camarillo, G., Johnston, A., Peterson, J., Sparks, R., Handley, M., and E. Schooler, "SIP: Session Initiation Protocol", RFC 3261, June 2002.

[RFC5226] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", BCP 26, RFC 5226, May 2008.

[I-D.ietf-p2psip-sip] Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "A SIP Usage for RELOAD", draft-ietf-p2psip-sip-06 (work in progress), July 2011.

[I-D.ietf-p2psip-base] Jennings, C., Lowekamp, B., Rescorla, E., Baset, S., and H. Schulzrinne, "REsource LOcation And Discovery (RELOAD) Base Protocol", draft-ietf-p2psip-base-19 (work in progress), October 2011.

[I-D.zheng-p2psip-diagnose] Yongchao, S. and X. Jiang, "Diagnose P2PSIP Overlay Network", draft-zheng-p2psip-diagnose-04 (work in progress), December 2008.

[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.

## 13.2. Informative References

[RFC4981] Risson, J. and T. Moors, "Survey of Research towards Robust Peer-to-Peer Networks: Search Methods", RFC 4981, September 2007.

[I-D.ietf-behave-rfc3489bis]
Rosenberg, J., Mahy, R., Matthews, P., and D. Wing, "Session Traversal Utilities for (NAT) (STUN)", draft-ietf-behave-rfc3489bis-18 (work in progress), July 2008.

[I-D.matuszewski-p2psip-security-requirements]
Yongchao, S., Matuszewski, M., and D. York, "P2PSIP Security Overview and Risk Analysis", draft-matuszewski-p2psip-security-requirements-06 (work in progress), September 2009.

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

[I-D.song-p2psip-security-eval]
Yongchao, S., Zhao, B., Jiang, X., and J. Haifeng, "P2PSIP Security Analysis and Evaluation", draft-song-p2psip-security-eval-00 (work in progress), February 2008.

[I-D.baset-p2psip-p2pp]
Baset, S., Schulzrinne, H., and M. Matuszewski, "Peer-to-Peer Protocol (P2PP)", draft-baset-p2psip-p2pp-01 (work in progress), November 2007.

[I-D.ietf-mmusic-ice]
Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", draft-ietf-mmusic-ice-19 (work in progress), October 2007.

[I-D.bryan-p2psip-app-scenarios]
Bryan, D., Shim, E., Lowekamp, B., and S. Dawkins, "Application Scenarios for Peer-to-Peer Session Initiation Protocol (P2PSIP)", draft-bryan-p2psip-app-scenarios-00 (work in progress), November 2007.

[I-D.bryan-p2psip-requirements]
Bryan, D., "P2PSIP Protocol Framework and Requirements", draft-bryan-p2psip-requirements-00 (work in progress), July 2007.

[I-D.ietf-p2psip-self-tuning]
Maenpaa, J., Camarillo, G., and J. Hautakorpi, "A Self-tuning Distributed Hash Table (DHT) for REsource LOcation And Discovery (RELOAD)", draft-ietf-p2psip-self-tuning-04

(work in progress), July 2011.

[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-04 (work in progress), October 2011.

## Authors' Addresses

Song Haibin
Huawei
No. 101, Software Avenue
Nanjing, Jiangsu Province 210012
China

Phone: +86-25-56624792
Email: haibin.song@huawei.com

Jiang Xingfeng
Huawei
No. 101, Software Avenue
Nanjing, Jiangsu Province 210012
China

Email: jiang.x.f@huawei.com

Roni Even
Huawei
14 David Hamelech
Tel Aviv 64953
Israel

Email: even.roni@huawei.com

David A. Bryan
Polycom
Williamsburg, Virginia
United States of America

Email: dbryan@ethernot.org