PPSP
INTERNET-DRAFT
Intended Status: Standards Track
Expires: June 15, 2015

Rui S. Cruz
Mario S. Nunes
IST/INESC-ID/INOV
Yingjie Gu
Jinwei Xia
Rachel Huang
Huawei
Joao P. Taveira
IST/INOV
Deng Lingli
China Mobile

December 12, 2014

PPSP Tracker Protocol-Base Protocol (PPSP-TP/1.0)
draft-ietf-ppsp-base-tracker-protocol-07

Abstract

This document specifies the base Peer-to-Peer Streaming Protocol-Tracker Protocol (PPSP-TP/1.0), an application-layer control (signaling) protocol for the exchange of meta information between trackers and peers. The specification outlines the architecture of the protocol and its functionality, and describes message flows, message processing instructions, message formats, formal syntax and semantics. The PPSP Tracker Protocol enables cooperating peers to form content streaming overlay networks to support near real-time Structured Media content delivery (audio, video, associated timed text and metadata), such as adaptive multi-rate, layered (scalable) and multi-view (3D) videos, in live, time-shifted and on-demand modes.

Copyright and License Notice

Table of Contents

1 Introduction

The Peer-to-Peer Streaming Protocol (PPSP) is composed of two protocols: the PPSP Tracker Protocol and the PPSP Peer Protocol. RFC 6972 [RFC6972] specifies that the Tracker Protocol should standardize the messages between PPSP peers and PPSP trackers and also defines the requirements.

The PPSP Tracker Protocol provides communication between trackers and peers, by which peers send meta information to trackers, report streaming status and obtain peer lists from trackers.

The PPSP architecture requires PPSP peers able to communicate with a tracker in order to participate in a particular streaming content swarm. This centralized tracker service is used by PPSP peers for content registration and location.

The signaling and the media data transfer between PPSP peers is not in the scope of this specification.

This document describes the base PPSP Tracker protocol and how it satisfies the requirements for the IETF Peer-to-Peer Streaming Protocol, in order to derive the implications for the standardization of the PPSP streaming protocols and to identify open issues and promote further discussion.

1.1 Terminology

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119 [KEYWORDS].

ABSOLUTE TIME: Absolute time is expressed as ISO 8601 timestamps, using zero UTC offset. Fractions of a second may be indicated. Example for December 25, 2010 at 14h56 and 20.25 seconds: basic format 20101225T145620.25Z or extended format 2010-12-25T14:56:20.25Z.

CHUNK: A Chunk is a basic unit of data organized in P2P streaming for storage, scheduling, advertisement and exchange among peers.

CHUNK ID: A unique resource identifier for a Chunk. The identifier type depends on the addressing scheme used, i.e., an integer, an HTTP-URL and possibly a byte-range, and is described in the MPD.

CONNECTION TRACKER: The node running the tracker service to which the PPSP peer will connect when it wants to get registered and join

the PPSP system.

LEECH: A Peer that has not yet completed the transfer of all Chunks of the media content.

LIVE STREAMING: It refers to a scenario where all the audiences receive streaming content for the same ongoing event. It is desired that the lags between the play points of the audiences and streaming source be small.

MEDIA PRESENTATION DESCRIPTION (MPD): Formalized description for a media presentation, i.e., describes the structure of the media, namely, the Representations, the codecs used, the Chunks, and the corresponding addressing scheme.

METHOD: The method is the primary function that a request from a peer is meant to invoke on a tracker. The method is carried in the request message itself.

ONLINE TIME: Online Time shows how long the peer has been in the P2P streaming system since it joined. This value indicates the stability of a peer, and can be calculated by the tracker whenever necessary.

PEER: A Peer refers to a participant in a P2P streaming system that not only receives streaming content, but also caches and streams streaming content to other participants.

PEER ID: The identifier of a Peer such that other Peers, or the Tracker, can refer to the Peer by using its ID. The Peer ID is mandatory, can take the form of a universal unique identifier (UUID), defined in [RFC4122], and can be bound to a network address of the Peer, i.e., an IP address, or a uniform resource identifier/locator (URI/URL) that uniquely identifies the corresponding Peer in the network. The Peer ID and any required security certificates are obtained from an offline enrollment server.

PEER LIST: A list of Peers which are in a same SWARM maintained by the Tracker. A Peer can fetch the Peer List of a SWARM from the Tracker or from other Peers in order to know which Peers have the required streaming content.

PPSP: The abbreviation of Peer-to-Peer Streaming Protocols. PPSP refer to the primary signaling protocols among various P2P streaming system components, including the Tracker and the Peer.

PPSP-TP: The abbreviation of Peer-to-Peer Streaming Protocols - Tracker Protocol.

REPRESENTATION: Structured collection of one or more media components.

REQUEST: A message sent from a Peer to a Tracker, for the purpose of invoking a particular operation.

RESPONSE: A message sent from a Tracker to a Peer, for indicating the status of a request sent from the Peer to the Tracker.

SEEDER: A Peer that holds and shares the complete media content.

SERVICE PORTAL: A logical entity typically used for client enrollment and content information publishing, searching and retrieval. It is usually located in a server of content provider.

SWARM: A Swarm refers to a group of Peers who exchange data to distribute Chunks of the same content (e.g., video/audio program, digital file, etc.) at a given time.

SWARM ID: The identifier of a Swarm containing a group of Peers sharing a common streaming content. The Swarm-ID may use a universal unique identifier (UUID), e.g., a 64 or 128 bit datum to refer to the content resource being shared among peers.

SUPER-NODE: A Super-Node is a special kind of Peer deployed by ISPs. This kind of Peer is more stable with higher computing, storage and bandwidth capabilities than normal Peers.

TRACKER: A Tracker refers to a directory service that maintains a list of Peers participating in a specific audio/video channel or in the distribution of a streaming file. Also, the Tracker answers Peer List queries received from Peers. The Tracker is a logical component which can be centralized or distributed.

TRANSACTION ID: The identifier of a REQUEST from the Peer to the Tracker. Used to disambiguate RESPONSES that may arrive in a different order of the corresponding REQUESTs.

VIDEO-ON-DEMAND (VoD): It refers to a scenario where different audiences may watch different parts of the same recorded streaming with downloaded content.

1.2 Design Overview

The functional entities related to PPSP protocols are the Client Media Player, the service Portal, the Tracker and the Peers. The complete description of Client Media Player and service Portal is not

discussed here, as not in the scope the specification. The functional entities directly involved in the PPSP Tracker Protocol are trackers and peers (which may support different capabilities).

The Client Media Player is a logical entity providing direct interface to the end user at the client device, and includes the functions to select, request, decode and render contents. The Client Media Player may interface with the local peer application using request and response standard formats for HTTP Request and Response messages [RFC2616].

The service Portal is a logical entity typically used for client enrollment and content information publishing, searching and retrieval.

A Peer corresponds to a logical entity (typically in a user device) that actually participates in sharing a media content. Peers are organized in (various) swarms corresponding each swarm to the group of peers streaming a certain content at any given time.

The Tracker is a logical entity that maintains the lists of peers storing Chunks for a specific Live media channel or on-demand media streaming content, answers queries from peers and collects information on the activity of peers. While a Tracker may have an underlying implementation consisting of more than one physical node, logically the Tracker can most simply be thought of as a single element, and in this document it will be treated as a single logical entity.

The Tracker Protocol is not used to exchange actual content data (either on-demand or Live streaming) with peers, but information about which peers can provide the content.

1.2.1 Typical Use Cases

When a peer wants to receive streaming of a selected content (Leech mode):

1. Peer connects to a Connection Tracker and joins a Swarm.
2. Peer acquires a list of other peers in the Swarm from the Connection Tracker.
3. [Peer Protocol] Peer exchanges its content availability with the peers on the obtained peer list.
4. [Peer Protocol] Peer identifies the peers with desired content.
5. [Peer Protocol] Peer requests content from the identified peers.

When a peer wants to share streaming contents (Seeder mode) with other peers:

1. Peer connects to the Connection Tracker.
2. Peer sends information to the Connection Tracker about the swarms it belongs to (joined swarms).

After having been disconnected due to some termination condition, a Peer can resume previous activity by connecting and re-joining the corresponding Swarm(s).

1.2.2 Enrollment and Bootstrap

In order to be able to bootstrap in the P2P network, a peer must first obtain a Peer ID (identifier of the peer) and any required security certificates or authorization tokens from an enrollment service (end-user registration). The specification of the format of the Peer ID is not in the scope of this document.

```
+--------+      +--------+     +--------+    +---------+  +--------+
| Player |      | Peer_1 |     | Portal |    | Tracker |  | Peer_2 |
+--------+      +--------+     +--------+    +---------+  +--------+
    |                |               |              |            |
(a) |--Page request----------------->|              |            |
    |<--------------Page with links--|              |            |
    |--Select stream (MPD Request)-->|              |            |
    |<--------------------OK+MPD(x)--|              |            |
(b) |--Start/Resume->|--CONNECT(join x)------------>|            |
    |<-----------OK--|<----------------OK+Peerlist--|            |
    |                |                              |            |
    |--Get(Chunk)--->|<---------- (Peer protocol) ------------->|
    |<--------Chunk--|<---------------------------------Chunks--|
    :                :               :              :            :
    |                |--STAT_REPORT---------------->|            |
    |                |<-------------------------OK--|            |
    :                :               :              :            :
    |                |--FIND----------------------->|            |
    |                |<----------------OK+Peerlist--|            |
    :                :               :              :            :
    |--Get(Chunk)--->|<---------- (Peer protocol) ------------->|
    |<--------Chunk--|<---------------------------------Chunks--|
    :                :               :              :            :
```

Figure 1: A typical PPSP session for streaming a content.

To join an existing P2P streaming service and to participate in content sharing, any Peer must first locate a Tracker.

As illustrated in Figure 1, a P2P streaming session may be initiated starting at point (a), with the Client Media Player browsing for the desired content in order to request it (to the local Peer_1 in the figure), or resume a previously initiated stream, but starting at point (b). For this example, the Peer_1 is in mode LEECH.

At point (a) in Figure 1, the Client Media Player accesses the Portal and selects the content of interest. The Portal returns the Media Presentation Description (MPD) file that includes information about the address of one or more Trackers (that can be grouped by tiers of priority) which are controlling the Swarm x for that media content (e.g., content x).

With the information from the MPD the Client Media Player is able to trigger the start of the streaming session, requesting to the local Peer_1 the Chunks of interest.

The PPSP streaming session is then started (or resumed) at Peer_1 by sending a PPSP-TP CONNECT message to the Tracker in order to join Swarm x. The Tracker will then return the OK response message containing a peer list, if the CONNECT message is successfully accepted. From that point onwards every Chunk request is addressed by Peer_1 to its neighbors (Peer_2 in Figure 1) using the PPSP Peer Protocol, returning the received Chunks to the Client Media Player.

Once CONNECTed, Peer_1 needs to periodically report its status and statistics data to the Tracker using a PPSP-TP STAT_REPORT message.

If Peer_1 needs to refresh its neighborhood (for example, due to churn) it will send a PPSP-TP FIND message (with the desired scope) to the Tracker.

Peers that are only SEEDERs (i.e., serving contents to other peers), as are the typical cases of service provider P2P edge caches and/or Media Servers, trigger their P2P streaming sessions for contents x, y, z... (Figure 2), not from Media Player signals, but from some "Start" activation signal received from the service provider provisioning mechanism. In this particular case the Peer starts or resumes all its streaming sessions just by sending a PPSP-TP CONNECT message to the Tracker, in order to "join" all the requested swarms (Figure 2).

Periodically, the Peer also report its status and statistics data to the Tracker using a PPSP-TP STAT_REPORT message.

```
                +---------+                     +---------+
                |  Seeder |                     | Tracker |
                +---------+                     +---------+
                     |                               |
              Start->|--CONNECT (join x,y,z)-------->|
                     |<--------------------------OK--|
                     :                               :
                     |                               |
                     |--STAT_REPORT----------------->|
                     |<--------------------------Ok--|
                     :                               :
                     |                               |
                     |--STAT_REPORT----------------->|
                     |<--------------------------Ok--|
                     :                               :
```

Figure 2: A typical PPSP session for a streaming Seeder.

The specification of the mechanisms used by the Client Media Player (or provisioning process) and the Peer to signal start/resume streams or request media chunks, obtain a Peer ID, security certificates or tokens are not in the scope of this document.

2 Protocol Architecture and Functional View

The PPSP Tracker Protocol architecture is intended to be compatible with the web infrastructure. PPSP-TP is designed with a layered approach i.e., a PPSP-TP Request/Response layer, a Message layer and a Transport layer. The PPSP-TP Request/Response layer deals with the interactions between Tracker and Peers using Request and Response codes (see Figure 3).

```
                    +----------------------+
                    |      Application     |
                    +----------------------+
                    |  Request/Response    |  PPSP-TP
                    |----------------------|
                    |   (HTTP) Message     |
                    +----------------------+
                    |       TRANSPORT      |
                    +----------------------+
```

Figure 3: Abstract layering of PPSP-TP.

The Message layer deals with the framing format, for encoding and transmitting the data through the underlying transport protocol, as well as the asynchronous nature of the interactions between Tracker and peers.

The Transport layer is responsible for the actual transmission of requests and responses over network transports, including the determination of the connection to use for a request or response message when using a connection-oriented transport like TCP [RFC0793], or TLS [RFC5246] over it.

2.1 Messaging Model

The messaging model of PPSP-TP aligns with HTTP protocol and the semantics of its messages, currently in version 1.1 [RFC2616], but intended to support future versions of HTTP. The exchange of messages of PPSP-TP is envisioned to be performed over a stream-oriented reliable transport protocol, like TCP [RFC0793].

2.2 Request/Response model

PPSP-TP uses a REST-Like (Representational State Transfer) design [Fielding] with the goal of leveraging current HTTP implementations and infrastructure, as well as familiarity with existing REST-like services in popular use. PPSP-TP messages use the UTF-8 character set [RFC3629] and are either requests from peers to a tracker service, or responses from a tracker service to peers. The Request and Response semantics are carried as entities (header and body) in messages which correspond to either HTTP request methods or HTTP response codes, respectively.

PPSP-TP uses the HTTP POST method to send parameters in requests. PPSP-TP messages use JavaScript Object Notation (JSON) [RFC7159] to encode message bodies.

Requests are sent, and responses returned to these requests. A single request generates a single response (neglecting fragmentation of messages in transport).

The response codes are consistent with HTTP response codes, however, not all HTTP response codes are used for the PPSP-TP (Section 6.3).

The Request Messages of the base protocol are listed in Table 1:

PPSP-TP/1.0 Request Messages
CONNECT
FIND
STAT_REPORT

Table 1: Request Messages

CONNECT: This Request message is an "action signal" used when a Peer registers in the Tracker (or if already registered) to notify it about the participation in named swarm(s). The Tracker records the Peer ID, connect-time (referenced to the absolute time), peer IP addresses (and associated location information), link status and Peer Mode for the named swarm(s). The Tracker also changes the content availability of the valid named swarm(s), i.e., changes the peers lists of the corresponding swarm(s) for the requester Peer ID. On receiving a CONNECT message, the Tracker first checks the peer mode type (SEED/LEECH) for the specified swarm(s) and then decides the next steps (more details are referred in section 6.1)

FIND: This Request message is an "action signal" used by peers to request to the Tracker, whenever needed, a list of peers active in the named swarm. On receiving a FIND message, the Tracker finds the peers, listed in content status of the specified swarm that can satisfy the requesting peer's requirements, returning the list to the requesting Peer. To create the peer list, the Tracker may take peer status, capabilities and peers priority into consideration. Peer priority may be determined by network topology preference, operator policy preference, etc.

STAT_REPORT: This Request message is an "information signal" that allows an active Peer to send status (and optionally statistic data) to the Tracker to signal continuing activity. This request message MUST be sent periodically to the Tracker while the Peer is active in the system.

2.3 State Machines and Flows of the Protocol

The state machine for the tracker is very simple, as shown in Figure 4. Peer ID registrations represent a dynamic piece of state maintained by the network.

```
             --------------------------------------------
            /                                            \
           |  +------------+    +=========+    +======+   |
            \-| TERMINATED |<---| STARTED |<---| INIT |<-/
              +------------+    +=========+    +======+
               (Transient)                           \- (start tracker)
```

Figure 4: Tracker State Machine

When there are no peers connected in the Tracker, the state machine is in the INIT state.

When the "first" Peer connects for registration with its Peer ID, the state machine moves from INIT to STARTED. As long as there is at least one active registration of a Peer ID, the state machine remains in the STARTED state. When the "last" Peer ID is removed, the state machine transitions to TERMINATED. From there, it immediately transitions back to the INIT state. Because of that, the TERMINATED state here is transient.

Once in STARTED state, each Peer is instantiated (per Peer ID) in the Tracker state machine with a dedicated transaction state machine (Figure 5), which is deleted when the Peer ID is removed.

```
            --------------------------------------------
           /                                            \
          |  +------------+    +=========+    +======+   |
           \-| TERMINATED |<---| STARTED |<---| INIT |<-/
             +------------+    +=========+    +======+
              (Transient)           | (1)        \- (start tracker)
                                    V
                +-----------+   +-------+  rcv CONNECT
    (Transient) | TERMINATE |   | START |  --------------- (1)
                +-----------+   +-------+  strt init timer
rcv FIND              ^             |
rcv STAT_REPORT       |             |
on registration error |             v
on action error       |      +------------+
---------------- (A)  +<-----| PEER       | (Transient)
stop init timer       |      | REGISTERED |
snd error             |      +------------+
                      |            |
                      |            |   process swarm actions
                      |            |   --------------------- (2)
on CONNECT Error (B)  |            |   snd OK (PeerList)
on timeout       (C)  |           /    stop init timer
----------------      |          /     strt track timer
stop track timer      |         /
clean peer info       |        |
del registration      |        |             rcv FIND
snd error (B)          \       |     ----    --------------- (3)
                 ----   \      |   /      \  snd OK (PeerList)
               /      \  \     |  |        | rst track timer
rcv CONNECT   |  (4)   |  |    |  |        |
-----------   |        v  |    v  v        | rcv STAT_REPORT
snd OK         \     +==============+     /  --------------- (3)
rst track timer  ----|   TRACKING   |----    snd OK response
                     +==============+        rst track timer
```

Figure 5: Per-Peer-ID Transaction State Machine and Flow Diagram

Unlike the Tracker state machine, which exists even when no Peer IDs are registered, the "per-Peer-ID" transaction state machine is instantiated only when the Peer ID starts registration in the tracker, and is deleted when the Peer ID is de-registered/removed. This allows for an implementation optimization whereby the tracker can destroy the objects associated with the "per-Peer-ID" transaction state machine once it enters the TERMINATE state (Figure 5).

When a new Peer ID is added, the corresponding "per-Peer-ID" state machine is instantiated, and it moves into the PEER REGISTERED state. Because of that, the START state here is transient.

When the Peer ID is no longer bound to a registration, the "per-Peer-ID" state machine moves to the TERMINATE state, and the state machine is destroyed.

During the lifetime of streaming activity of a peer, the instantiated "per-Peer-ID" transaction state machine progresses from one state to another in response to various events. The events that may potentially advance the state include:

- o Reception of CONNECT, FIND and STAT_REPORT messages, or
- o Timeout events.

The state diagram in Figure 5 illustrates state changes, together with the causing events and resulting actions. Specific error conditions are not shown in the state diagram.

2.3.1 Normal Operation

On normal operation the process consists of the following steps:

1) When a Peer wants to access the system it needs to register on a tracker by sending a CONNECT message asking for the swarm(s) it wants to join. This CONNECT request from a new Peer ID triggers the instantiation in the Tracker of a "per-Peer-ID" State Machine. In the START state of the new "per-Peer-ID" SM, the Tracker initiates the registration of the Peer ID and associated information (IP addresses), starts the "init timer" and moves to PEER REGISTERED state.

2) In PEER REGISTERED state, if Peer ID is valid, the Tracker either a) processes the requested action(s) for the valid swarm information contained in the CONNECT request and in case of success the tracker stops the "init timer", starts the "track timer" and sends the response to the Peer (the response MAY contain the appropriate list of peers for the joining swarm(s), as detailed in section 6.1, or b) moves the valid FIND request to

TRACKING state.

3) In TRACKING state, STAT_REPORT or FIND messages received from that Peer ID will reset the "track timer" and are respectively responded with a) a successful condition, b) a successful condition containing the appropriate list of peers for the named swarm (section 6.2).

4) While TRACKING, a CONNECT message received from that Peer ID with valid swarm actions information (section 6.1.1) resets the "track timer" and is responded with a successful condition.

2.3.2 Error Conditions

Peers MUST NOT generate protocol elements that are invalid. However, several situations of a Peer may lead to abnormal conditions in the interaction with the Tracker. The situations may be related with Peer malfunction or communications errors. The Tracker reacts to the abnormal situations depending on its current state related to a Peer ID, as follows:

A) At PEER REGISTERED state, when a CONNECT Request only contains invalid swarm actions (section 6.1.1), the Tracker responds with error code 403 Forbidden, deletes the registration, transition to TERMINATE state for that Peer ID and the SM is destroyed.

At the PEER REGISTERED state, if the Peer ID is considered invalid (in the case of a CONNECT request or in the case of FIND or STAT_REPORT requests received from an unregistered Peer ID), the Tracker responds with either error codes 401 Unauthorized or 403 Forbidden (described in section 6.3), transitions to TERMINATE state for that Peer ID and the SM is destroyed.

B) At the TRACKING state (while the "track timer" has not expired) receiving a CONNECT message from that Peer ID with invalid swarm actions (section 5.1) is considered an error condition. The Tracker responds with error code 403 Forbidden (described in section 6.3), stops the "track timer", deletes the registration, transitions to TERMINATE state for that Peer ID and the SM is destroyed.

C) In TRACKING state, without receiving messages from the Peer, on timeout (track timer) the Tracker cleans all the information associated with the Peer ID in all swarms it was joined, deletes the registration, transitions to TERMINATE state for that Peer ID and and the SM is destroyed.

NOTE: These situations may correspond to malfunctions at the Peer or

to malicious conditions. As preventive measure, the Tracker proceeds to TERMINATE state for that Peer ID.

3 Protocol Specification: Presentation Language

PPSP-TP uses a REST-Like design, encoding the requests and responses using JSON [RFC7159].

For a generalization of the definition of protocol elements and fields, their types and structures, this document uses a C-style notation, similar to the presentation language used to define TLS [RFC5246], turning the definitions for JSON objects extensible. A JSON object consists of name/value pairs. The JSON names of the pairs are indicated with "". In this presentation language, comments begin with "//", and the "ppsp_tp_string_t" and "ppsp_tp_integer_t" types are used to indicate the JSON string and number, respectively. Optional fields are enclosed in "[ ]" double brackets. An array is indicated by two numbers in angle brackets, <min..max>, where "min" indicates the minimal number of values and "max" the maximum. An "*" is used to denote a no upper bound value for "max".

3.1 Constants

Typed constants can be defined by declaring a symbol of the desired type and assigning values to it.

3.2 Enumerated Types

A field of type "enum" can only assume the values declared in its definition, and every element of an enumerated type must be assigned with a unique value, in any order. Only "enum" fields of the same type may be assigned or compared.

```
enum { e1(v1), e2(v2), ... , en(vn) } type_name;
```

The names of the elements of an enumeration are scoped within the defined type.

3.3 Constructed Types

Similarly to enumerated types, fields of type "struct" can be constructed from primitive types and each definition declares a new and unique type.

```
struct {
        s1  e1;
        s2  e2;
        ...
        sn  en;
} type_name;
```

The names of the elements of a "struct" are scoped within the defined type. The elements within a "struct" field may be qualified using the name of the type with a syntax similar to the enumerated type.

To allow extensibility in the specification some structures must be identified using a well known code, e.g., STREAM_STATS = 0x01:

```
enum {
      STREAM_STATS = 0x01
} ppsp_tp_stat_type_t;
```

For those extensible element types, the protocol implementer can be able to access the data of the specified structure from its type code.

4 Protocol Specification: Resource Element Types

This section details the format of PPSP-TP resource element types.

4.1 Version

For both requests and responses, the version of PPSP-TP being used MUST be indicated by the attribute @version, defined as follows:

```
enum {
     PPSP_TP_BASE = 0x10
} ppsp_tp_version_t;
```

4.2 Peer Number Element

The PeerNum element is a scope selector in requests and MAY contain the attribute @ability_nat to inform the Tracker on the preferred type of peers to be returned in a peer list, related to their NAT traversal situation.

The definition of the scope selector element and attributes is defined as follows:

```
Object {
        ppsp_tp_integer_t  peer_count;
        enum {
              NO_NAT,
              STUN,
              TURN,
              PROXY
        } ability_nat;
        enum {
              NORMAL,
              LOW,
              HIGH
        } concurrent_links;
        enum {
              NORMAL,
              HIGH
        } online_time;
        enum {
              NORMAL,
              HIGH
        } upload_bandwidth_level;
} ppsp_tp_peer_num_t;
```

4.3 Swarm Action Element

The swarm action element identifies the action(s) to be taken in the named swarm(s) as well as the corresponding Peer Mode (if the peer is LEECH or SEEDER in that swarm).

```
Object {
        ppsp_tp_string_t  swarm_id;
        enum {
              JOIN,
              LEAVE
        } action;
        enum {
              SEED,
              LEECH
        } peer_mode;
        ppsp_tp_string_t  transaction_id;
} ppsp_tp_swarm_action_t;
```

The response from the Tracker upon the requested action MUST uniquely identify the corresponding transaction ID:

```
Object {
        ppsp_tp_string_t  action_id;
        ppsp_tp_response_type_t  response_type;
} ppsp_tp_swarm_action_result_t;
```

4.4 Peer Information Elements

The Peer information elements provide network identification information of peers as well as the associated swarm(s).

A Peer information consists of its identifier and the IP related addressing information, and optionally the associated swarm.

```
Object {
        ppsp_tp_string_t  peer_id;
       [ppsp_tp_string_t  swarm_id;]
        ppsp_tp_peer_addr_t  peer_address<1..*>;
} ppsp_tp_peer_info_t
```

The IP addressing information element includes the IP address and port, with a few optional attributes related with connection type and network location (in terms of ASN) as well as, optionally, the identifier of the Peer Protocol being used.

The IP address is also encoded as a string. The exact characters and format depend on address type. The IPv4 address is encoded as specified by the IPv4address rule in Section 3.2.2 of [RFC3986]. The IPv6 address is encoded as specified in section 4 of [RFC5952]. In this document IP address is specified as ppsp_tp_add_value:

```
Object {
        ppsp_tp_string_t  AddressType;
        ppsp_tp_add_value  Address;
} ppsp_tp_ip_address;
```

```
Object {
        ppsp_tp_ip_address  ip_address;
        ppsp_tp_string_t  port;
        ppsp_tp_integer_t  priority;
        enum {
              HOST,
              REFLEXIVE,
              PROXY
        } type;
        enum {
              3G,
              ADSL,
              LTE,
              ETHER
        } connection;
        ppsp_tp_string_t  asn;
       [ppsp_tp_peer_protocol_t  peer_protocol;]
} ppsp_tp_peer_addr_t;
```

The Peer Information in requests or responses is grouped in an array element:

```
Object {
        ppsp_tp_peer_info_t  peer_info<1..*>;
} ppsp_tp_peer_group_t
```

4.5 Statistics and Status Information Element

The Stat element is used to describe several properties relevant to the P2P network. These properties can be related with stream statistics and peer status information. Each Stat element will correspond to a "property" type and several Stat blocks can be reported in a single STAT_REPORT message, corresponding to some or all the swarms the peer is actively involved.

The definition of the statistic element elements and attributes are as follows:

```
Object {
        ppsp_tp_stat_type_t  type;
        union {
               Object {
                       ppsp_tp_string_t  swarm_id;
                       ppsp_tp_integer_t  uploaded_bytes;
                       ppsp_tp_integer_t  downloaded_bytes;
                       ppsp_tp_integer_t  available_bandwidth;
                   } stream_stats;
        } stat_data;
} ppsp_tp_stat_t;
```

The Stat Information in requests is grouped in an array element:

```
Object {
        ppsp_tp_stat_t  stat<1..*>;
} ppsp_tp_stat_group_t
```

Other properties may be defined, related, for example, with incentives and reputation mechanisms like "peer online time", or connectivity conditions like physical "link status", etc.

For that purpose, the Stat element may be extended to provide additional specific information for new properties, elements or attributes (guidelines in section 9).

4.6 Requests and Responses

This section defines the structure of PPSP-TP requests and responses.

4.6.1 Request Types

The request type includes CONNECT, FIND and STAT_REPORT, defined as follows:

```
enum ppsp_tp_request_type {
     PPSP_TP_CONNECT = 0x02,
     PPSP_TP_FIND = 0x04,
     PPSP_TP_STAT_REPORT = 0x08
} ppsp_tp_request_type_t;
```

4.6.2 Response Types

Response type corresponds to the response method type of the message,

defined as follows:

```
enum ppsp_tp_response_type {
          PPSP_TP_SUCCESSFUL = 0x00,
          PPSP_TP_AUTH_REQUIRED = 0x01
} ppsp_tp_response_type_t;
```

4.6.3 Request Element

The Request element MUST be present in requests and corresponds to the request method type for the message.

The generic definition of a PPSP-TP Request is the following:

```
Object {
        ppsp_tp_version_t  version;
        ppsp_tp_request_type_t  request_type;
        ppsp_tp_string_t  transaction_id;
        ppsp_tp_string_t  peer_id;
        union {
               Object {
                      [ppsp_tp_peer_num_t  peer_num;]
                      [ppsp_tp_peer_group_t  peer_group;]
                       ppsp_tp_swarm_action_t  swarm_action<1..*>;
               } connect;
               Object {
                       ppsp_tp_string_t  swarm_id;
                       ppsp_tp_peer_num_t  peer_num;
               } find;
               ppsp_tp_stat_group_t  stat_report;
        } request_data;
} ppsp_tp_request;
```

4.6.4 Response Element

The generic definition of a PPSP-TP Response is the following:

```
Object {
        ppsp_tp_version_t  version;
        ppsp_tp_response_type_t  response_type;
        ppsp_tp_string_t  transaction_id;
       [ppsp_tp_swarm_action_result_t  result<1..*>;]
       [ppsp_tp_peer_group_t  peer_group;]
} ppsp_tp_response;
```

4.7 PPSP-TP Message Element

PPSP-TP messages (requests or responses) are designed to have a similar structure with a root field named "PPSPTrackerProtocol" containing meta information and data pertaining to a request or a response.

The base type of each PPSP-TP message is defined as follows:

```
union {
       ppsp_tp_request  Request;
       ppsp_tp_response  Response;
} ppsp_tp_message_t;
```

with the field PPSPTrackerProtocol defined as:

```
Object {
        ppsp_tp_message_t  PPSPTrackerProtocol;
} ppsp_tp_message_root;
```

5 Protocol Specification: Semantics of Protocol Elements

This section describes the semantics of the PPSP-TP protocol elements and attributes. Table 2 describes the semantics of Request and Response Elements.

Element or Attribute	Use	Description
PPSPTrackerProtocol	1	The root element
@version	M	Provides the version of PPSP-TP
Request	0...1	Provides the request method and MUST be present in Request
Response	0...1	Provides the response method and MUST be present in Response
TransactionID	M	Root transaction Identification
Result	0...N	Result of @action MUST be present in Responses
@transactionID	CM	Identifier of the @action
PeerID	0...1	Peer Identifier MUST be present in Request
SwarmID	0...N	Swarm Identifier MUST be present in Requests
@action	CM	Must be set to JOIN or LEAVE
@peerMode	CM	Mode of Peer participation in the swarm, LEECH or SEED
@transactionID	CM	Identifier for the @action
PeerNUM	0...1	Maximum peers in PeeerList, =<30
@abilityNAT	OP	Type of NAT traversal peers, as No-NAT, STUN, TURN or PROXY
@concurrentLinks	OP	Concurrent connectivity level of peers, HIGH, LOW or NORMAL
@onlineTime	OP	Availability or online duration of peers, HIGH or NORMAL
@uploadBWlevel	OP	Upload bandwidth capability of peers, HIGH or NORMAL
PeerGroup	0...1	Information on peers (Table 3)
StatisticsGroup	0...1	Statistic data of peer (Table 4)

Legend:
Use for attributes: M=Mandatory, OP=Optional,
CM=Conditionally Mandatory
Use for elements: minOccurs...maxOccurs (N=unbounded)
Elements are represented by their name (case-sensitive)
Attribute names (case-sensitive) are preceded with an @

Table 2: Semantics of the Request and Response Elements.

The PeerNum element in Table 2 is a scope selector that MAY be present in CONNECT and FIND requests and MAY contain the attribute @ability_nat to inform the Tracker on the preferred type of peers to be returned in a peer list, in what concerns their NAT traversal situation.

The semantics of Peer Information elements and attributes are described in Table 3.

Element or Attribute	Use	Description
PeerGroup	0...1	Contains description of peers
PeerInfo	1...N	Provides information on a peer
@swarmID	0...1	Swarm Identifier
PeerID	0...1	Peer Identifier
PeerAddress	0...N	IP Address information
@addrType	M	Type of IP address, which can be ipv4 or ipv6
@priority	CM	The priority of this interface
@type	CM	Describes the address for NAT traversal, which can be HOST REFLEXIVE or PROXY
@connection	OP	Access type (3G, ADSL, etc.)
@asn	OP	Autonomous System Number
@ip	M	IP address value
@port	M	IP service port value
@peerProtocol	OP	PPSP Peer Protocol supported

Legend:
Use for attributes: M=Mandatory, OP=Optional,
CM=Conditionally Mandatory
Use for elements: minOccurs...maxOccurs (N=unbounded)
Elements are represented by their name (case-sensitive)
Attribute names (case-sensitive) are preceded with an @

Table 3: Semantics of Peer Information elements and attributes.

If STUN-like functions are enabled in the tracker and a PPSP-ICE method [RFC5245] is used the PeerAddress attributes @type and @priority MUST be returned with the transport address candidates in responses to CONNECT requests.

The @asn attribute MAY be used to inform about the network location, in terms of Autonomous System, for each of the active public network interfaces of the peer.

The @connection attribute is informative on the type of access network of the respective interface.

Element or Attribute	Use	Description
StatisticsGroup	0...1	Provides statistic data on peer and content
Stat	1...N	Groups statistics property data
@property	M	The property to be reported Property values and elements in Table 5

Legend:
Use for attributes: M=Mandatory, OP=Optional, CM=Conditionally Mandatory
Use for elements: minOccurs...maxOccurs (N=unbounded)
Elements are represented by their name (case-sensitive)
Attribute names (case-sensitive) are preceded with an @

Table 4: Semantics of Statistics and Status Information Elements.

Each Stat element in Table 4 will correspond to a @property type and several Stat blocks can be reported in a single STAT_REPORT message, corresponding to some or all the swarms the peer is actively involved. The @property "StreamStatistics" is described in Table 5.

Element or Attribute	Use	Description
"StreamStatistics"		Property for swarm statistics
SwarmID	0...1	Swarm Identifier
UploadedBytes	0...1	Bytes sent to swarm
DownloadedBytes	0...1	Bytes received from swarm
AvailBandwidth	0...1	Upstream Bandwidth available

Legend:
Use for attributes: M=Mandatory, OP=Optional, CM=Conditionally Mandatory
Use for elements: minOccurs...maxOccurs (N=unbounded)
Elements are represented by their name (case-sensitive)
Attribute names (case-sensitive) are preceded with an @

Table 5: Semantics of "StreamStatistics" property.

Other properties may be defined, related, for example, with

incentives and reputation mechanisms like "peer online time", or connectivity conditions like physical "link status", etc.

For that purpose, the Stat element may be extended to provide additional scheme specific information for new @property groups, new sibling elements and new attributes (guidelines in section 9).

6 Protocol Specification: Encoding and Operation

PPSP-TP is a message-oriented request/response protocol. PPSP-TP messages use a text type encoding in JSON [RFC7159], which MUST be indicated in the Content-Type field in HTTP/1.1 [RFC2616], specifying the generic application/json media type for all PPSP-TP request parameters and responses.

Implementations MUST support the "https" URI scheme [RFC2818] and Transport Layer Security (TLS) [RFC5246].

For deployment scenarios where Peer (Client) authentication is desired at the Tracker, HTTP Digest Authentication MUST be supported, with TLS Client Authentication as the preferred mechanism, if available.

Upon reception, a message is examined to ensure that it is properly formed. The receiver MUST check that the HTTP message itself is properly formed, and if not, appropriate standard HTTP errors MUST be generated.

6.1 Requests

The section describes the operation of the three types of Requests of PPSP-TP and provides some examples of usage.

PPSP-TP uses the HTTP POST method to send parameters in requests to provide information resources that are the function of one or more of those input parameters. Input parameters are encoded in JSON in the HTTP entity body of the request.

6.1.1 CONNECT Request

This method is used when a peer registers to the system and/or requests swarm actions. The peer MUST properly set the Request type to CONNECT, generate and set the TransactionIDs, set the PeerInfo and MAY include the swarm the peer is interested in, followed by the corresponding action_type and peer_mode.

- o When a peer already possesses a content and agrees to share it to others, it should set the action_type to the value JOIN, as well as set the peer_mode to SEED during its start (or re-start) period.

- o When a peer makes a request to join a swarm to consume content, it should set the action_type to the value JOIN, as well as set the peer_mode to LEECH during its start (or re-start) period.

In the above cases, the peer can provide optional information on the

addresses of its network interface(s), for example, the priority, type, connection and ASN.

When a peer plans to leave a previously joined swarm, it should set action_type to LEAVE, regardless of the peer_mode.

When receiving a well-formed CONNECT Request message, the tracker MAY, when applicable, start by pre-processing the peer authentication information (provided as Authorization scheme and token in the HTTP message) to check whether it is valid and that it can connect to the service, then proceed to register the peer in the service and perform the swarm actions requested. In case of success a Response message with a corresponding response value of SUCCESSFUL will be generated.

The valid sets of SwarmID whose action_type is combined with peer_mode for the CONNECT Request logic are enumerated in Table 6 (referring to the tracker "per-Peer-ID" state machine in Section 2.3).

SwarmID Elements	@peerMode value	@action value	Initial State	Final State	Request validity
1	LEECH	JOIN	START	TRACKING	Valid
1	LEECH	LEAVE	START	TERMINATE	Invalid
1	LEECH	LEAVE	TRACKING	TERMINATE	Valid
1 1	LEECH LEECH	JOIN LEAVE	START	TERMINATE	Invalid
1 1	LEECH LEECH	JOIN LEAVE	TRACKING	TRACKING	Valid
N	SEED	JOIN	START	TRACKING	Valid
N	SEED	JOIN	TRACKING	TERMINATE	Invalid
N	SEED	LEAVE	TRACKING	TERMINATE	Valid

Table 6: Validity of @action combinations in CONNECT Request.

In the CONNECT Request the element SwarmID MUST be present with cardinality 1 to N, each containing the request for @action, the @peerMode of the peer and the child @transactionID for that swarm. The @peerMode element MUST be set to the type of participation of the

peer in the swarm (SEED or LEECH).

The element PeerInfo, if present, MAY contain multiple PeerAddress child elements with attributes @addrType, @ip, @port and @peerProtocol, and optionally @priority and @type (if PPSP-ICE NAT traversal techniques are used) corresponding to each of the network interfaces the peer wants to advertise.

The element PeerNum indicates to the tracker the number of peers to be returned in a list corresponding to the indicated properties, being @abilityNAT for NAT traversal (considering that PPSP-ICE NAT traversal techniques may be used), and optionally @concurrentLinks, @onlineTime and @uploadBWlevel for the preferred capabilities. If STUN-like function is enabled in the tracker, the response MAY include the peer reflexive address.

The element Transaction_ID MUST be present in requests to uniquely identify the transaction. Responses to completed transactions use the same TransactionID as the request they correspond to.

The Response MUST include PeerInfo data of the requesting peer public IP address. If STUN-like function is enabled in the tracker, the PeerAddress includes the attribute @type with a value of REFLEXIVE, corresponding to the transport address "candidate" of the peer. The PeerGroup MAY also include PeerInfo data corresponding to the Peer IDs and public IP addresses of the selected active peers in the requested swarm. The tracker MAY also include the attribute @asn with network location information of the transport address, corresponding to the Autonomous System Number of the access network provider of the referenced peer.

In case the @peerMode is SEED, the tracker responds with a SUCCESSFUL response and enters the peer information into the corresponding swarm activity. In case the @peerMode is LEECH (or if the peer Seeder includes a PeerNum element in the request) the tracker will search and select an appropriate list of peers satisfying the conditions set by the requesting peer. The peer list returned MUST contain the Peer IDs and the corresponding IP Addresses. To create the peer list, the tracker may take peer status and network location information into consideration, to express network topology preferences or Operators' policy preferences, with regard to the possibility of connecting with other IETF efforts such as ALTO [RFC7285].

IMPLEMENTATION NOTE: If no PeerNum attributes are present in the request the tracker MAY return a random sample from the peer population.

6.1.1.1 Example

The following example of a CONNECT Request corresponds to a peer that wants to start (or re-start) sharing its previously streamed contents (peerMode is of SEED). Note for this case that the peer also requests from the Tracker an appropriate list of peers (PeerNum element) already active in the swarm, i.e., a list of 15 peers having STUN capabilities in terms of NAT. In the case of a Super-Node peer of an ISP, the CONNECT request would be similar but, optionally not including the PeerNum element:

```
POST / HTTP/1.1
Host: tracker.example.com
Content-Length: 494
Content-Type: application/ppsp+json
Accept: application/ppsp+json
```


```
{
  "PPSPTrackerProtocol": {
    "@version":              "1.0",
    "Request":               "CONNECT",
    "PeerID":                "656164657220",
    "PeerNum": {
      "@abilityNAT":         "STUN",
      "$":                   15
    },
    "SwarmID": [
      {
        "@action":           "JOIN",
        "@peerMode":         "SEED",
        "@transactionID":    "12345.1",
        "$":                 "1111"
      },
      {
        "@action":           "JOIN",
        "@peerMode":         "SEED",
        "@transactionID":    "12345.2",
        "$":                 "2222"
      }
    ],
    "TransactionID":         "12345.0"
  }
}
```

Another example of the message-body of a CONNECT Request corresponds to a peer (PeerMode is LEECH, meaning that the peer is not in possession of the content) requesting join to a swarm, in order to start receiving the stream, and providing optional information on the addresses of its network interface(s):


```
{
  "PPSPTrackerProtocol": {
    "@version":               "1.0",
    "Request":                "CONNECT",
    "PeerID":                 "656164657221",
    "PeerNum": {
      "@abilityNAT":          "STUN",
      "$":                    5
    },
    "SwarmID": {
      "@action":              "JOIN",
      "@peerMode":            "LEECH",
      "@transactionID":       "12345.1",
      "$": "1111"
    },
    "TransactionID":          "12345.0",
    "PeerGroup": {
      "PeerInfo": {
        "PeerAddress": [
          {
            "@addrType":     "ipv4",
            "@ip":           "192.0.2.2",
            "@port":         "80",
            "@priority":     1,
            "@peerProtocol": "PPSP-PP"
          },
          {
            "@addrType":     "ipv6",
            "@ip":           "2001:db8::2",
            "@port":         "80",
            "@priority":     2,
            "@peerProtocol": "PPSP-PP"
          }
        ]
      }
    }
  }
}
```

The next example of a CONNECT Request corresponds to a peer "leaving" a previously joined swarm and requesting join to a new swarm. This is the typical example of a user watching a live channel but then deciding to switch to a different one:


```
{
  "PPSPTrackerProtocol": {
    "@version":               "1.0",
    "Request":                "CONNECT",
    "PeerID":                 "656164657221",
    "PeerNum": {
      "@abilityNAT":          "STUN",
      "$":                    5
    },
    "SwarmID": [
      {
        "@action":            "LEAVE",
        "@peerMode":          "LEECH",
        "@transactionID":     "12345.1",
        "$":                  "1111"
      },
      {
        "@action":            "JOIN",
        "@peerMode":          "LEECH",
        "@transactionID":     "12345.2",
        "$":                  "2222"
      }
    ],
    "TransactionID":          "12345.0"
  }
}
```

The next example illustrates the Response for the previous example of CONNECT Request where the peer requested two swarm actions and not more than 5 other peers, receiving from the Tracker a peer list with only 2 two other peers in the swarm "2222":

```
HTTP/1.1 200 OK
Content-Length: 1342
Content-Type: application/ppsp+json
```


```
{
  "PPSPTrackerProtocol": {
    "@version":              "1.0",
    "Response":              "SUCCESSFUL",
    "TransactionID": {
      "Result": [
        {
          "@transactionID":  "12345.0",
          "$":               "200 OK"
        },
        {
          "@transactionID":  "12345.1",
          "$":               "200 OK"
        },
        {
          "@transactionID":  "12345.2",
          "$":               "200 OK"
        }
      ]
    },
    "PeerGroup": {
      "PeerInfo": [
        {
          "PeerID":          "656164657221",
          "PeerAddress": {
            "@addrType":     "ipv4",
            "@ip":           "198.51.100.1",
            "@port":         "80",
            "@priority":     1,
            "@type":         "REFLEXIVE",
            "@connection":   "3G",
            "@asn":          "64496"
          }
        },
        {
          "@swarmID":        "2222",
          "PeerID":          "956264622298",
          "PeerAddress": {
            "@addrType":     "ipv4",
```

```
                  "@ip":           "198.51.100.22",
                  "@port":         "80",
                  "@asn":          "64496",
                  "@peerProtocol": "PPSP-PP"
                }
              },
              {
                "@swarmID":        "2222",
                "PeerID":          "3332001256741",
                "PeerAddress": {
                  "@addrType":     "ipv4",
                  "@ip":           "198.51.100.201",
                  "@port":         "80",
                  "@asn":          "64496",
                  "@peerProtocol": "PPSP-PP"
                }
              }
            ]
          }
        }
      }
```


6.1.2 FIND Request

This method allows peers to request to the tracker, whenever needed, a new peer list for the swarm.

The FIND request MAY include a peer_number element to indicate to the tracker the maximum number of peers to be returned in a list corresponding to the indicated conditions set by the requesting peer, being AbilityNAT for NAT traversal (considering that PPSP-ICE NAT traversal techniques may be used), and optionally ConcurrentLinks, OnlineTime and UploadBWlevel for the preferred capabilities.

When receiving a well-formed FIND Request the tracker processes the information to check if it is valid. In case of success a response message with a Response value of SUCCESSFUL will be generated and the tracker will search out the list of peers for the swarm and select an appropriate peer list satisfying the conditions set by the requesting peer. The peer list returned MUST contain the Peer IDs and the corresponding IP Addresses.

The tracker may take peer status and network location information into consideration when selecting the peer list to return, to express network topology preferences or Operators' policy preferences, with regard to the possibility of connecting with other IETF efforts such as ALTO [RFC7285].

To provide more choices for the requesting peer, the tracker may select a new peer list with lower priority from the list of peers and return it to the requesting peer later.

The Response MUST include PeerInfo data that includes the public IP addresses of the selected active peers in the swarm.

The peer list MUST contain the Peer IDs and the corresponding IP Addresses, MAY also include the attribute ASN with network location information of the transport address, corresponding to the Autonomous System Number of the access network provider of the referenced peer.

The tracker MAY also include the attribute @asn with network location information of the transport addresses of the peers, corresponding to the Autonomous System Numbers of the access network provider of each peer in the list.

The response MAY also include PeerInfo data that includes the requesting peer public IP address. If STUN-like function is enabled in the tracker, the PeerAddress includes the attribute @type with a value of REFLEXIVE, corresponding to the transport address "candidate" of the peer.

IMPLEMENTATION NOTE: If no PeerNum attributes are present in the request the tracker MAY return a random sample from the peer population.

6.1.2.1 Example

An example of the message-body of a FIND Request, where the peer requests to the Tracker an list of not more than 5 peers in the swarm "1111" conforming to the characteristics expressed (concurrent links, online time, and upload bandwidth level) is the following:


```
{
  "PPSPTrackerProtocol": {
    "@version":              "1.0",
    "Request":               "FIND",
    "PeerID":                "656164657221",
    "SwarmID":               "1111",
    "TransactionID":         "12345",
    "PeerNum": {
      "@abilityNAT":         "STUN",
      "@concurrentLinks":    "HIGH",
      "@onlineTime":         "NORMAL",
      "@uploadBWlevel":      "NORMAL",
      "$":                   5
    }
  }
}
```


An example of the message-body of a Response for the above FIND Request, including the requesting peer public IP address information, is the following:


```
{
  "PPSPTrackerProtocol": {
    "@version":              "1.0",
    "Response":              "SUCCESSFUL",
    "TransactionID":         "12345",
    "PeerGroup": {
      "PeerInfo": [
        {
          "PeerID":          "656164657221",
          "PeerAddress": {
            "@addrType":     "ipv4",
            "@ip":           "198.51.100.1",
            "@port":         "80",
            "@priority":     1,
            "@type":         "REFLEXIVE",
            "@connection":   "3G",
            "@asn":          "64496"
          }
        },
        {
```

```
              "@swarmID":       "1111",
              "PeerID":         "956264622298",
              "PeerAddress": {
                "@addrType":    "ipv4",
                "@ip":          "198.51.100.22",
                "@port":        "80",
                "@asn":         "64496"
              }
            },
            {
              "@swarmID":       "1111",
              "PeerID":         "3332001256741",
              "PeerAddress": {
                "@addrType":    "ipv4",
                "@ip":          "198.51.100.201",
                "@port":        "80",
                "@asn":         "64496"
              }
            }
          ]
        }
      }
    }
```

6.1.3 STAT_REPORT Request

This method allows peers to send status and statistic data to trackers. The method is initiated by the peer, periodically while active.

The peer MUST set the Request method to STAT_REPORT, set the PeerID with the identifier of the peer, and generate and set the TransactionID.

The report MAY include multiple statistics elements describing several properties relevant to a specific swarm. These properties can be related with stream statistics and peer status information, including ploadedBytes, DownloadedBytes, AvailBandwidth and etc.

Other properties may be defined (guidelines in section 7.1) related for example, with incentives and reputation mechanisms. In case no StatisticsGroup is included, the STAT_REPORT is used as a "keep-alive" message to prevent the tracker from de-registering the peer when "track timer" expires.

If the request is valid the tracker processes the received information for future use, and generates a response message with a

Response value of SUCCESSFUL.

The response MUST have the same TransactionID value as the request.

6.1.3.1 Example

An example of the message-body of a STAT_REPORT Request is:


```
{
  "PPSPTrackerProtocol": {
    "@version":             "1.0",
    "Request":              "STAT_REPORT",
    "PeerID":               "656164657221",
    "TransactionID":        "12345",
    "StatisticsGroup": {
      "Stat": {
        "@property":        "StreamStatistics",
        "SwarmID":          "1111",
        "UploadedBytes":    512,
        "DownloadedBytes":  768,
        "AvailBandwidth":   1024000
      }
    }
  }
}
```


An example of the message-body of a Response for the START_REPORT Request is:


```
{
  "PPSPTrackerProtocol": {
    "@version":             "1.0",
    "Response":             "SUCCESSFUL",
    "TransactionID":        "12345"
  }
}
```


6.2 Response element in response Messages

Response messages not requiring message-body only use the standard HTTP Status-Code and Reason-Phrase (appended, if appropriate, with detail phrase, as described in Section 6.3).

Otherwise, the response elements will include the response elements, related with the corresponding requests. Table 7 indicates the HTTP Status-Code and Reason-Phrase for Response messages that require message-body. These values MUST be treated as case-sensitive.

Response Element	HTTP Status-Code and Reason-Phrase
SUCCESSFUL	200 OK
AUTH_REQUIRED	401 Unauthorized

Table 7: Valid Strings for Response element of responses.

SUCCESSFUL: indicates that the request has been processed properly and the desired operation has completed. The body of the response message includes the requested information and MUST include the same TransactionID of the corresponding request.

In CONNECT Request: returns information about the successful registration of the peer and/or of each swarm @action requested. MAY additionally return the list of peers corresponding to the join @action requested.

In FIND Request: returns the list of peers corresponding to the requested scope.

In STAT_REPORT Request: confirms the success of the requested operation.

AUTH_REQUIRED: authentication is required for the peer to make the request.

6.3 Error and Recovery conditions

If the peer fails to read the tracker response, the same Request with identical content, including the same TransactionID, SHOULD be repeated, if the condition is transient.

The TransactionID on a Request can be reused if and only if all of the content is identical, including Date/Time information. Details of the retry process (including time intervals to pause, number of retries to attempt, and timeouts for retrying) are implementation dependent.

The tracker SHOULD be prepared to receive a Request with a repeated TransactionID.

Error situations resulting from the Normal Operation or from abnormal conditions (Section 2.3.2) MUST be responded with the adequate response codes, as described here:

If the message is found to be incorrectly formed, the receiver MUST respond with a 400 (Bad Request) response with an empty message-body. The Reason-Phrase SHOULD identify the syntax problem in more detail, for example, "Missing Content-Type header field".

If the version number of the protocol is for a version the receiver does not supports, the receiver MUST respond with a 400 (Bad Request) with an empty message-body. Additional information SHOULD be provided in the Reason-Phrase, for example, "PPSP Version #.#".

If the length of the received message does not matches the Content-Length specified in the message header, or the message is received without a defined Content-Length, the receiver MUST respond with a 411 (Length Required) response with an empty message-body.

If the Request-URI in a Request message is longer than the tracker is willing to interpret, the tracker MUST respond with a 414 (Request-URI Too Long) response with an empty message-body.

In the PEER REGISTERED and TRACKING states of the tracker, certain requests are not allowed (Section 2.3.2). The tracker MUST respond with a 403 (Forbidden) response with an empty message-body. The Reason-Phrase SHOULD identify the error condition in more detail, for example, "Action not allowed".

If the tracker is unable to process a Request message due to unexpected condition, it SHOULD respond with a 500 (Internal Server Error) response with an empty message-body.

If the tracker is unable to process a Request message for being in an overloaded state, it SHOULD respond with a 503 (Service Unavailable) response with an empty message-body.

6.4 Parsing of Unknown Fields in Message-body

This document only details object fields used by this specification. Extensions may include additional fields within JSON objects defined in this document. PPSP-TP implementations MUST ignore unknown fields when processing PPSP-TP messages.

7 Operations and Manageability

This section provides the operational and managements aspects that are required to be considered in implementations of the PPSP Tracker Protocol. These aspects follow the recommendations expressed in RFC 5706 [RFC5706].

7.1 Operational Considerations

The PPSP-TP provides communication between trackers and peers and is conceived as a "client-server" mechanism, allowing the exchange of information about the participant peers sharing multimedia streaming contents.

The "serving" component, i.e., the Tracker, is a logical entity that can be envisioned as a centralized service (implemented in one or more physical nodes), or a fully distributed service.

The "client" component can be implemented at each peer participating in the streaming of contents.

7.1.1 Installation and Initial Setup

Content providers wishing to use PPSP for content distribution should setup at least a PPSP Tracker and a service Portal (public web server) to publish links of the content descriptions, for access to their on-demand or live original contents sources. Content/Service providers should also create conditions to generate Peer IDs and any required security certificates, as well as Chunk IDs and Swarm IDs for each streaming content. The configuration processes for the PPSP Tracking facility, the service Portal and content sources are not standardized, enabling all the flexibility for implementers.

The Swarm IDs of available contents, as well as the addresses of the PPSP Tracking facility, can be distributed to end-users in various ways, but it is common practice to include both the Swarm ID and the corresponding PPSP Tracker addresses (as URLs) in the MPD of the content, which is obtainable (a link) from the service Portal.

End-users browse and search for the desired contents in the service Portal, selecting by clicking the links of the corresponding MPDs. This action typically launches the Client Media Player (with PPSP awareness) which will then, using PPSP-TP, contact the PPSP Tracker to join the corresponding swarm and obtain the transport addresses of other PPSP peers in order to start streaming the content.

7.1.2 Migration Path

Since there is no previous standard protocol providing similar functionality, this specification does not details a migration path.

7.1.3 Requirements on Other Protocols and Functional Components

For security reasons, when using PPSP Peer protocol with PPSP-TP, the mechanisms described in section 8.1 should be observed.

7.1.4 Impact on Network Operation

As the messaging model of PPSP-TP aligns with HTTP protocol and the semantics of its messages, the impact on Network Operation is similar to using HTTP.

7.1.5 Verifying Correct Operation

The correct operation of PPSP-TP can be verified both at the Tracker and at the peer by logging the behavior of PPSP-TP. Additionally, the PPSP Tracker collects the status of the peers including peer's activity, and such information can be used to monitor and obtain the global view of the operation.

7.2 Management Considerations

The management considerations for PPSP-TP are similar to other solutions using HTTP for large-scale content distribution. The PPSP Tracker can be realized by geographically distributed tracker nodes or multiple server nodes in a data center. As these nodes are akin to WWW nodes, their configuration procedures, detection of faults, measurement of performance, usage accounting and security measures can be achieved by standard solutions and facilities.

7.2.1 Interoperability

Interoperability refers to allowing information sharing and operations between multiple devices and multiple management applications. For PPSP-TP, distinct types of devices host PPSP-TP servers (Trackers) and clients (Peers). Therefore, support for multiple standard schema languages, management protocols and information models, suited to different purposes, was considered in the PPSP-TP design. Specifically, management functionalities for PPSP-TP devices can be achieved with Simple Network Management Protocol (SNMP) [RFC3410], syslog [RFC5424] and NETCONF [RFC6241].

7.2.2 Management Information

PPSP Trackers may implement SNMP management interfaces, namely the Application Management MIB [RFC2564] without the need to instrument the Tracker application itself. The channel, connections and transaction objects of the the Application Management MIB can be used to report the basic behavior of the PPSP Tracker service.

The Application Performance Measurement MIB (APM-MIB) [RFC3729] and the Transport Performance Metrics MIB (TPM-MIB) [RFC4150] can be used with PPSP-TP, providing adequate metrics for the analysis of performance for transaction flows in the network, in direct relationship to the transport of PPSP-TP.

The Host Resources MIB [RFC2790] can be used to supply information on the hardware, the operating system, and the installed and running software on a PPSP Tracker host.

The TCP-MIB [RFC4022] can additionally be considered for network monitoring.

Logging is an important functionality for PPSP-TP server (Tracker) and client (Peer), done via syslog [RFC5424].

7.2.3 Fault Management

As PPSP Tracker failures can be mainly attributed to host or network conditions, the facilities previously described for verifying the correct operation of PPSP-TP and the management of PPSP Tracker servers, appear sufficient for PPSP-TP fault monitoring.

7.2.4 Configuration Management

PPSP Tracker deployments, when realized by geographically distributed tracker nodes or multiple server nodes in a data center, may benefit from a standard way of replicating atomic configuration updates over a set of server nodes. This functionality can be provided via NETCONF [RFC6241].

7.2.5 Accounting Management

PPSP-TP implementations, namely for content provider environments, can benefit from accounting standardization efforts as defined in [RFC2975], in terms of resource consumption data, for the purposes of capacity and trend analysis, cost allocation, auditing, and billing.

7.2.6 Performance Management

Being transaction-oriented, PPSP-TP performance, in terms of availability and responsiveness, can be measured with the facilities of the APM-MIB [RFC3729] and the TPM-MIB [RFC4150].

7.2.7 Security Management

Standard SNMP notifications for PPSP Tracker management and syslog messages [RFC5424] can be used, to alert operators to the conditions identified in the security considerations (Section 8).

The statistics collected about the operation of PPSP-TP can be used for detecting attacks, such as the receipt of malformed messages, messages out of order, or messages with invalid timestamps.

8 Security Considerations

P2P streaming systems are subject to attacks by malicious/unfriendly peers/trackers that may eavesdrop on signaling, forge/deny information/knowledge about streaming content and/or its availability, impersonating to be another valid participant, or launch DoS attacks to a chosen victim.

No security system can guarantees complete security in an open P2P streaming system where participants may be malicious or uncooperative. The goal of security considerations described here is to provide sufficient protection for maintaining some security properties during the tracker-peer communication even in the face of a large number of malicious peers and/or eventual distrustful trackers (under the distributed tracker deployment scenario).

Since the protocol uses HTTP to transfer signaling most of the same security considerations described in RFC 2616 also apply [RFC2616].

8.1 Authentication between Tracker and Peers

To protect the PPSP-TP signaling from attackers pretending to be valid peers (or peers other than themselves) all messages received in the tracker SHOULD be received from authorized peers. For that purpose a peer SHOULD enroll in the system via a centralized enrollment server. The enrollment server is expected to provide a proper Peer ID for the peer and information about the authentication mechanisms. The specification of the enrollment method and the provision of identifiers and authentication tokens is out of scope of this specification.

A channel-oriented security mechanism should be used in the

communication between peers and tracker, such as the Transport Layer Security (TLS) to provide privacy and data integrity.

Due to the transactional nature of the communication between peers and tracker the method for adding authentication and data security services can be the OAuth 2.0 Authorization [RFC6749] with bearer token, which provides the peer with the information required to successfully utilize an access token to make protected requests to the tracker [RFC6750].

8.2 Content Integrity protection against polluting peers/trackers

Malicious peers may declaim ownership of popular content to the tracker but try to serve polluted (i.e., decoy content or even virus/trojan infected contents) to other peers.

This kind of pollution can be detected by incorporating integrity verification schemes for published shared contents. As content chunks are transferred independently and concurrently, a correspondent chunk-level integrity verification MUST be used, checked with signed fingerprints received from authentic origin.

8.3 Residual attacks and mitigation

To mitigate the impact of Sybil attackers, impersonating a large number of valid participants by repeatedly acquiring different peer identities, the enrollment server SHOULD carefully regulate the rate of peer/tracker admission.

There is no guarantee that peers honestly report their status to the tracker, or serve authentic content to other peers as they claim to the tracker. It is expected that a global trust mechanism, where the credit of each peer is accumulated from evaluations for previous transactions, may be taken into account by other peers when selecting partners for future transactions, helping to mitigate the impact of such malicious behaviors. A globally trusted tracker MAY also take part of the trust mechanism by collecting evaluations, computing credit values and providing them to joining peers.

8.4 Pro-incentive parameter trustfulness

Property types for STAT_REPORT messages may consider additional pro-incentive parameters (guidelines for extension in Section 9), which can enable the tracker to improve the performance of the whole P2P streaming system. Trustworthiness of these pro-incentive parameters is critical to the effectiveness of the incentive mechanisms. Furthermore, both the amount of uploaded and downloaded data should be reported to the tracker to allow checking if there is any

inconsistency between the upload and download report, and establish an appropriate credit/trust system.

One such solution could be a reputation-incentive mechanism, based on the notions of reputation, social awareness and fairness. The mechanism would promote cooperation among participants (via each peer's reputation) based on the history of past transactions, such as, count of chunk requests (sent, received) in a swarm, contribution time of the peer, cumulative uploaded and downloaded content, JOIN and LEAVE timestamps, attainable rate, etc.

Alternatively, exchange of cryptographic receipts signed by receiving peers can be used to attest to the upload contribution of a peer to the swarm, as suggested in [Contracts].

9 Guidelines for Extending PPSP-TP

Extension mechanisms allow designers to add new features or to customize existing features of a protocol for different operating environments [RFC6709].

Extending a protocol implies either the addition of features without changing the protocol itself or the addition of new elements creating new versions of an existing schema and therefore new versions of the protocol.

In PPSP-TP it means that an extension MUST NOT alter an existing protocol schema as the changes would result in a new version of an existing schema, not an extension of an existing schema, typically non-backwards-compatible.

Additionally, a designer MUST remember that extensions themselves MAY also be extensible.

Extensions MUST adhere to the principles described in this section in order to be considered valid.

Extensions MAY be documented as Internet-Draft and RFC documents if there are requirements for coordination, interoperability, and broad distribution.

Extensions need not be published as Internet-Draft or RFC documents if they are intended for operation in a closed environment or are otherwise intended for a limited audience.

9.1 Forms of PPSP-TP Extension

In PPSP-TP two extension mechanisms can be used: a Request-Response

Extension or a Protocol-level Extension.

- o Request-Response Extension: Adding elements or attributes to an existing element mapping in the schema is the simplest form of extension. This form should be explored before any other. This task can be accomplished by extending an existing element mapping.

  For example, an element mapping for the StatisticsGroup can be extended to include additional elements needed to express status information about the activity of the peer, such as OnlineTime for the Stat element.

- o Protocol-level Extension: If there is no existing element mapping that can be extended to meet the requirements and the existing PPSP-TP Request and Response message structures are insufficient, then extending the protocol should be considered in order to define new operational Requests and Responses.

  For example, to enhance the level of control and the granularity of the operations, a new version of the protocol with new messages (JOIN, DISCONNECT), a retro-compatible change in semantics of an existing CONNECT Request/Response and an extension in STAT_REPORT could be considered.

  As illustrated in Figure 6, the peer would use an enhanced CONNECT Request to perform the initial registration in the system. Then it would JOIN a first swarm as SEEDER, later JOIN a second swarm as LEECH, and then DISCONNECT from the latter swarm but keeping as SEEDER for the first one. When deciding to leave the system, the peer DISCONNECTs gracefully from it:

```
                 +--------+                       +---------+
                 |  Peer  |                       | Tracker |
                 +--------+                       +---------+
                     |                                |
                     |--CONNECT--------------------->|
                     |<--------------------------OK--|
                     |--JOIN(swarm_a;SEED)---------->|
                     |<--------------------------OK--|
                     :                                :
                     |--STAT_REPORT(activity)------->|
                     |<--------------------------Ok--|
                     :                                :
                     |--JOIN(swarm_b;LEECH)--------->|
                     |<-----------------OK+PeerList--|
                     :                                :
                     |--STAT_REPORT(ChunkMap_b)----->|
                     |<--------------------------Ok--|
                     :                                :
                     |--DISCONNECT(swarm_b)--------->|
                     |<--------------------------Ok--|
                     :                                :
                     |--STAT_REPORT(activity)------->|
                     |<--------------------------Ok--|
                     :                                :
                     |--DISCONNECT------------------>|
                     |<---------------------Ok(BYE)--|
```

Figure 6: Example of a session for a PPSP-TP extended version.

9.2 Issues to Be Addressed in PPSP-TP Extensions

There are several issues that all extensions should take into consideration.

- Overview of the Extension: It is RECOMMENDED that extensions to PPSP-TP have a protocol overview section that discusses the basic operation of the extension. The most important processing rules for the elements in the message flows SHOULD also be mentioned.

- Backward Compatibility: One of the most important issues to consider is whether the new extension is backward compatible with the base PPST-TP.

- Syntactic Issues: Extensions that define new Request/Response methods SHOULD use all capitals for the method name, keeping with a long-standing convention in many protocols, such as HTTP. Method names are case sensitive in PPSP-TP. Method names SHOULD be shorter than 16 characters and SHOULD attempt to convey the

general meaning of the Request or Response.

- Semantic Issues: PPSP-TP extensions MUST clearly define the semantics of the extensions. Specifically, the extension MUST specify the behaviors expected from both the Peer and the Tracker in processing the extension, with the processing rules in temporal order of the common messaging scenario.

  Processing rules generally specify actions to be taken on receipt of messages and expiration of timers.

  The extension SHOULD specify procedures to be taken in exceptional conditions that are recoverable. Handling of unrecoverable errors does not require specification.

- Security Issues: Being security an important component of any protocol, designers of PPSP-TP extensions need to carefully consider security requirements, namely authorization requirements and requirements for end-to-end integrity.

- Examples of Usage: The specification of the extension SHOULD give examples of message flows and message formatting and include examples of messages containing new syntax. Examples of message flows should be given to cover common cases and at least one failure or unusual case.

10 IANA Considerations

This document defines registry for application/ppsp+json media types.

Type name: application

Subtype name: ppsp+json

Required parameters: n/a

Optional parameters: n/a

Encoding considerations: Encoding considerations are identical to those specified for the "application/json" media type. See [RFC7159].

Security considerations: See Section 8.

Interoperability considerations: This document specifies format of conforming messages and the interpretation thereof.

Published specification: This document.

Applications that use this media type: PPSP trackers and peers either stand alone or embedded within other applications.

Additional information:

Magic number(s): n/a

File extension(s): This document uses the MIME type to refer to protocol messages, therefore it does not requires a file extension.

Macintosh file type code(s): n/a

Person & email address to contact for further information: See Authors' Addresses section.

Intended usage: COMMON

Restrictions on usage: none

Author: See Authors' Addresses section.

Change controller: IESG (iesg@ietf.org)

11 Acknowledgments

The authors would like to thank many people for for their help and comments, particularly: Zhang Yunfei, Liao Hongluan, Roni Even, Dave Cottlehuber, Bhumip Khasnabish, Wu Yichuan, Peng Jin, Chi Jing, Zong Ning, Song Haibin, Chen Wei, Zhijia Chen, Christian Schmidt, Lars Eggert, David Harrington, Henning Schulzrinne, Kangheng Wu, Martin Stiemerling, Jianyin Zhang, Johan Pouwelse, Riccardo Petrocco and Arno Bakker.

Rui Cruz, Mario Nunes and Joao Taveira were partially supported by the SARACEN project [SARACEN], a research project of the European Union 7th Framework Programme (contract no. ICT-248474).

The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of the SARACEN project, the European Commission, Huawei or China Mobile.

12 References

12.1 Normative References

[RFC0793] Postel, J., "Transmission Control Protocol", STD 7,

RFC 793, September 1981, <http://www.rfc-editor.org/info/rfc793>.

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

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999, <http://www.rfc-editor.org/info/rfc2616>.

[RFC3629] Yergeau, F., "UTF-8, a transformation format of ISO 10646", STD 63, RFC 3629, November 2003, <http://www.rfc-editor.org/info/rfc3629>.

12.2 Informative References

[RFC7159] Bray, T., Ed., "The JavaScript Object Notation (JSON) Data Interchange Format", RFC 7159, March 2014, <http://www.rfc-editor.org/info/rfc7159>.

[RFC2564] Kalbfleisch, C., Krupczak, C., Presuhn, R., and J. Saperia, "Application Management MIB", RFC 2564, May 1999, <http://www.rfc-editor.org/info/rfc2564>.

[RFC2790] Waldbusser, S. and P. Grillo, "Host Resources MIB", RFC 2790, March 2000, <http://www.rfc-editor.org/info/rfc2790>.

[RFC2818] Rescorla, E., "HTTP Over TLS", RFC 2818, May 2000, <http://www.rfc-editor.org/info/rfc2818>.

[RFC2975] Aboba, B., Arkko, J., and D. Harrington, "Introduction to Accounting Management", RFC 2975, October 2000, <http://www.rfc-editor.org/info/rfc2975>.

[RFC3410] Case, J., Mundy, R., Partain, D., and B. Stewart, "Introduction and Applicability Statements for Internet-Standard Management Framework", RFC 3410, December 2002, <http://www.rfc-editor.org/info/rfc3410>.

[RFC3729] Waldbusser, S., "Application Performance Measurement MIB", RFC 3729, March 2004, <http://www.rfc-editor.org/info/rfc3729>.

[RFC3986] Berners-Lee, T., Fielding, R., and L. Masinter, "Uniform

Resource Identifier (URI): Generic Syntax", STD 66, RFC 3986, January 2005, <http://www.rfc-editor.org/info/rfc3986>.

[RFC4022] Raghunarayan, R., Ed., "Management Information Base for the Transmission Control Protocol (TCP)", RFC 4022, March 2005, <http://www.rfc-editor.org/info/rfc4022>.

[RFC4122] Leach, P., Mealling, M., and R. Salz, "A Universally Unique IDentifier (UUID) URN Namespace", RFC 4122, July 2005, <http://www.rfc-editor.org/info/rfc4122>.

[RFC4150] Dietz, R. and R. Cole, "Transport Performance Metrics MIB", RFC 4150, August 2005, <http://www.rfc-editor.org/info/rfc4150>.

[RFC5245] Rosenberg, J., "Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal for Offer/Answer Protocols", RFC 5245, April 2010, <http://www.rfc-editor.org/info/rfc5245>.

[RFC5246] Dierks, T. and E. Rescorla, "The Transport Layer Security (TLS) Protocol Version 1.2", RFC 5246, August 2008, <http://www.rfc-editor.org/info/rfc5246>.

[RFC5424] Gerhards, R., "The Syslog Protocol", RFC 5424, March 2009, <http://www.rfc-editor.org/info/rfc5424>.

[RFC5706] Harrington, D., "Guidelines for Considering Operations and Management of New Protocols and Protocol Extensions", RFC 5706, November 2009, <http://www.rfc-editor.org/info/rfc5706>.

[RFC5952] Kawamura, S. and M. Kawashima, "A Recommendation for IPv6 Address Text Representation", RFC 5952, August 2010, <http://www.rfc-editor.org/info/rfc5952>.

[RFC6241] Enns, R., Ed., Bjorklund, M., Ed., Schoenwaelder, J., Ed., and A. Bierman, Ed., "Network Configuration Protocol (NETCONF)", RFC 6241, June 2011, <http://www.rfc-editor.org/info/rfc6241>.

[RFC6709] Carpenter, B., Aboba, B., Ed., and S. Cheshire, "Design Considerations for Protocol Extensions", RFC 6709, September 2012, <http://www.rfc-editor.org/info/rfc6709>.

[RFC6749] Hardt, D., Ed., "The OAuth 2.0 Authorization Framework", RFC 6749, October 2012, <http://www.rfc-

editor.org/info/rfc6749>.

[RFC6750] Jones, M. and D. Hardt, "The OAuth 2.0 Authorization Framework: Bearer Token Usage", RFC 6750, October 2012, <http://www.rfc-editor.org/info/rfc6750>.

[RFC6972] Zhang, Y. and N. Zong, "Problem Statement and Requirements of the Peer-to-Peer Streaming Protocol (PPSP)", RFC 6972, July 2013, <http://www.rfc-editor.org/info/rfc6972>.

[RFC7285] Alimi, R., Ed., Penno, R., Ed., Yang, Y., Ed., Kiesel, S., Previdi, S., Roome, W., Shalunov, S., and R. Woundy, "Application-Layer Traffic Optimization (ALTO) Protocol", RFC 7285, September 2014, <http://www.rfc-editor.org/info/rfc7285>.

[Fielding] Fielding, R., "Architectural Styles and the Design of Network-based Software Architectures", University of California, Irvine, Dissertation 2000, 2000.

[SARACEN] "SARACEN Project Website", http://www.saracen-p2p.eu/.

[Contracts] Piatek, M., Venkataramani, A., Yang, R., Zhang, D. and A. Jaffe, "Contracts: Practical Contribution Incentives for P2P Live Streaming", in NSDI '10: USENIX Symposium on Networked Systems Design and Implementation, April 2010.

Authors' Addresses

Rui Santos Cruz
IST/INESC-ID/INOV
Phone: +351.939060939
Email: rui.cruz@ieee.org

Mario Serafim Nunes
IST/INESC-ID/INOV
Rua Alves Redol, n.9
1000-029 LISBOA, Portugal
Phone: +351.213100256
Email: mario.nunes@inov.pt

Rachel Huang
Huawei
Email: rachel.huang@huawei.com

Jinwei Xia
Huawei
Nanjing, Baixia District 210001, China
Phone: +86-025-86622310
Email: xiajinwei@huawei.com

Joao P. Taveira
IST/INOV
Email: joao.silva@inov.pt

Deng Lingli
China Mobile
Email: denglingli@chinamobile.com

Gu Yingjie
Email: guyingjie@gmail.com