Abstract

This document specifies the Peer-to-Peer Streaming Protocol--Tracker Protocol (PPSP-TP), an application-layer control (signaling) protocol for the exchange of meta information between trackers and peers, such as initial offer/request of participation in multimedia content streaming, content information, peer lists and reports of activity and status. 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 (audio, video, associated text/metadata) delivery, such as adaptive multi-rate, layered (scalable) and multi-view (3D), in live, time-shifted and on-demand modes.

Status of this Memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference.
material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at
http://www.ietf.org/1id-abstracts.html

The list of Internet-Draft Shadow Directories can be accessed at
http://www.ietf.org/shadow.html

Copyright and License Notice

Copyright (c) 2012 IETF Trust and the persons identified as the
document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal
Provisions Relating to IETF Documents
(http://trustee.ietf.org/license-info) in effect on the date of
publication of this document. Please review these documents
carefully, as they describe your rights and restrictions with respect
to this document. Code Components extracted from this document must
include Simplified BSD License text as described in Section 4.e of
the Trust Legal Provisions and are provided without warranty as
described in the Simplified BSD License.
C.2. PPSP Tracker Protocol Requirements . . . . . . . . . . . . . . . . 50
C.3. PPSP Security Considerations . . . . . . . . . . . . . . . . . . 51
Authors' Addresses . . . . . . . . . . . . . . . . . . . . . . . . . . . 51
1. Introduction

The Peer-to-Peer Streaming Protocol (PPSP) is composed of two protocols: the PPSP Tracker Protocol and the PPSP Peer Protocol. [I-D.ietf-ppsp-problem-statement] specifies that the Tracker Protocol should standardize format/encoding of information and messages between PPSP peers and PPSP trackers and [I-D.ietf-ppsp-reqs] 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. Content information metafiles (Media Presentation Descriptions) are also stored in the tracker system allowing active peers in the swarm to interpret content structure.

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

This document describes the PPSP Tracker protocol and how it satisfies the requirements for the IETF Peer-to-Peer Streaming Protocol (PPSP-TP), 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. Use Scenarios and Streaming Modes

This section is tutorial in nature and does not contain any normative statements.

This section describes some aspects of the use of PPSP-TP. The examples were chosen to illustrate the basic operation, but not to limit what PPSP-TP may be used for.

The functional entities related to PPSP protocols are the Client Media Player, the service Portal, the tracker and the peers. The complete description of these entities is not discussed here, as not in the scope the specification.

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.

The tracker is a logical entity that maintains the lists of PPSP active peers storing and exchanging a specific media content. The tracker also stores the status of active peers in swarms, to help in the selection of appropriate peers for a requesting peer. The tracker can be realized by geographically distributed tracker nodes or multiple server nodes in a data center, increasing the content availability, the service robustness and the network scalability or reliability. The management and locating of content index information are totally internal behaviors of the tracker system, which is invisible to the PPSP Peer [I-D.xiao-ppsp-reload-distributed-tracker].

The peer is also a logical entity embedding the P2P core engine, with a client serving side interface to respond to Client Media Player requests and a network side interface to exchange data and PPSP signaling with trackers and other peers.

The streaming technique is chunk-based, i.e., client peers obtain media chunks from serving peers and handle the buffering that is necessary for the playback processes during the download of the media chunks.

In Live streaming, all end users are interested in a specific media coming from an ongoing program, which means that all respective peers share nearly the same streaming content at a given point of time. Peers may store the live media for further distribution (known as time-shift TV), where the stored media is distributed in a VoD-like manner.

In VoD, different end users watch different parts of the recorded media content during a past event. In this case, each respective peer obtains from other peers the information on media chunks they store and then gets the required media from a selected set of those peers. While watching VoD, an end user can also switch to any place of the content, e.g., skip the credits part, or skip the part that it is not interested in. In this case the respective participating peer may not store all the content segments. From the whole swarm point of view, the participating peers typically store different parts of content.
1.2. Assumptions

This section is tutorial in nature and does not contain any normative statements.

The process used for media streaming distribution assumes a segment (chunk) transfer scheme whereby the original content (that can be encoded using adaptive or scalable techniques) is chopped into small segments (and subsegments) having the following representations:

1. Adaptive - alternate representations with different qualities and bitrates; a single representation is non-adaptive;
2. Scalable description levels - multiple additive descriptions (i.e., addition of descriptions refine the quality of the video);
3. Scalable layered levels - nested dependent layers corresponding to several hierarchical levels of quality, i.e., higher enhancement layers refine the quality of the video of lower layers.
4. Scalable multiple views - views correspond to mono (2D) and stereoscopic (3D) videos, with several hierarchical levels of quality.

These streaming distribution techniques support dynamic variations in video streaming quality while ensuring support for a plethora of end user devices and network connections.

1.2.1. Enrollment and Bootstrap

In order to join an existing P2P streaming service and to participate in content sharing, any peer must first locate a tracker, using for example, the following method (as illustrated in Figure 1):

```
+--------+      +--------+     +--------+    +---------+  +--------+
| Player |      | Peer 1 |     | Portal |    | Tracker |  | Peer 2 |
|        +--------+      +--------+     +--------+    +---------+  +--------+
|                |               |              |           |
|--Page request----------------->|              |           |
|<--------------Page with links--|              |           |
|--Select stream (MPD Request)--|              |           |
|<------------------------OK+MPD--|              |           |
|--MPD---------->|--CONNECT-------------------->|           |
|                |<-------------------------OK--|           |
|                |--JOIN----------------------->|           |
|<-----------OK--|<----------------OK+Peerlist--|           |
|                |                              |           |
|-- Get(Chunk) ->|<---------- (Peer protocol) ------------->|
|<---- Chunk ----|<-------------------------------- Chunk --|
```

Figure 1: A typical PPSP session
1. From a service provider provisioning mechanism: this is a typical case used on the provider Super-Seeders (edge caches and/or Media Servers).

2. From a web page: a Publishing and Searching Portal may provide tracker location information to end users.

3. From the MPD file of a content: this metainfo file must contain information about the address of one or more trackers (that can be grouped by tiers of priority) which are controlling the swarm for that media content.

In order to be able to bootstrap, a peer must first obtain a Peer-ID (identity associated with the end user authentication credentials) and any required security certificates or authorization tokens from an enrollment service (end user registration).

The specification of the mechanism used to obtain a Peer-ID, certificates or tokens is not in the scope of this document.

1.2.2. NAT Traversal

It is assumed that all trackers must be in the public Internet and have been placed there deliberately. This document will not describe NAT Traversal mechanisms but the protocol facilitates flexible NAT Traversal techniques, such as those based on ICE [RFC5245], considering that the tracker node may provide NAT traversal services, as a STUN-like tracker. Being a STUN-like tracker, it can discover the reflexive candidate addresses of a peer and make them available in responses to requesting peers, a mechanism named PPSP-ICE in [I-D.li-ppsp-nat-traversal-02].

1.2.3. Content Information Metadata

Multimedia contents may consist of several media components (for example, audio, video, and text), each of which might have different characteristics.

The representations of a media content correspond to encoded alternative of the same media component, varying from other representations by bitrate, resolution, number of channels, or other characteristics. Each representation consists of one or multiple segments. Segments are the media stream chunks in temporal sequence.

These characteristics may be described in a Media Presentation Description (MPD). Examples of MPD for on-demand and Live programs are illustrated in Appendix B. It is envisioned that the content information metadata used in PPSP may align with the MPD format of ISO/IEC 23009-1 [ISO.IEC.23009-1].
1.2.4. Authentication, Confidentiality, Integrity

Channel-oriented security should be used in the communication between peers and tracker, such as the Transport Layer Security (TLS) to provide privacy and data integrity. HTTP/1.1 over TLS (HTTPS) [RFC2818] is the preferred approach for preventing disclosure of peer critical information via the communication channel.

Due to the transactional nature of the communication between peers and tracker a method for adding authentication and data security services via replaceable mechanisms should be employed. One such method is the OAuth 2.0 Authorization [I-D.ietf-oauth-v2] with bearer token, providing the peer with the information required to successfully utilize the access token to make protected requests to the tracker [I-D.ietf-oauth-v2-bearer].

2. 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 [RFC2119].

This draft uses terms defined in [I-D.ietf-ppsp-problem-statement] and in [I-D.xiao-ppsp-reload-distributed-tracker].

Absolute Time: Absolute time is expressed as ISO 8601 [ISO.8601.2004] timestamps, using zero UTC offset (GMT). 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.

Adaptive Streaming: Multiple alternate representations (different qualities and bitrates) of the same media content co-exist for the same streaming session; each alternate representation corresponds to a different media quality level; peers can choose among the alternate representations for decode and playback.

Base Layer: The playable representation level in Scalable Video Coding (SVC) required by all upper level Enhancements Layers for proper decoding of the video.

Chunk: A chunk is a generic term used whenever no ambiguity is raised, to refer to a segment or a subsegment of partitioned streaming media.

Complementary Representation: Representation in a set of representations which have inter-representation dependencies and which when combined result in a single representation for decoding.
and presentation.

Connection Tracker: The tracker node to which the PPSP peer will connect when it wants to get registered and join the PPSP system.

Continuous media: Media with an inherent notion of time, for example, speech, audio, video, timed text or timed metadata.

Enhancement Layer: Enhancement differential quality level (complementary representation) in Scalable Video Coding (SVC) used to produce a higher quality, higher definition video in terms of space (i.e., image resolution), time (i.e., frame rate) or Signal-to-Noise Ratio (SNR) when combined with the playable Base Layer [ITU-T.H.264].

Join Time: Join time is the absolute time when a peer registers on a tracker. This value is recorded by the tracker and is used to calculate Online Time.

Live streaming: The scenario where all clients receive streaming content for the same ongoing event. The lags between the play points of the clients and that of the streaming source are small.

Media Component: An encoded version of one individual media type such as audio, video or timed text with specific attributes, e.g., bandwidth, language, or resolution.

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 segments (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 joins. This value indicates the stability of a peer, and it is calculated by tracker when necessary.

Peer: A peer refers to a participant in a P2P streaming system that not only receives streaming content, but also stores and uploads streaming content to other participants.

Peer-ID: Unique identifier for the peer. The Peer-ID and any required security certificates are obtained from an offline enrollment server.

Peer-Peer Messages (i.e., Peer Protocol): The Peer Protocol messages
enable each peer to exchange content availability with other peers and request other peers for content.

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

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.

Scalable Streaming: With Multiple Description Coding (MDC), multiple additive descriptions (that can be independently played-out) to refine the quality of the video when combined together. With Scalable Video Coding (SVC), nested dependent enhancement layers (hierarchical levels of quality), refine the quality of lower layers, from the lowest level (the playable Base Layer). With Multiple View Coding (MVC), multiple views allow the video to be played in 3D when the views are combined together.

Segment: A segment is a resource that can be identified, by an ID or an HTTP-URL and possibly a byte-range, and is included in the MPD. The segment is a basic unit of partitioned streaming media, which is used by a peer for the purpose of storage, advertisement and exchange among peers.

Subsegment: Smallest unit within segments which may be indexed at the segment level.

Swarm: A swarm refers to a group of peers sharing the same content (e.g., video/audio program, digital file, etc.) at a given time.

Swarm-ID: Unique identifier for a swarm. It is used to describe a specific resource shared among peers.

Tracker: A tracker refers to a centralized logical directory service used to communicate with PPSP Peers for content registration and location, which maintains the lists of PPSP peers storing segments for a specific live content channel or streaming media and answers queries from PPSP peers.

Tracker-Peer Messages (i.e., Tracker Protocol): The Tracker Protocol messages provide communication between peers and trackers, by which
peers provide content availability, report streaming status and request peer lists from trackers.

Video-on-demand (VoD): A kind of application that allows users to select and watch video content on demand.

3. Architectural and Functional View

The PPSP Tracker Protocol architecture uses a two-layer approach i.e., a PPSP-TP messaging layer and a PPSP-TP request/response layer.

The PPSP-TP messaging layer deals with the underlying transport protocol and the asynchronous nature of the interactions between tracker and peers.

The PPSP-TP request/response layer deals with the interactions between tracker and peers using Method and Response codes (see Figure 2).

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 when using a connection-oriented transport like TCP, or TLS [RFC5246] over it.

```
+----------------------+
|      Application     |
+----------------------+
<table>
<thead>
<tr>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Request/Response</td>
</tr>
</tbody>
</table>
|----------------------|  PPSP-TP
| Messages             |
+----------------------+
| TRANSPORT            |
+----------------------+
```

Figure 2: Abstract layering of PPSP-TP

The functional entities involved in the PPSP Tracker Protocol are trackers and peers (which may support different capabilities).

Peers correspond to devices that actually participate in sharing a media content and are organized in (various) swarms corresponding each swarm to the group of peers streaming that content at any given time. Each peer stores chunks of the media content, called segments (or subsegments), and contacts the tracker to advertise which information it has available. When a peer wishes to obtain information about the swarm, it contacts the tracker to find other
peers participating in that specific swarm.

The tracker is a logical entity that maintains the lists of peers storing chunks for a specific Live media channel or 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 VoD or Live streaming) with peers, but information about which peers can provide which pieces of content.

When a peer wants to receive streaming of a selected content:

1. Peer connects to a local connection tracker and joins a swarm.
2. Peer acquires a list of peers from the connection tracker.
3. Peer exchanges its content availability with the peers on the obtained peer list.
4.Peer identifies the peers with desired content.
5. Peer requests for the content from the identified peers.

When a peer wants to share streaming of certain content with other peers:

1. Peer connects to the connection tracker.
2. Peer sends information to the connection tracker about the swarm it belongs to (joins), plus streaming status and/or content availability.

A P2P streaming process is summarized in Figure 3.

---

---

Figure 3: A PPSP streaming process
4. Messaging Model

The messaging model of PPSP-TP is based on the exchange of messages that follow the syntax and semantics of the current HTTP/1.1 specification [RFC2616]. The exchange of messages is envisioned to be performed over a stream-oriented reliable transport protocol, like TCP.

PPSP-TP is a text-based protocol, uses the UTF-8 character set [RFC3629] and the protocol messages are either requests from client peers to a tracker server, or responses from a tracker server to client peers.

5. Request/Response model

PPSP-TP request and response semantics are carried as entities (header and body) in PPSP-TP messages which correspond to either HTTP/1.1 request methods or HTTP/1.1 response codes, respectively.

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/1.1 response codes, however, not all HTTP/1.1 response codes are used for the PPSP-TP (section 7).

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

<table>
<thead>
<tr>
<th>PPSP Tracker Req. Messages</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECT</td>
</tr>
<tr>
<td>DISCONNECT</td>
</tr>
<tr>
<td>JOIN</td>
</tr>
<tr>
<td>FIND</td>
</tr>
<tr>
<td>STAT_REPORT</td>
</tr>
</tbody>
</table>

Table 1: Request Messages

CONNECT: This request message is used when a peer registers in the tracker. The tracker records the Peer-ID, connect-time (referenced to the absolute time), peer IP addresses and link status.

DISCONNECT: This request message is used when the peer intends to no longer participate in a specific swarm, or in all swarms. The
tracker deletes the corresponding activity records related to the peer (including its status and all content status for the corresponding swarms).

JOIN: This request message is used for a peer to notify the tracker that it wishes to participate in a swarm. The tracker records the content availability, i.e., adds the peer to the peers list for the swarm. On receiving a JOIN message, the tracker first checks the PeerMode type and then decides the next step (more details are referred in section 8.3).

FIND: This request message allows a peer to request to the tracker the peer list for a specific content representation or specific chunks of a media component in a swarm, before it can request the content from the peers. 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 allows the exchange of statistic and status data between an active peer and a tracker to improve system performance. This request message is sent periodically to the tracker.

6. The Tracker State Machine

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.

```
+-------+             +---------+            +------------+
| INIT |------------>| STARTED |----------->| TERMINATED |
+-------+             +---------+            +------------+
```

Figure 4: Tracker State Machine

When there are no peers registered in the tracker, the state machine is in the INIT state. When the first peer is registered 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.

In addition to this state machine, each registered Peer-ID is modeled with its own transaction state machine (Figure 5), instantiated per Peer-ID registered in the tracker, and deleted when it is removed. Unlike the state machine for the Peer-ID registration, which exists even when no Peer-IDs are registered, the per-Peer-ID transaction state machine is instantiated when the Peer-ID is registered, and deleted when the Peer-ID is 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).

![Per-Peer-ID Transaction State Machine Diagram]

When a new Peer-ID is added, the per-Peer-ID state machine for it 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 life time of streaming activity of a peer, the 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:

- Reception of CONNECT, JOIN, FIND, DISCONNECT and STAT_REPORT messages, or
- 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.

6.1. Normal Operation

On normal operation the process consists of the following steps:

1) When a CONNECT message is received from a peer, if successfully authenticated and validated, the tracker registers the Peer-ID and associated information (IP addresses), sends the response of successful registration to peer and starts the "init timer" waiting for a new message from the peer.

2) While PEER REGISTERED, when a JOIN message is received with valid swarm information, the tracker stops the "init timer", starts the "track timer" and sends the response of successful join to the peer. The response MAY contain the appropriate list of peers in the swarm, depending on PeerMode (section 8.3). A successful first JOIN starts the TRACKING state associated with the peer-ID for the requested swarm.

3) While TRACKING, a JOIN or FIND message received with valid swarm information from the peer resets the "track timer" and is responded with a successful condition, either for the JOIN to (an additional) swarm or for including the appropriate list of peers for the scope in the FIND request.

4) While TRACKING, a DISCONNECT(x) message received from the peer, containing a valid x=Swarm-ID resets the "track timer" and is responded with a successful condition. The tracker cleans the information associated with the participation of the Peer-ID in
the specified swarm(s).

In TRACKING state a STAT_REPORT message received from the peer resets the "track timer" and is responded with a successful condition. The STAT_REPORT message MAY contain information related with Swarm-IDs to which the peer is joined.

5) From either PEER REGISTERED or TRACKING states a DISCONNECT(x) message received from the peer, where x=nil, the tracker stops the "track timer", cleans the information associated with the participation of the Peer-ID in the the swarm(s) joined, responds with a successful condition, deletes the registration of the Peer-ID and transitions to TERMINATED state for that Peer-ID.

6) From TRACKING state a DISCONNECT(x) message received from the peer, where x=ALL or x=Swarm-ID is the last swarm, the tracker stops the "track timer", cleans the information associated with the participation of the Peer-ID in the the swarm(s) joined, responds with a successful condition and transitions to PEER REGISTERED state.

6.2. 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 the PEER REGISTERED state (while the "init timer" has not expired) receiving FIND, CONNECT or STAT_REPORT messages from the peer is considered an error condition. The tracker responds with error code 403 Forbidden (described in section 7), and resets the "init timer" one last time.

B) At the TRACKING state (while the "track timer" has not expired) receiving a CONNECT message from the peer is considered an error condition. The tracker responds with error code 403 Forbidden (described in section 7), and resets the "track timer".

NOTE: This situation may correspond to a malfunction at the peer or to malicious conditions. A preventive measure would be to reset the "track timer" one last time and if no valid message is received proceed to TERMINATE state for the Peer-ID by de-registering the peer and cleaning all peer information.

C) Without receiving messages from the peer, either from PEER
REGISTERED state (init timer) or TRACKING state (track timer), on timeout the tracker cleans all the information associated with the Peer-ID in all swarms it was joined, deletes the registration, and transitions to TERMINATE state for that Peer-ID. The same action is taken if no valid message is received at the PEER REGISTERED state after the last "init timer" expires.

7. Protocol Specification

7.1. Messages Syntax

PPSP-TP messages use the generic message format of RFC 5322 [RFC5322] for transferring the payload of the message (Requests and Responses).

PPSP-TP messages consist of a start-line, one or more header fields, an empty line indicating the end of the header fields, and, when applicable, a message-body.

The start-line, each message-header line, and the empty line MUST be terminated by a carriage-return line-feed sequence (CRLF). Note that the empty line MUST be present even if the message-body is not.

The PPSP-TP message and header field syntax is identical to HTTP/1.1 [RFC2616].

A Request message is a standard HTTP/1.1 message starting with a Request-Line generated by the HTTP client peer. The Request-Line contains a method name, a Request-URI, and the protocol version separated by a single space (SP) character.

Request-Line =
  Method SP Request-URI SP HTTP-Version CRLF

A Request message example is the following:

<Method> /<Resource> HTTP/1.1
Host: <Host>
Content-Lenght: <ContentLenght>
Content-Type: <ContentType>
Authorization: <AuthToken>

[Request_Body]

The HTTP Method token and Request-URI (the Resource) identifies the resource upon which to apply the operation requested.
The Response message is also a standard HTTP/1.1 message starting with a Status-Line generated by the tracker. The Status-Line consists of the protocol version followed by a numeric Status-Code and its associated Reason-Phrase, with each element separated by a single SP character.

Status-Line = HTTP-Version SP Status-Code SP Reason-Phrase CRLF

A Response message example is the following:

HTTP/1.1 <Status-Code> <Reason-Phrase>
Content-Length: <ContentLength>
Content-Type: <ContentType>
Content-Encoding: <ContentCoding>

[Response_Body]

The Status-Code element is a 3-digit integer result code that indicates the outcome of an attempt to understand and satisfy a request.

The Reason-Phrase element is intended to give a short textual description of the Status-Code.

7.1.1. Header Fields

The header fields are identical to HTTP/1.1 header fields in both syntax and semantics.

Some header fields only make sense in requests or responses. If a header field appears in a message not matching its category (such as a request header field in a response), it MUST be ignored.

The Host request-header field in the request message follows the standard rules for the HTTP/1.1 Host header.

The Content-Type entity-header field MUST be used in requests and responses containing message-bodies to define the Internet media type of the message-body.

The Content-Encoding entity-header field MAY be used in response messages with "gzip" compression scheme [RFC2616] for faster transmission times and less network bandwidth usage.

The Content-Length entity-header field MUST be used in messages containing message-bodies to locate the end of each message in a stream.
The Authorization header field in the request message allows a peer to authenticate itself with a tracker, containing authentication information.

7.1.2. Methods

PPSP-TP uses HTTP/1.1 POST method token for all request messages.

7.1.3. Message Bodies

PPSP-TP requests MUST contain message-bodies.

PPSP-TP responses MAY include a message-body.

If the message-body has undergone any encoding such as compression, then this MUST be indicated by the Content-Encoding header field; otherwise, Content-Encoding MUST be omitted.

If applicable, the character set of the message body is indicated as part of the Content-Type header-field, and the default value for PPSP-TP messages is "UTF-8".

7.1.4. Message Response Codes

The response codes in PPSP-TP response messages are consistent with HTTP/1.1 response status-codes. However, not all HTTP/1.1 response status-codes are appropriate for PPSP-TP, and only those that are appropriate are given here. Other HTTP/1.1 response codes SHOULD NOT be used in PPSP-TP.

The class of the response is defined by the first digit of the Status-Code. The last two digits do not have any categorization role. For this reason, any response with a Status-Code between 200 and 299 is referred to as a "2xx response", and similarly to the other supported classes:

2xx: Success -- the action was successfully received, understood, and accepted;
4xx: Peer Error -- the request contains bad syntax or cannot be fulfilled at this tracker;
5xx: Tracker Error -- the tracker failed to fulfill an apparently valid request;

The valid response codes are the following (Status-Code Reason-Phrase):

200 OK -- The request has succeeded. The information returned with the response describes or contains the result of the action;
400 Bad Request -- The request could not be understood due to malformed syntax.

401 Unauthorized -- The request requires authentication.

403 Forbidden -- The tracker understood the request, but is refusing to fulfill it. The request SHOULD NOT be repeated.

404 Not Found -- This status is returned if the tracker did not find anything matching the Request-URI.

408 Request Timeout -- The peer did not produce a request within the time that the tracker was prepared to wait.

411 Length Required -- The tracker refuses to accept the request without a defined Content-Length. The peer MAY repeat the request if it adds a valid Content-Length header field containing the length of the message-body in the request message.

414 Request-URI Too Long -- The tracker is refusing to service the request because the Request-URI is longer than the tracker is willing to interpret. This rare condition is likely to occur when the tracker is under attack by a client attempting to exploit security holes.

500 Internal Server Error -- The tracker encountered an unexpected condition which prevented it from fulfilling the request.

503 Service Unavailable -- The tracker is currently unable to handle the request due to a temporary overloading or maintenance condition.

7.2. Request/Response Syntax and Format

The message-body for Requests and Responses requiring it, is encoded in XML.

The XML message-body MUST begin with an XML declaration line specifying the version of XML being used and indicating the character encoding, that SHOULD be "UTF-8". The root element MUST be PPSPTackerProtocol.
The generic format of a Request is the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PPSPTrackerProtocol version="1.0">
  <Request>
    <TransactionID/>
    <PeerID/>
    <SwarmID/>
    <PeerNum/>
    <PeerMode/>
    <PeerGroup/>
    <ContentGroup/>
    <StatisticsGroup/>
  </Request>
</PPSPTrackerProtocol>
```

The generic format of a Response is the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PPSPTrackerProtocol version="1.0">
  <Response>
    <TransactionID/>
    <SwarmID/>
    <PeerGroup/>
  </Response>
</PPSPTrackerProtocol>
```

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

The Response element MUST be present in responses and corresponds to the response method type of the message.

The element TransactionID 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 version of PPSP-TP being used is indicated by the attribute @version of the root element.

All Request messages MUST contain a PeerID element to uniquely identify the peer (Peer-ID) in the network.

The PeerID information may be present on the following levels:
- On PPSPTrackerProtocol level in PPSPTrackerProtocol.PeerID element. For details refer to 7.2.1 Table 2.
- On PeerGroup level in PeerGroup.PeerInfo.PeerID element. For details refer to 7.2.1 Table 3.
The SwarmID element MUST be present in JOIN, FIND and DISCONNECT requests. The SwarmID element MUST be present in JOIN and FIND responses. Details of usage in 8.2, 8.3 and 8.4.

The SwarmID information may be present on the following levels:
- On PPSPTackerProtocol level in PPSPTackerProtocol.SwarmID element. For details refer to 7.2.1 Table 2.

The PeerMode element MUST be present in JOIN requests. Details of usage in 8.3.

The PeerMode information may be present on the following levels:
- On PPSPTackerProtocol level in PPSPTackerProtocol.PeerMode element. For details refer to 7.2.1 Table 2.
- On PeerGroup level in PeerGroup.PeerMode element. For details refer to 7.2.1 Table 5.

The PeerNum element MUST be present in JOIN requests and MAY contain the attribute @abilityNAT to inform the tracker on the preferred type of peers, in what concerns their NAT traversal situation, to be returned in a peer list. Details of usage in 8.2, 8.3 and 8.4.

The PeerGroup element MUST be present in CONNECT requests and responses and MAY be present in responses to JOIN and FIND requests if the corresponding response returns information about peers. Details of usage in 8.1, 8.3 and 8.4.

The ContentGroup element MAY be present in requests referencing content, i.e., JOIN and FIND, if the request includes a content scope. Details of usage in 8.3 and 8.4.

The StatisticsGroup element MAY be present in STAT_REPORT requests. Details of usage in 8.5.

The semantics of the attributes and elements within a PPSPTackerProtocol root element is described in subsection 7.2.1.

Request and Response processing is provided in section 8 for each message.

The XML-syntax of the of PPSP-TP XML elements for Requests and Responses is provided in the XML-Schema of Appendix A.
7.2.1. Semantics of PPSPTrackerProtocol elements

The semantics of PPSPTrackerProtocol elements and attributes are described in the following tables.

<table>
<thead>
<tr>
<th>Element Name or Attribute Name</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>PPSPTrackerProtocol</td>
<td>1</td>
<td>The root element.</td>
</tr>
<tr>
<td>@version</td>
<td>M</td>
<td>Provides the version of PPSP-TP.</td>
</tr>
<tr>
<td>Request</td>
<td>0...1</td>
<td>Provides the request method and MUST be present in Request.</td>
</tr>
<tr>
<td>Response</td>
<td>0...1</td>
<td>Provides the response method and MUST be present in Response.</td>
</tr>
<tr>
<td>PeerID</td>
<td>0...1</td>
<td>Peer Identification.</td>
</tr>
<tr>
<td>SwarmID</td>
<td>0...1</td>
<td>Swarm Identification.</td>
</tr>
<tr>
<td>PeerMode</td>
<td>0...1</td>
<td>Mode of Peer participation in a swarm, which can be &quot;LEECH&quot; or &quot;SEED&quot;.</td>
</tr>
<tr>
<td>PeerNUM</td>
<td>0...1</td>
<td>Maximum peers to be received in with capabilities indicated.</td>
</tr>
<tr>
<td>@abilityNAT</td>
<td>CM</td>
<td>Type of NAT traversal peers, as &quot;NoNAT&quot;, &quot;STUN&quot;, &quot;TURN&quot; or &quot;PROXY&quot;</td>
</tr>
<tr>
<td>@concurrentLinks</td>
<td>CM</td>
<td>Concurrent connectivity level of peers, &quot;HIGH&quot;, &quot;LOW&quot; or &quot;NORMAL&quot;</td>
</tr>
<tr>
<td>@onlineTime</td>
<td>CM</td>
<td>Availability or online duration of peers, &quot;HIGH&quot; or &quot;NORMAL&quot;</td>
</tr>
<tr>
<td>@uploadBWlevel</td>
<td>CM</td>
<td>Upload bandwidth capability of peers, &quot;HIGH&quot; or &quot;NORMAL&quot;</td>
</tr>
<tr>
<td>PeerGroup</td>
<td>0...1</td>
<td>Provides information on peers.</td>
</tr>
<tr>
<td>ContentGroup</td>
<td>0...1</td>
<td>Provides information on content.</td>
</tr>
<tr>
<td>StatisticsGroup</td>
<td>0...1</td>
<td>Provides statistic data of peer and content. Details in Table 5</td>
</tr>
</tbody>
</table>

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 PPSPTrackerProtocol.
### Table 3: Semantics of PeerGroup.

If STUN-like functions are enabled in the tracker and a PPSP-ICE method is used, as described in [I-D.li-ppsp-nat-traversal-02], the attributes `@type` and `@priority` MUST be returned with the transport address candidates in responses to CONNECT, JOIN or FIND 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.
<table>
<thead>
<tr>
<th>Element Name or Attribute Name</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>ContentGroup</td>
<td>0...1</td>
<td>Provides information on content.</td>
</tr>
<tr>
<td>Representation</td>
<td>1...N</td>
<td>Describes a component of content.</td>
</tr>
<tr>
<td>@id</td>
<td>M</td>
<td>Unique identifier for this Representation.</td>
</tr>
<tr>
<td>SegmentInfo</td>
<td>1</td>
<td>Provides segment information.</td>
</tr>
<tr>
<td>@startIndex</td>
<td>M</td>
<td>The index of the first media segment in the request scope for this Representation.</td>
</tr>
<tr>
<td>@endIndex</td>
<td>OP</td>
<td>The index of the last media segment in the request scope for this Representation.</td>
</tr>
</tbody>
</table>

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 ContentGroup.

The Representation element describes a component of a content identified by its attribute @id in the MPD. This element MAY be present for each component desired in the scope of the JOIN or FIND request. The scope of each Representation is indicated in the SegmentInfo element by the attribute @startIndex and, optionally, @endIndex.

The peer may use this information in JOIN or FIND requests, for example, to join a swarm starting from a specific point (as is the case of a live program, by specifying the adequate @startIndex) and/or find adequate peers in the swarm for that content scope.

An example of on-demand usage is the case of an end-user that previously watched a content with a certain audio language, then interrupted for a while (having disconnected) and later continued by re-joining from that point onwards but selecting a different available audio language. In this case the JOIN request would specify the required Representations and the @startIndex for each, i.e., all the adequate video components and the selected audio component. An example is illustrated in subsection 8.3.
<table>
<thead>
<tr>
<th>Element Name or Attribute Name</th>
<th>Use</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StatisticsGroup</td>
<td>0...1</td>
<td>Provides statistic data on peer and content.</td>
</tr>
<tr>
<td>Stat</td>
<td>1...N</td>
<td>Groups statistics property data.</td>
</tr>
<tr>
<td>@property</td>
<td>M</td>
<td>The property to be reported.</td>
</tr>
<tr>
<td>SwarmID</td>
<td>0...1</td>
<td>Swarm Identification.</td>
</tr>
<tr>
<td>UploadedBytes</td>
<td>0...1</td>
<td>Bytes sent to swarm.</td>
</tr>
<tr>
<td>DownloadedBytes</td>
<td>0...1</td>
<td>Bytes received from swarm.</td>
</tr>
<tr>
<td>AvailBandwidth</td>
<td>0...1</td>
<td>Upstream Bandwidth available.</td>
</tr>
<tr>
<td>Representation</td>
<td>0...N</td>
<td>Describes a component of content.</td>
</tr>
<tr>
<td>@id</td>
<td>CM</td>
<td>Unique identifier for this Representation.</td>
</tr>
<tr>
<td>SegmentInfo</td>
<td>1...N</td>
<td>Provides segment information by segment range. The chunkmap can be encoded in Base64 [RFC4648].</td>
</tr>
<tr>
<td>@startIndex</td>
<td>CM</td>
<td>The index of the first media segment in the chunkmap report for this Representation.</td>
</tr>
<tr>
<td>@endIndex</td>
<td>CM</td>
<td>The index of the last media segment in the chunkmap report for this Representation.</td>
</tr>
<tr>
<td>@chunkmapSize</td>
<td>CM</td>
<td>Size of chunkmap reported.</td>
</tr>
</tbody>
</table>

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 StatisticsGroup.

The Stat element is used to describe several properties relevant to the P2P network. These properties can be related with stream statistics, peer status information and content data information, like chunkmaps. 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.

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 elements and new attributes.

<table>
<thead>
<tr>
<th>@property</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>StreamStatistics</td>
<td>Stream statistic values per SwarmID</td>
</tr>
<tr>
<td>ContentMap</td>
<td>Reports map of chunks the peer has per Representation of the content</td>
</tr>
</tbody>
</table>

Table 6: StatisticsGroup default Stat @property values.

An example of a STAT_REPORT for multiple properties is illustrated in subsection 8.5.

7.2.2. Request element in request Messages

Table 7 defines the valid string representations for the requests. These values MUST be treated as case-sensitive.

<table>
<thead>
<tr>
<th>XML Request Methods</th>
<th>String Values</th>
</tr>
</thead>
<tbody>
<tr>
<td>CONNECT</td>
<td></td>
</tr>
<tr>
<td>DISCONNECT</td>
<td></td>
</tr>
<tr>
<td>JOIN</td>
<td></td>
</tr>
<tr>
<td>FIND</td>
<td></td>
</tr>
<tr>
<td>STAT_REPORT</td>
<td></td>
</tr>
</tbody>
</table>

Table 7: Valid Strings for Request element of requests.

7.2.3. Response element in response Messages

Table 8 defines the valid string representations for Response messages that require message-body. These values MUST be treated as case-sensitive.

Response messages not requiring message-body only use the standard HTTP/1.1 Status-Code and Reason-Phrase (appended, if appropriate, with detail phrase, as described in section 8.6).
<table>
<thead>
<tr>
<th>XML Response Method</th>
<th>HTTP Status-Code and Reason-Phrase</th>
</tr>
</thead>
<tbody>
<tr>
<td>SUCCESSFUL</td>
<td>200 OK</td>
</tr>
<tr>
<td>AUTHENTICATION REQUIRED</td>
<td>401 Unauthorized</td>
</tr>
</tbody>
</table>

Table 8: 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.

CONNECT: returns information about the successful registration of the peer.

DISCONNECT and STAT_REPORT: confirms the success of the requested operation.

JOIN and FIND: MAY return the list of peers meeting the desired criteria.

AUTHENTICATION REQUIRED: Authentication is required for the peer to make the request.

8. Request/Response Processing

When a PPSP-TP message is received some basic processing is performed, regardless of the message type.

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. The receiver must also verify that the XML body is properly formed. In case of error due to malformed messages appropriate responses MUST be returned, as described in 8.6.

8.1. CONNECT Request

This method is used when a peer registers to the system. The tracker records the Peer-ID, connect-time, IP addresses and link status.

The peer MUST properly form the XML message-body, set the Request method to CONNECT, generate and set the TransactionID, and set the PeerID with the identifier of the peer. The peer SHOULD also include
the IP addresses of its network interfaces in the CONNECT message.

An example of the message-body of a CONNECT Request is the following:

```
<?xml version="1.0" encoding="UTF-8"?>
<PPSPTrackerProtocol version="1.0">
  <Request>CONNECT</Request>
  <PeerID>656164657221</PeerID>
  <TransactionID>12345</TransactionID>
  <PeerGroup>
    <PeerInfo>
      <PeerAddress addrType="ipv4" ip="192.0.2.1" port="80"
                    priority="1" />
      <PeerAddress addrType="ipv6" ip="2001:db8::1" port="80"
                    priority="2"
                    type="HOST"
                    connection="3G" />
    </PeerInfo>
    <PeerGroup>
      <PeerInfo>
        <!-- Additional Peer Addresses here -->
      </PeerInfo>
    </PeerGroup>
  </PeerGroup>
</PPSPTrackerProtocol>
```

When receiving a well-formed CONNECT Request message, the tracker
will first processes 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, and then proceed
to register the peer in the service. In case of success a Response
message with a corresponding response value of SUCCESSFULL will be
generated.

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

If STUN-like function is enabled in the tracker, the response MAY
include the peer reflexive address [I-D.li-ppsp-nat-traversal-02].

The response MUST have the same TransactionID value as the request.
An example of a Response message for the CONNECT Request is the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PPSPTrackerProtocol version="1.0">
  <Response>SUCCESSFUL</Response>
  <TransactionID>12345</TransactionID>
  <PeerGroup>
    <PeerInfo>
      <PeerAddress addrType="ipv4" ip="198.51.100.1" port="80"
        priority="1"
        type="REFLEXIVE"
        connection="ADSL"
        asn="64496"/>
    </PeerInfo>
  </PeerGroup>
</PPSPTrackerProtocol>
```

The Response MUST include a PeerGroup with PeerInfo data that includes the 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 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.

8.2. DISCONNECT Request

This method is used when the peer intends to leave a specific swarm, or the system, and no longer participate.

The tracker SHOULD delete the corresponding activity records related with the peer in the corresponding swarms (including its status and all content status).

The peer MUST properly form the XML message-body, set the Request method to DISCONNECT, set the PeerID with the identifier of the peer, randomly generate and set the TransactionID and include the SwarmID information.

The SwarmID value MUST be either a specific Swarm-ID the peer had previously joined, the value "ALL" to designate all joined swarms, or the value "nil" to completely disconnect from the system.
An example of the message-body of a DISCONNECT Request is the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PPSPTrackerProtocol version="1.0">
  <Request>DISCONNECT</Request>
  <PeerID>656164657221</PeerID>
  <SwarmID>ALL</SwarmID>
  <TransactionID>12345</TransactionID>
</PPSPTrackerProtocol>
```

In case of success a Response message with a corresponding response value of SUCCESSFULL will be generated. The response MUST have the same TransactionID value as the request.

Upon receiving a DISCONNECT message, the tracker cleans the information associated with the participation of the Peer-ID in the specified swarm (or in all swarms).

An example of a Response message for the DISCONNECT Request is the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PPSPTrackerProtocol version="1.0">
  <Response>SUCCESSFUL</Response>
  <TransactionID>12345</TransactionID>
</PPSPTrackerProtocol>
```

If the scope of SwarmID in the DISCONNECT request is "nil" the tracker will also delete the registration of the Peer-ID.

8.3. JOIN Request

This method is used for peers to notify the tracker that they wish to participate in a particular swarm.

The JOIN message is used when the peer has none or just some chunks (LEECH), or has all the chunks (SEED) of a content. The JOIN is used for both on-demand or Live streaming modes.

The peer MUST properly form the XML message-body, set the Request method to JOIN, set the PeerID with the identifier of the peer, set the SwarmID with the identifier of the swarm it is interested in, and randomly generate and set the TransactionID.
An example of the message-body of a JOIN Request is the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PPSPTrackerProtocol version="1.0">
  <Request>JOIN</Request>
  <PeerID>656164657221</PeerID>
  <SwarmID>1111</SwarmID>
  <TransactionID>12345</TransactionID>
  <PeerNum abilityNAT="STUN" concurrentLinks="HIGH" onlineTime="NORMAL" uploadBWlevel="NORMAL">5</PeerNum>
  <PeerMode>LEECH</PeerMode>
  <ContentGroup>
    <Representation id="tag0">
      <SegmentInfo startIndex="20" />
    </Representation>
    <Representation id="tag6">
      <SegmentInfo startIndex="20" />
    </Representation>
  </ContentGroup>
</PPSPTrackerProtocol>
```

The JOIN request MAY include a PeerNum element to indicate 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.

The PeerMode element SHOULD be set to the type of participation of the peer in the swarm (SEED or LEECH).

In the case of a JOIN to a specific point in a stream the request SHOULD include a ContentGroup to specify the joining point in terms of content Representations. The above example of a JOIN request would be for the case of an end-user that previously watched a content with a certain audio language, then interrupted for a while (having disconnected) and later continued by re-joining from that point onwards but selecting a different available audio language (Representation with @id="tag6" in the MPD of Appendix B).

When receiving a well-formed JOIN Request the tracker processes the information to check if it is valid and if the peer can join the swarm of interest. In case of success a response message with a Response value of SUCCESSFULL will be generated and the tracker enters the peer information into the corresponding swarm activity.
In case the PeerMode is SEED, the tracker just responds with a SUCCESSFUL response and enters the peer information into the corresponding swarm activity.

In case the PeerMode is LEECH the tracker will search and select an appropriate list of peers satisfying the conditions requested. The peer list 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 [I.D.ietf-alto-protocol].

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

An example of a Response message for the JOIN Request is:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PPSPTrackerProtocol version="1.0">
  <Response>SUCCESSFUL</Response>
  <TransactionID>12345</TransactionID>
  <PeerGroup>
    <PeerInfo>
      <PeerID>956264622298</PeerID>
      <PeerAddress addrType="ipv4" ip="198.51.100.22" port="80"
                  asn="64496" />
    </PeerInfo>
    <PeerInfo>
      <PeerID>3332001256741</PeerID>
      <PeerAddress addrType="ipv4" ip="198.51.100.201" port="80"
                  asn="64496" />
    </PeerInfo>
  </PeerGroup>
</PPSPTrackerProtocol>
```

The Response MUST include a PeerGroup with PeerInfo data that includes the public IP address of the selected active peers in the swarm.

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.

8.4. FIND Request

This method allows peers to request to the tracker, whenever needed and after being joined to a swarm, a new peer list for the swarm or
for specific scope of chunks of a media content Representation of that swarm.

The peer MUST properly form the XML message-body, set the Request method to FIND, set the PeerID with the identifier of the peer, set the SwarmID with the identifier of the swarm the peer is interested, and optionally, in order to find peers having the specific chunks, include information about the content.

The peer MUST also generate and set the TransactionID for the request.

An example of the message-body of a FIND Request is the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PPSPTrackerProtocol version="1.0">
  <Request>FIND</Request>
  <PeerID>656164657221</PeerID>
  <SwarmID>1111</SwarmID>
  <TransactionID>12345</TransactionID>
  <PeerNum abilityNAT="STUN"
    concurrentLinks="HIGH"
    onlineTime="NORMAL"
    uploadBWlevel="NORMAL">5</PeerNum>
  <ContentGroup>
    <Representation id="tag4" startIndex="110" endIndex="150" />
  </ContentGroup>
</PPSPTrackerProtocol>
```

The FIND request MAY include a PeerNum element to indicate 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.

In the case of a FIND with a specific scope of a stream content the request SHOULD include a ContentGroup to specify the content Representations segment range of interest.

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 SUCCESSFULL will be generated and the tracker will include the appropriate list of peers satisfying the conditions requested. 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 [I.D.ietf-alto-protocol].

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

An example of a Response message for the FIND Request is the following:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PPSPTrackerProtocol version="1.0">
  <Response>SUCCESSFUL</Response>
  <TransactionID>12345</TransactionID>
  <PeerGroup>
    <PeerInfo>
      <PeerID>956264622298</PeerID>
      <PeerAddress addrType="ipv4" ip="198.51.100.22" port="80" asn="64496" />
    </PeerInfo>
    <PeerInfo>
      <PeerID>3332001256741</PeerID>
      <PeerAddress addrType="ipv4" ip="198.51.100.201" port="80" asn="64496" />
    </PeerInfo>
  </PeerGroup>
</PPSPTrackerProtocol>
```

The Response MUST include a PeerGroup with PeerInfo data that includes the public IP address of the selected active peers in the swarm.

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.

8.5. STAT_REPORT Request

This method allows the exchange of statistic and status data between peers and trackers to improve system performance. The method is initiated by the peer, periodically while active.

The peer MUST properly form the XML message-body, 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 a StatisticsGroup containing multiple Stat elements describing several properties relevant to the P2P network. These properties can be related with stream statistics, peer status information and content data information, like chunkmaps.

Other properties may be defined, related for example, with incentives and reputation mechanisms.

In case no StatisticsGroup is included, the STAT_REPORT may be used as a "keep-alive" message, to prevent the Tracker from de-registering the peer when timer expired.

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PPSPTrackerProtocol version="1.0">
  <Request>STAT_REPORT</Request>
  <PeerID>656164657221</PeerID>
  <TransactionID>12345</TransactionID>
  <StatisticsGroup>
    <Stat property="StreamStatistics">
      <SwarmID>1111</SwarmID>
      <UploadedBytes>512</UploadedBytes>
      <DownloadedBytes>768</DownloadedBytes>
      <AvailBandwidth>1024000</AvailBandwidth>
    </Stat>
    <Stat property="StreamStatistics">
      <SwarmID>2222</SwarmID>
      <UploadedBytes>1024</UploadedBytes>
      <DownloadedBytes>2048</DownloadedBytes>
      <AvailBandwidth>512000</AvailBandwidth>
    </Stat>
    <Stat property="ContentMap">
      <SwarmID>1111</SwarmID>
      <Representation id="tag0">
        <SegmentInfo startIndex="0" endIndex="24">
          A/BD/8D/8D/wP/A/BD/8D/8D/wP/A/BD/8D/wP/....
        </SegmentInfo>
      </Representation>
      <Representation id="tag1">
        <SegmentInfo startIndex="0" endIndex="14">
          A/BD/8D/wP/A/BD/wP/A/BD/8D/wP/....
        </SegmentInfo>
      </Representation>
    </Stat>
  </StatisticsGroup>
</PPSPTrackerProtocol>
```
If the request is valid the tracker process the received information for future use, and generates a response message with a Response value of SUCCESSFULL.

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

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

```xml
<?xml version="1.0" encoding="UTF-8"?>
<PPSPTrackerProtocol version="1.0">
  <Response>SUCCESSFUL</Response>
  <TransactionID>12345</TransactionID>
</PPSPTrackerProtocol>
```
8.6. 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 eventual 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 6.2) 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 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 6.2). 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, "Already Connected".

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.

9. 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].

9.1. 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 are required to be received from authorized peers.

For that purpose a peer must 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 [I-D.ietf-oauth-v2] with bearer token, which provides the peer with the information required to successfully utilize an access token to make protected requests to the tracker [I-D.ietf-oauth-v2-bearer].
9.2. 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.

9.3. 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.

9.4. Pro-incentive parameter trustfulness

Property types for STAT_REPORT messages may consider pro-incentive parameters, 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. For example, ChunkMaps are essential, and need to be accurate. The P2P system should be designed in a way such that a peer will have the incentive to report truthfully its ChunkMaps (otherwise it may penalize itself, as in the case of under-reporting addressed in [prTorrent]).

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. 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].

10. IANA Considerations

There are presently no IANA considerations with this document.

11. Acknowledgments

The authors would like to thank many people for for their help and comments, particularly: Zhang Yunfei, Liao Hongluan, Roni Even, 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 and Arno Bakker.

The authors would also like to thank the people participating in the EU FP7 project SARACEN (contract no. ICT-248474) [refs.saracenwebpage] for contributions and feedback to this document.

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 or the European Commission.
12. References

12.1. Normative References


12.2. Informative References


(work in progress), November 2011.


Appendix B. Media Presentation Description (MPD)

The MPD file describes a Media Presentation, i.e., a bounded or unbounded presentation of media content. In particular, it defines formats to announce resource identifiers for segments and subsegments (layers in case of SVC, descriptions in case of MDC, or views in case of 3D) and to provide the context for these identified resources within a Media Presentation, i.e., describes the structure of the media, the codecs used (as registered with the MP4 registration authority [MP4REG]), the segments and the corresponding mapping within a container file system.

The MPD contains information about the preferred Connection Trackers, than can be classified in tiers of priority (attribute @tier).

The MPD is a Well-Formed XML Document, encoded as double-byte Unicode. The XML-Schema of the MPD aligns with ISO/IEC 23009-1 [ISO.IEC.23009-1].

The following example of MPD is for an on-demand media program encoded in SVC with two alternative SVC streams, two audio streams and a text stream. The example SVC stream has one base layer representation with two complementary enhancement layers for one video resolution and another SVC stream with a base layer and one complementary enhancement representation for a higher video resolution, an audio stream in English and another in Portuguese, and a timed subtitle file in Portuguese. The contents have protection schemes and include the root fingerprints (attribute @hash of element RootFP) in each video and audio groups (for integrity verification purposes).

```xml
<?xml version="1.0" encoding="UTF-8"?>
<MPD type="OnDemand">
  <ProgramInformation>
    <Title>Movie in SVC</Title>
  </ProgramInformation>
  <Trackers>
    <Tracker url="http://example.com:80" tier="1" />
    <Tracker url="http://example.net:80" tier="2" />
  </Trackers>
</MPD>
```
</Trackers>
<SwarmID>1234</SwarmID>
<Period>
<BaseURL>Program01</BaseURL>
<Group mimeType="video; codecs=h264/SVC" lang="en">
  <Representation frameRate="15" width="1280" height="720"
    id="tag0" bandwidth="32000">
    <ContentProtection schemeIdUri="urn:uuid:706D6953-656C....">
      <RootFP hash="57438tgfkv...." />
    </ContentProtection>
    <SegmentInfo startIndex="0" endIndex="150"
      duration="PT2.00S" levels="3" />
  </Representation>
  <Representation frameRate="30" width="1920" height="1080"
    id="tag3" bandwidth="256000">
    <ContentProtection schemeIdUri="urn:uuid:706D6953-656C....">
      <RootFP hash="95448trf6v...." />
    </ContentProtection>
    <SegmentInfo startIndex="0" endIndex="150"
      duration="PT2.00S" levels="2" />
  </Representation>
</Group>
<Group mimeType="video; codecs=h264/SVC" lang="en">
  <Representation frameRate="30" width="1280" height="720"
    id="tag1" bandwidth="64000"
    dependencyId="tag0">
    <ContentProtection schemeIdUri="urn:uuid:706D6953-656C....">
      <RootFP hash="2356ac468k...." />
    </ContentProtection>
    <SegmentInfo startIndex="0" endIndex="150"
      duration="PT2.00S" />
  </Representation>
  <Representation frameRate="60" width="1920" height="1080"
    id="tag4" bandwidth="512000"
    dependencyId="tag3">
    <ContentProtection schemeIdUri="urn:uuid:706D6953-656C....">
      <RootFP hash="98216d99ab...." />
    </ContentProtection>
    <SegmentInfo startIndex="0" endIndex="150"
      duration="PT2.00S" />
  </Representation>
</Group>
<Group mimeType="video; codecs=h264/SVC" lang="en">
  <ContentProtection schemeIdUri="urn:uuid:706D6953-656C....">
    <RootFP hash="364t96au9d...." />
  </ContentProtection>
  <Representation frameRate="60" width="1280" height="720"
    id="tag2" bandwidth="256000"
dependencyId="tag0 tag1">
  <SegmentInfo startIndex="0" endIndex="150"
duration="PT2.00S" />
</Representation>
</Group>

<Group mimeType="audio/mp4; codecs=mp4a" lang="en"
  bandwidth="64000">
  <ContentProtection schemeIdUri="http://example.net/drm">
    <RootFP hash="26ft54zd9a...." />
  </ContentProtection>
  <Representation id="tag5">
    <SegmentInfo startIndex="0" endIndex="150"
duration="PT2.00S" />
  </Representation>
</Group>

<Group mimeType="audio/mp4; codecs=mp4a" lang="pt"
  bandwidth="64000">
  <ContentProtection schemeIdUri="http://example.net/drm">
    <RootFP hash="64fg53zn53...." />
  </ContentProtection>
  <Representation id="tag6">
    <SegmentInfo startIndex="0" endIndex="150"
duration="PT2.00S" />
  </Representation>
</Group>

<Group mimeType="application/ttml+xml" lang="pt">
  <Representation id="tag7">
    <SegmentInfo>subtitles/Program01-pt.xml</SegmentInfo>
  </Representation>
</Group>
</Period>
</MPD>

The MPD file for P2P Streaming contains tracker information and can be compressed with GZIP file format [RFC1952] in order to be used with HTTP compression [RFC2616] for faster transmission times and less network bandwidth usage.

The Client Media Player parses the downloaded MPD file and, if it includes information for P2P Streaming, sends the information to the peer and waits for the response in order to start requesting media chunks to decode and play-out.

The MPD file for Live Streaming has a similar structure but describes a sliding window of a small range in the SegmentInfo element from the live program stream timeline (typically, 10 seconds of video). The sliding window is updated for every new encoded segments (a range of chunks defined by the attributes @startIndex and @endIndex) of the
program stream.

The following excerpt of MPD is for a Live scalable video content. The MPD is updated every 10 seconds while the content is being encoded in real-time. Note that each segment set defined in the Live MPD is self-contained and the necessary information related to eventual content protection and integrity verification keys for the set is provided:

```xml
<?xml version="1.0" encoding="UTF-8"?>
<MPD type="Live"
 availabilityStartTime="2001-12-17T09:40Z"
 availabilityEndTime="2001-12-17T09:50Z"
 minBufferTime="PT10.00S"
 minimumUpdatePeriodMPD="PT10S">
<SwarmID>654321xyz</SwarmID>
<Period start="PT11S">
  <Group mimeType="video; codecs=h264/SVC" lang="en">
    <Representation frameRate="15" width="1280" height="720"
 id="tag0" bandwidth="32000">
      <ContentProtection schemeIdUri="urn:uuid:706D6953-656C....">
        <RootFP hash="57438tgfkv...." /></ContentProtection>
      <SegmentInfo startIndex="5" endIndex="9" duration="PT2.00S" levels="3" />
    </Representation>
    .... more descriptions ....
  </Group>
  .... more descriptions ....
</Period>
</MPD>
```

Appendix C. PPSP Requirements Compliance

C.1. PPSP Basic Requirements

PPSP.REQ-1: The design of the Tracker protocol in this document allows the Peer Protocol to be similar in terms of design, message formats and flows.

PPSP.REQ-2: The design of the Tracker protocol in this document enables peers to receive streaming content within required time constraints.

PPSP.REQ-3: Each peer has a unique ID (i.e., Peer-ID) that identifies the peer in all swarms joined.
PPSP.REQ-4: Each streaming content is uniquely identified by a Swarm-ID.

PPSP.REQ-5: The streaming content is partitioned into chunks individually addressable.

PPSP.REQ-6: Each chunk has an unique ID in the swarm and is individually addressable.

PPSP.REQ-7: The Tracker Protocol is carried over TCP.

PPSP.REQ-8: The Tracker Protocol is designed to facilitate acceptable QoS, supporting, without special algorithms, adaptive and scalable video and 3D video, for both Video On Demand (VoD) and Live video services, allowing additionally complementary mechanisms like super peers, in-network storage, alternative peer addresses and usage of QoS information for advanced peer selection.

C.2. PPSP Tracker Protocol Requirements

PPSP.TP.REQ-1: The Tracker Protocol implements the reception of queries from peers, such as those in JOIN and FIND messages and periodical peer status reports (STAT_REPORT), as well as the corresponding replies.

PPSP.TP.REQ-2: The peer MUST implement the Tracker Protocol designed in this draft.

PPSP.TP.REQ-3: The tracker request messages JOIN and FIND allow the requesting of peer list from the tracker with respect to a specific Swarm-ID and include preferred number of peers in the peer list as well as peer properties which enable appropriate candidate peer selections by the tracker.

PPSP.TP.REQ-4: The tracker responses from JOIN and FIND messages allow the tracker to offer the peer list to the requesting peer with respect to a specific Swarm-ID.

PPSP.TP.REQ-5: The Tracker supports generating the peer lists with the help of traffic optimization services like ALTO.

PPSP.TP.REQ-6: The STATUS_REPORT message informs the Tracker about the peer’s activity in the swarm.

PPSP.TP.REQ-7: The chunk availability information (ChunkMaps) of the Peer (for all joined swarms) is reported to the tracker in STATUS_REPORT messages.
PPSP.TP.REQ-8: The ChunkMaps exchanged between peer and tracker can be expressed as compact encoded strings.

PPSP.TP.REQ-9: The STATUS_REPORT message informs the tracker about the peer status and capabilities.

C.3. PPSP Security Considerations

PPSP.SEC.REQ-1: The Tracker Protocol supports closed swarms, where the peers are required to be authenticated.

PPSP.SEC.REQ-2: Confidentiality of the streaming content can be supported, and the corresponding key management mechanisms can be negotiated in the authentication and authorization phase (via CONNECT message) before the peer JOINs the swarm.

PPSP.SEC.REQ-3: The Tracker Protocol uses security layers to encrypt the data exchanged among the PPSP entities.

PPSP.SEC.REQ-4: The Tracker Protocol security layer mechanisms help to limit potential damages caused by malfunctioning and badly behaving peers in the P2P streaming system. The streaming mechanisms considered in the PPSP-TP model prevent pollution of contents.

PPSP.SEC.REQ-6: The use of trusted trackers and peer authentication and authorization mechanisms capable to provide additional security and confidentiality, allow to mitigate and prevent peers from DoS attacks.

PPSP.SEC.REQ-7: The Tracker Protocol design supports distributed tracker architectures, providing robustness to the streaming service in case of centralized tracker failure.

PPSP.SEC.REQ-8: The Tracker Protocol use of Transport Layer Security mechanisms avoids the need for developing new security mechanisms.

PPSP.SEC.REQ-9: The Tracker Protocol together with the Media Presentation Description (MPD) allow the use of streaming content integrity mechanisms.

Authors’ Addresses

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

The Peer-to-Peer Streaming Peer Protocol (PPSPP) is a peer-to-peer based transport protocol for content dissemination. It can be used for streaming on-demand and live video content, as well as conventional downloading. In PPSPP, the clients consuming the content participate in the dissemination by forwarding the content to other clients via a mesh-like structure. It is a generic protocol which can run directly on top of UDP, TCP, or as a RTP profile. Features of PPSPP are short time-till-playback and extensibility. Hence, it can use different mechanisms to prevent freeriding, and work with different peer discovery schemes (centralized trackers or Distributed Hash Tables). Depending on the underlying transport protocol, PPSPP can also use different congestion control algorithms, such as LEDBAT, and offer transparent NAT traversal. Finally, PPSPP maintains only a small amount of state per peer and detects malicious modification of content. This documents describes PPSPP and how it satisfies the requirements for the IETF Peer-to-Peer Streaming Protocol (PPSP) Working Group’s peer protocol.

Status of this memo

This Internet-Draft is submitted to IETF in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt.

The list of Internet-Draft Shadow Directories can be accessed at
Table of Contents

1. Introduction ................................................. 3
   1.1. Purpose ............................................. 3
   1.2. Conventions Used in This Document .......... 4
   1.3. Terminology ......................................... 5
2. Overall Operation ........................................... 6
   2.1. Joining a Swarm .................................... 6
   2.2. Exchanging Chunks .................................. 6
   2.3. Leaving a Swarm .................................... 7
3. Messages .................................................. 7
   3.1. HANDSHAKE .......................................... 8
   3.3. HAVE ................................................. 8
      3.3.1. Bin Numbers .................................. 8
      3.3.2. HAVE Message .................................. 9
   3.4. ACK ................................................. 9
   3.5. DATA and HASH ...................................... 10
      3.5.1. Merkle Hash Tree ............................. 10
      3.5.2. Content Integrity Verification ............. 11
      3.5.3. The Atomic Datagram Principle .......... 12
      3.5.4. DATA and HASH Messages .................... 12
      3.5.5. Overhead ...................................... 13
   3.6. HINT ............................................... 13
   3.7. Peer Address Exchange and NAT Hole Punching 14
      3.7.1 PEX_REQ and PEX_RES Messages ............... 14
      3.7.2 Hole Punching via PPSPP Messages .......... 14
   3.8. Keep Alive Signaling ............................... 15
   3.9. VERSION ............................................ 15
   3.10. Conveying Peer Capabilities .................... 15
   3.11. Directory Lists .................................. 15
   3.12. Storage Independence .............................. 15
4. Automatic Detection of Content Size ..................... 16
1. Introduction

1.1. Purpose

This document describes the Peer-to-Peer Streaming Peer Protocol (PPSPP), designed from the ground up for the task of disseminating the same content to a group of interested parties. PPSPP supports streaming on-demand and live video content, as well as conventional downloading, thus covering today’s three major use cases for content
distribution. To fulfill this task, clients consuming the content are put on equal footing with the servers initially providing the content to create a peer-to-peer system where everyone can provide data.

PPSPP uses a simple method of naming content based on self-certification. In particular, content in PPSPP is identified by a single cryptographic hash that is the root hash in a Merkle hash tree calculated recursively from the content [ABMRKL]. This self-certifying hash tree allows every peer to directly detect when a malicious peer tries to distribute fake content. It also ensures only a small amount of information is needed to start a download (just the root hash and some peer addresses).

PPSPP uses a novel method of addressing chunks of content called "bin numbers". Bin numbers allow the addressing of a binary interval of data using a single integer. This reduces the amount of state that needs to be recorded per peer and the space needed to denote intervals on the wire, making the protocol light-weight. In general, this numbering system allows PPSPP to work with simpler data structures, e.g. to use arrays instead of binary trees, thus reducing complexity.

PPSPP is a generic protocol which can run directly on top of UDP, TCP, or as a layer below RTP, similar to SRTP [RFC3711]. As such, PPSPP defines a common set of messages that make up the protocol, which can have different representations on the wire depending on the lower-level protocol used. When the lower-level transport is UDP, PPSPP can also use different congestion control algorithms and facilitate NAT traversal.

In addition, PPSPP is extensible in the mechanisms it uses to promote client contribution and prevent freeriding, that is, how to deal with peers that only download content but never upload to others. Furthermore, it can work with different peer discovery schemes, such as centralized trackers or fast Distributed Hash Tables [JIM11].

This documents describes not only the PPSPP protocol but also how it satisfies the requirements for the IETF Peer-to-Peer Streaming Protocol (PPSP) Working Group’s peer protocol [PPSPCHART,I-D.ietf-ppsp-reqs]. A reference implementation of PPSPP over UDP is available [SWIFTIMPL].

1.2. Conventions Used in This Document

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].
1.3. Terminology

message
   The basic unit of PPSPP communication. A message will have
different representations on the wire depending on the transport
protocol used. Messages are typically multiplexed into a
datagram for transmission.

datagram
   A sequence of messages that is offered as a unit to the
underlying transport protocol (UDP, etc.). The datagram is
PPSPP’s Protocol Data Unit (PDU).

content
   Either a live transmission, a pre-recorded multimedia asset, or
a file.

bin
   A number denoting a specific binary interval of the content
   (i.e., one or more consecutive chunks).

chunk
   The basic unit in which the content is divided. E.g. a block of
   N kilobyte.

hash
   The result of applying a cryptographic hash function, more
   specifically a modification detection code (MDC) [HAC01], such
   as SHA1 [FIPS180-2], to a piece of data.

root hash
   The root in a Merkle hash tree calculated recursively from the
   content.

swarm
   A group of peers participating in the distribution of the same
   content.

swarm ID
   Unique identifier for a swarm of peers, in PPSPP the root hash
   of the content (video-on-demand, download) or a public key (live
   streaming).

tracker
   An entity that records the addresses of peers participating in a
   swarm, usually for a set of swarms, and makes this membership
   information available to other peers on request.
2. Overall Operation

The basic unit of communication in PPSPP is the message. Multiple messages are multiplexed into a single datagram for transmission. A datagram (and hence the messages it contains) will have different representations on the wire depending on the transport protocol used (see Sec. 6).

The overall operation of PPSPP is illustrated in the following examples. The examples assume that the recommended methods for content integrity protection and chunk addressing are used, and a specific policy for which selecting chunks to download.

2.1. Joining a Swarm

Consider a peer A that wants to download a certain content asset. To commence a PPSPP download, peer A must have the swarm ID of the content and a list of one or more tracker contact points (e.g. host+port). The list of trackers is optional in the presence of a decentralized tracking mechanism. The swarm ID consists of the PPSPP root hash of the content (video-on-demand, downloading) or a public key (live streaming).

Peer A now registers with the tracker following e.g. the PPSP tracker protocol [I-D.ietf.psp-reqs] and receives the IP address and port of peers already in the swarm, say B, C, and D. Peer A now sends a datagram containing a HANDSHAKE message to B, C, and D. This message serves as an end-to-end check that the peers are actually in the correct swarm, and contains the root hash of the swarm. Peer B and C respond with datagrams containing a HANDSHAKE message and one or more HAVE messages. A HAVE message conveys (part of) the chunk availability of a peer and thus contains a bin number that denotes what chunks of the content peer B, resp. C have. Peer D sends a datagram with just a HANDSHAKE and omits HAVE messages as a way of choking A.

2.2. Exchanging Chunks

In response to B and C, A sends new datagrams to B and C containing HINT messages. A HINT or request message indicates the chunks that a peer wants to download, and contains a bin number. The HINT messages to B and C refer to disjunct sets of chunks. B and C respond with
Internet-Draft                   PPSPP                  January 30, 2012

datagrams containing HASH, HAVE and DATA messages. The HASH messages
contains all cryptographic hashes that peer A needs to verify the
integrity of the content chunk sent in the DATA message, using the
content’s root hash as trusted anchor, see Sec. 3.5. Using these
hashes peer A verifies that the chunks received from B and C are
correct. It also updates the chunk availability of B and C using the
information in the received HAVE messages.

After processing, A sends a datagram containing HAVE messages for the
chunks it just received to all its peers. In the datagram to B and C
it includes an ACK message acknowledging the receipt of the chunks,
and adds HINT messages for new chunks. ACK messages are not used when
a reliable transport protocol is used. When e.g. C finds that A
obtained a chunk (from B) that C did not yet have, C’s next datagram
includes a HINT for that chunk.

Peer D does not send HAVE messages to A when it downloads chunks from
other peers, until D decides to unchoke peer A. In the case, it sends
a datagram with HAVE messages to inform A of its current
availability. If B or C decide to choke A they stop sending HAVE and
DATA messages and A should then rerequest from other peers. They may
continue to send HINT messages, or periodic KEEPALIVE messages such
that A keeps sending them HAVE messages.

Once peer A has received all content (video-on-demand use case) it
stops sending messages to all other peers that have all content
(a.k.a. seeders). Peer A MAY also contact the tracker or another
source again to obtain more peer addresses.

2.3. Leaving a Swarm

Depending on the transport protocol used, peers should either use
explicit leave messages or implicitly leave a swarm by stopping to
respond to messages. Peers that learn about the departure should
remove these peers from the current peer list. The implicit-leave
mechanism works for both graceful and ungraceful leaves (i.e., peer
crashes or disconnects). When leaving gracefully, a peer should
deregister from the tracker following the (PPSP) tracker protocol.

3. Messages

In general, no error codes or responses are used in the protocol;
absence of any response indicates an error. Invalid messages are
discarded.

For the sake of simplicity, one swarm of peers always deals with one
content asset (e.g. file) only. Retrieval of large collections of files is done by retrieving a directory list file and then recursively retrieving files, which might also turn to be directory lists, as described in Sec. 3.11.

3.1. HANDSHAKE

As an end-to-end check that the peers are actually in the correct swarm, the initiating peer and the addressed peer SHOULD send a HANDSHAKE message in the first datagrams they exchange. The only payload of the HANDSHAKE message is the root hash of the content.

After the handshakes are exchanged, the initiator knows that the peer really responds. Hence, the second datagram the initiator sends MAY already contain some heavy payload. To minimize the number of initialization roundtrips, implementations MAY dispense with the HANDSHAKE message. To the same end, the first two datagrams exchanged MAY also contain some minor payload, e.g. HAVE messages to indicate the current progress of a peer or a HINT (see Sec. 3.6).

3.3. HAVE

The HAVE message is used to convey which chunks a peer has available, expressed in a new content addressing scheme called "bin numbers".

3.3.1. Bin Numbers

PPSPP employs a generic content addressing scheme based on binary intervals ("bins" in short). The smallest interval is a chunk (e.g. a N kilobyte block), the top interval is the complete 2**63 range. A novel addition to the classical scheme are "bin numbers", a scheme of numbering binary intervals which lays them out into a vector nicely.

Consider an chunk interval of width W. To derive the bin numbers of the complete interval and the subintervals, a minimal balanced binary tree is built that is at least W chunks wide at the base. The leaves from left-to-right correspond to the chunks 0..W in the interval, and have bin number I*2 where I is the index of the chunk (counting beyond W-1 to balance the tree). The higher level nodes P in the tree have bin number

\[ \text{binP} = (\text{binL} + \text{binR}) / 2 \]

where binL is the bin of node P’s left-hand child and binR is the bin of node P’s right-hand child. Given that each node in the tree represents a subinterval of the original interval, each such subinterval now is addressable by a bin number, a single integer. The
bin number tree of a interval of width W=8 looks like this:

```
        7
       / \  \\
      /   \ \\
     /     \\
   /       \\
  3 \      11 \\
  / \    / \\
 /   \  /   \\
1 \ 5 / 9 / 13 \\
/ \ / / / /
0 2 4 6 8 10 12 14
```

So bin 7 represents the complete interval, 3 represents the interval of chunk 0..3 and 1 represents the interval of chunks 0 and 1. The special numbers 0xFFFFFFFF (32-bit) or 0xFFFFFFFFFFFFFFFF (64-bit) stands for an empty interval, and 0x7FFF...FFF stands for "everything".

3.3.2. HAVE Message

When a receiving peer has successfully checked the integrity of a chunk or interval of chunks it MUST send a HAVE message to all peers it wants to interact with. The latter allows the HAVE message to be used as a method of choking. The HAVE message MUST contain the bin number of the biggest complete interval of all chunks the receiver has received and checked so far that fully includes the interval of chunks just received. So the bin number MUST denote at least the interval received, but the receiver is supposed to aggregate and acknowledge bigger bins, when possible.

As a result, every single chunk is acknowledged a logarithmic number of times. That provides some necessary redundancy of acknowledgments and sufficiently compensates for unreliable transport protocols.

To record which chunks a peer has in the state that an implementation keeps for each peer, an implementation MAY use the "binmap" data structure, which is a hybrid of a bitmap and a binary tree, discussed in detail in [BINMAP].
When PPSPP is run over an unreliable transport protocol, an implementation MAY choose to use ACK messages to acknowledge received data. When a receiving peer has successfully checked the integrity of a chunk or interval of chunks C it MUST send a ACK message containing the bin number of its biggest, complete, interval covering C to the sending peer (see HAVE). To facilitate delay-based congestion control, an ACK message contains a timestamp.

3.5. DATA and HASH

The DATA message is used to transfer chunks of content. The associated HASH message carries cryptographic hashes that are necessary for a receiver to check the integrity of the chunk. PPSPP's content integrity protection is based on a Merkle hash tree and works as follows.

3.5.1. Merkle Hash Tree

PPSPP uses a method of naming content based on self-certification. In particular, content in PPSPP is identified by a single cryptographic hash that is the root hash in a Merkle hash tree calculated recursively from the content [ABMRKL]. This self-certifying hash tree allows every peer to directly detect when a malicious peer tries to distribute fake content. It also ensures only a small amount of information is needed to start a download (the root hash and some peer addresses). For live streaming public keys and dynamic trees are used, see below.

The Merkle hash tree of a content asset that is divided into N chunks is constructed as follows. Note the construction does not assume chunks of content to be fixed size. Given a cryptographic hash function, more specifically a modification detection code (MDC) [HAC01], such as SHA1, the hashes of all the chunks of the content are calculated. Next, a binary tree of sufficient height is created. Sufficient height means that the lowest level in the tree has enough nodes to hold all chunk hashes in the set, as before, see HAVE message. The figure below shows the tree for a content asset consisting of 7 chunks. As before with the content addressing scheme, the leaves of the tree correspond to a chunk and in this case are assigned the hash of that chunk, starting at the left-most leaf. As the base of the tree may be wider than the number of chunks, any remaining leaves in the tree are assigned a empty hash value of all zeros. Finally, the hash values of the higher levels in the tree are calculated, by concatenating the hash values of the two children (again left to right) and computing the hash of that aggregate. This process ends in a hash value for the root node, which is called the "root hash". Note the root hash only depends on the content and any
modification of the content will result in a different root hash.

```
7 = root hash
   /
  /  
/    
3*   11
   /  
  |  /  
  |/  
1   5   9  13* = uncle hash
   /  
  |  /  
  |/  
0  2  4  6  8  10* 12  14
C0 C1 C2 C3 C4 C5 C6 E
=chunk index ^^ = empty hash
```

### 3.5.2. Content Integrity Verification

Assuming a peer receives the root hash of the content it wants to download from a trusted source, it can check the integrity of any chunk of that content it receives as follows. It first calculates the hash of the chunk it received, for example chunk C4 in the previous figure. Along with this chunk it MUST receive the hashes required to check the integrity of that chunk. In principle, these are the hash of the chunk's sibling (C5) and that of its "uncles". A chunk's uncles are the sibling Y of its parent X, and the uncle of that Y, recursively until the root is reached. For chunk C4 its uncles are bins 13 and 3, marked with * in the figure. Using this information the peer recalculates the root hash of the tree, and compares it to the root hash it received from the trusted source. If they match the chunk of content has been positively verified to be the requested part of the content. Otherwise, the sending peer either sent the wrong content or the wrong sibling or uncle hashes. For simplicity, the set of sibling and uncles hashes is collectively referred to as the "uncle hashes".

In the case of live streaming the tree of chunks grows dynamically and content is identified with a public key instead of a root hash, as the root hash is undefined or, more precisely, transient, as long as new data is generated by the live source. Live streaming is described in more detail below, but content verification works the same for both live and predefined content.
3.5.3. The Atomic Datagram Principle

As explained above, a datagram consists of a sequence of messages. Ideally, every datagram sent must be independent of other datagrams, so each datagram SHOULD be processed separately and a loss of one datagram MUST NOT disrupt the flow. Thus, as a datagram carries zero or more messages, neither messages nor message interdependencies should span over multiple datagrams.

This principle implies that as any chunk is verified using its uncle hashes the necessary hashes MUST be put into the same datagram as the chunk’s data (Sec. 3.5.4). As a general rule, if some additional data is still missing to process a message within a datagram, the message SHOULD be dropped.

The hashes necessary to verify a chunk are in principle its sibling’s hash and all its uncle hashes, but the set of hashes to sent can be optimized. Before sending a packet of data to the receiver, the sender inspects the receiver’s previous acknowledgments (HAVE or ACK) to derive which hashes the receiver already has for sure. Suppose, the receiver had acknowledged bin 1 (first two chunks of the file), then it must already have uncle hashes 5, 11 and so on. That is because those hashes are necessary to check packets of bin 1 against the root hash. Then, hashes 3, 7 and so on must be also known as they are calculated in the process of checking the uncle hash chain. Hence, to send bin 12 (i.e. the 7th chunk of content), the sender needs to include just the hashes for bins 14 and 9, which let the data be checked against hash 11 which is already known to the receiver.

The sender MAY optimistically skip hashes which were sent out in previous, still unacknowledged datagrams. It is an optimization tradeoff between redundant hash transmission and possibility of collateral data loss in the case some necessary hashes were lost in the network so some delivered data cannot be verified and thus has to be dropped. In either case, the receiver builds the Merkle tree on-demand, incrementally, starting from the root hash, and uses it for data validation.

In short, the sender MUST put into the datagram the missing hashes necessary for the receiver to verify the chunk.

3.5.4. DATA and HASH Messages

Concretely, a peer that wants to send a chunk of content creates a datagram that MUST consist of one or more HASH messages and a DATA message. The datagram MUST contain a HASH message for each hash the receiver misses for integrity checking. A HASH message MUST contain
the bin number and hash data for each of those hashes. The DATA message MUST contain the bin number of the chunk and chunk itself. A peer MAY send the required messages for multiple chunks in the same datagram.

3.5.5. Overhead

The overhead of using Merkle hash trees is limited. The size of the hash tree expressed as the total number of nodes depends on the number of chunks the content is divided (and hence the size of chunks) following this formula:

\[ \text{nnodes} = \text{math.pow}(2, \text{math.log(nchunks,2)}+1) \]

In principle, the hash values of all these nodes will have to be sent to a peer once for it to verify all chunks. Hence the maximum on-the-wire overhead is hashsize * nnodes. However, the actual number of hashes transmitted can be optimized as described in Sec. 3.5.2. To see a peer can verify all chunks whilst receiving not all hashes, consider the example tree in Sec. 3.5.1.

In case of a simple progressive download, of chunks 0,1,4,6, etc. the sending peer will send the following hashes:

<table>
<thead>
<tr>
<th>Chunk</th>
<th>Bins of hashes sent</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>2,5,11</td>
</tr>
<tr>
<td>1</td>
<td>- (receiver already knows all)</td>
</tr>
<tr>
<td>4</td>
<td>6</td>
</tr>
<tr>
<td>6</td>
<td>-</td>
</tr>
<tr>
<td>8</td>
<td>10,13 (hash 3 can be calculated from 0,2,5)</td>
</tr>
<tr>
<td>10</td>
<td>-</td>
</tr>
<tr>
<td>12</td>
<td>14</td>
</tr>
<tr>
<td>14</td>
<td>-</td>
</tr>
</tbody>
</table>

So the number of hashes sent in total (7) is less than the total number of hashes in the tree (16), as a peer does not need to send hashes that are calculated and verified as part of earlier chunks.

3.6. HINT

While bulk download protocols normally do explicit requests for certain ranges of data (i.e., use a pull model, for example, BitTorrent [BITTORRENT]), live streaming protocols quite often use a request-less push model to save round trips. PPSPP supports both models of operation.
A peer MUST send a HINT message containing the bin of the chunk interval it wants to download. A peer receiving a HINT message MAY send out requested pieces. When it receives multiple HINTs the peer SHOULD process the HINTs sequentially. Multiple HINT messages MAY be sent in one datagram, for example, when the peer wants to request several rare chunks at once.

When live streaming, a peer receiving HINTs also may send some other chunks in case it runs out of requests or for some other reason. In that case the only purpose of HINT messages is to coordinate peers and to avoid unnecessary data retransmission, hence the name.

3.7. Peer Address Exchange and NAT Hole Punching

3.7.1 PEX_REQ and PEX_RES Messages

Peer address exchange messages (or PEX messages for short) are common for many peer-to-peer protocols. By exchanging peer addresses in gossip fashion, peers relieve central coordinating entities (the trackers) from unnecessary work. PPSPP optionally features two types of PEX messages: PEX_REQ and PEX_RES. A peer that wants to retrieve some peer addresses MUST send a PEX_REQ message. The receiving peer MAY respond with a PEX_RES message containing the addresses of several peers. The addresses MUST be of peers it has recently exchanged messages with to guarantee liveness. Alternatively, the receiving peer MAY ignore PEX_REQ messages if uninterested in obtaining new peers or because of security considerations (rate limiting) or any other reason.

The PEX messages can be used to construct a dedicated tracker peer.

3.7.2 Hole Punching via PPSPP Messages

PPSPP can be used in combination with STUN [RFC5389]. In addition, the native PEX_* messages can be used to do simple NAT hole punching [SNP]. To implement this feature, the sending pattern of PEX messages is restricted. In particular, when peer A introduces peer B to peer C by sending a PEX_RES message to C, it SHOULD also send a message to B introducing C. These messages SHOULD be within 2 seconds from each other, but MAY not be, simultaneous, instead leaving a gap of twice the "typical" RTT, i.e. 300-600ms. As a result, the peers are supposed to initiate handshakes to each other thus forming a simple NAT hole punching pattern where the introducing peer effectively acts as a STUN server [RFC5389]. Note that the PEX_RES message is sent without a prior PEX_REQ in this case.
3.8. Keep Alive Signaling

A peer MUST send a "keep alive" message periodically to each peer it wants to interact with in the future, but has no other messages to send them at present. PPSPP does not define an explicit message type for "keep alive" messages. In the PPSP-over-UDP mapping they are implemented as simple datagrams consisting of a 4-byte channel number only, see Sec. 6.1.3 and 6.1.9. When PPSPP is used over TCP, each datagram is prefixed with 4 bytes containing its size, the common method of turning TCP’s stream of bytes into a sequence of datagrams. In that case, a size of 0 is used as keep alive, as in BitTorrent [BITTORRENT].

3.9. VERSION

Peers MUST convey which version of the PPSPP protocol they support using a VERSION message. This message MUST be included in the initial (handshake) datagrams and MUST indicate which version of the PPSPP protocol the sending peer supports.

3.10. Conveying Peer Capabilities

Peers may support just a subset of the PPSPP messages. For example, peers running over TCP may not accept ACK messages, or peers used with a centralized tracking infrastructure may not accept PEX messages. For these reasons, peers SHOULD signal which subset of the PPSPP messages they support by means of the MSGTYPE_RCVD message. This message SHOULD be included in the initial (handshake) datagrams and MUST indicate which PPSPP protocol messages the sending peer supports.

3.11. Directory Lists

Directory list files MUST start with magic bytes ".\n..\n". The rest of the file is a newline-separated list of hashes and file names for the content of the directory. An example:

```
..  
1234567890ABCDEFG1234567890ABCDEFG12345678 readme.txt 
01234567890ABCDEFG1234567890ABCDEFG1234567 big_file.dat
```

3.12. Storage Independence

Note PPSPP does not prescribe how chunks are stored. This also allows users of PPSPP to map different files into a single swarm as in BitTorrent multi-file
torrents [BITTORRENT], and more innovative storage solutions when
variable-sized chunks are used.

4. Automatic Detection of Content Size

In PPSPP, the root hash of a static content asset, such as a video
file, along with some peer addresses is sufficient to start a
download. In addition, PPSPP can reliably and automatically derive
the size of such content from information received from the network
when fixed sized chunks are used. As a result, it is not necessary to
include the size of the content asset as the metadata of the content,
in addition to the root hash. Implementations of PPSPP MAY use this
automatic detection feature. Note this feature is the only feature of
PPSPP that requires that a fixed-sized chunk is used.

4.1. Peak Hashes

The ability for a newcomer peer to detect the size of the content
depends heavily on the concept of peak hashes. Peak hashes, in
general, enable two cornerstone features of PPSPP: reliable file size
detection and download/live streaming unification (see Sec. 5). The
concept of peak hashes depends on the concepts of filled and
incomplete bins. Recall that when constructing the binary trees for
content verification and addressing the base of the tree may have
more leaves than the number of chunks in the content. In the Merkle
hash tree these leaves were assigned empty all-zero hashes to be able
to calculate the higher level hashes. A filled bin is now defined as
a bin number that addresses an interval of leaves that consists only
of hashes of content chunks, not empty hashes. Reversely, an
incomplete (not filled) bin addresses an interval that contains also
empty hashes, typically an interval that extends past the end of the
file. In the following figure bins 7, 11, 13 and 14 are incomplete
the rest is filled.

Formally, a peak hash is a hash in the Merkle tree defined over a
filled bin, whose sibling is defined over an incomplete bin.
Practically, suppose a file is 7162 bytes long and a chunk is 1
kilobyte. That file fits into 7 chunks, the tail chunk being 1018
bytes long. The Merkle tree for that file looks as follows. Following
the definition the peak hashes of this file are in bins 3, 9 and 12,
denoted with a *. E denotes an empty hash.

```
   7
  /  \
```
Peak hashes can be explained by the binary representation of the number of chunks the file occupies. The binary representation for 7 is 111. Every "1" in binary representation of the file's packet length corresponds to a peak hash. For this particular file there are indeed three peaks, bin numbers 3, 9, 12. The number of peak hashes for a file is therefore also at most logarithmic with its size.

A peer knowing which bins contain the peak hashes for the file can therefore calculate the number of chunks it consists of, and thus get an estimate of the file size (given all chunks but the last are fixed size). Which bins are the peaks can be securely communicated from one (untrusted) peer A to another B by letting A send the peak hashes and their bin numbers to B. It can be shown that the root hash that B obtained from a trusted source is sufficient to verify that these are indeed the right peak hashes, as follows.

Lemma: Peak hashes can be checked against the root hash.

Proof: (a) Any peak hash is always the left sibling. Otherwise, be it the right sibling, its left neighbor/sibling must also be defined over a filled bin, because of the way chunks are laid out in the leaves, contradiction. (b) For the rightmost peak hash, its right sibling is zero. (c) For any peak hash, its right sibling might be calculated using peak hashes to the left and zeros for empty bins. (d) Once the right sibling of the leftmost peak hash is calculated, its parent might be calculated. (e) Once that parent is calculated, we might trivially get to the root hash by concatenating the hash with zeros and hashing it repeatedly.

Informally, the Lemma might be expressed as follows: peak hashes cover all data, so the remaining hashes are either trivial (zeros) or might be calculated from peak hashes and zero hashes.

Finally, once peer B has obtained the number of chunks in the content
it can determine the exact file size as follows. Given that all chunks except the last are fixed size B just needs to know the size of the last chunk. Knowing the number of chunks B can calculate the bin number of the last chunk and download it. As always B verifies the integrity of this chunk against the trusted root hash. As there is only one chunk of data that leads to a successful verification the size of this chunk must be correct. B can then determine the exact file size as

\[(\text{number of chunks} - 1) \times \text{fixed chunk size} + \text{size of last chunk}\]

4.2. Procedure

A PPSPP implementation that wants to use automatic size detection MUST operate as follows. When a peer B sends a DATA message for the first time to a peer A, B MUST include all the peak hashes for the content in the same datagram, unless A has already signalled earlier in the exchange that it knows the peak hashes by having acknowledged any bin, even the empty one. The receiver A MUST check the peak hashes against the root hash to determine the approximate content size. To obtain the definite content size peer A MUST download the last chunk of the content from any peer that offers it.

5. Live streaming

In the case of live streaming a transfer is bootstrapped with a public key instead of a root hash, as the root hash is undefined or, more precisely, transient, as long as new data is being generated by the live source. Live/download unification is achieved by sending signed peak hashes on-demand, ahead of the actual data. As before, the sender might use acknowledgements to derive which content range the receiver has peak hashes for and to prepend the data hashes with the necessary (signed) peak hashes. Except for the fact that the set of peak hashes changes with time, other parts of the algorithm work as described in Sec. 3.

As with static content assets in the previous section, in live streaming content length is not known on advance, but derived on-the-go from the peak hashes. Suppose, our 7 KB stream extended to another kilobyte. Thus, new hash 7 becomes the only peak hash, eating hashes 3, 9 and 12. So, the source sends out a SIGNED_HASH message to announce the fact.
The number of cryptographic operations will be limited. For example, consider a 25 frame/second video transmitted over UDP. When each frame is transmitted in its own chunk, only 25 signature verification operations per second are required at the receiver for bitrates up to ~12.8 megabit/second. For higher bitrates multiple UDP packets per frame are needed and the number of verifications doubles.

6. Transport Protocols and Encapsulation

6.1. UDP

6.1.1. Chunk Size

Currently, PPSPP-over-UDP is the preferred deployment option. Effectively, UDP allows the use of IP with minimal overhead and it also allows userspace implementations. The default is to use chunks of 1 kilobyte such that a datagram fits in an Ethernet-sized IP packet. The bin numbering allows to use PPSPP over Jumbo frames/datagrams. Both DATA and HAVE/ACK messages may use e.g. 8 kilobyte packets instead of the standard 1 KiB. The hashing scheme stays the same. Using PPSPP with 512 or 256-byte packets is theoretically possible with 64-bit byte-precise bin numbers, but IP fragmentation might be a better method to achieve the same result.

6.1.2. Datagrams and Messages

When using UDP, the abstract datagram described above corresponds directly to a UDP datagram. Each message within a datagram has a fixed length, which depends on the type of the message. The first byte of a message denotes its type. The currently defined types are:

```
HANDSHAKE = 0x00
DATA = 0x01
ACK = 0x02
HAVE = 0x03
HASH = 0x04
PEX_RES = 0x05
PEX_REQ = 0x06
SIGNED_HASH = 0x07
HINT = 0x08
MSGTYPE_RCVD = 0x09
VERSION = 0x10
```
Furthermore, integers are serialized in the network (big-endian) byte order. So consider the example of an ACK message (Sec 3.4). It has message type of 0x02 and a payload of a bin number, a four-byte integer (say, 1); hence, its on the wire representation for UDP can be written in hex as: "02 00000001". This hex-like two character-per-byte notation is used to represent message formats in the rest of this section.

6.1.3. Channels

As it is increasingly complex for peers to enable UDP communication between each other due to NATs and firewalls, PPSPP-over-UDP uses a multiplexing scheme, called "channels", to allow multiple swarms to use the same UDP port. Channels loosely correspond to TCP connections and each channel belongs to a single swarm. When channels are used, each datagram starts with four bytes corresponding to the receiving channel number.

6.1.4. HANDSHAKE and VERSION

A channel is established with a handshake. To start a handshake, the initiating peer needs to know:

(1) the IP address of a peer
(2) peer’s UDP port and
(3) the root hash of the content (see Sec. 3.5.1).

To do the handshake the initiating peer sends a datagram that MUST start with an all 0-zeros channel number followed by a VERSION message, then a HASH message whose payload is the root hash, and a HANDSHAKE message, whose only payload is a locally unused channel number.

On the wire the datagram will look something like this:

00000000 10 01
04 7FFFFFFF 1234123412341234123412341234123412341234
00 00000011
(to unknown channel, handshake from channel 0x11 speaking protocol version 0x01, initiating a transfer of a file with a root hash 123...1234)

The receiving peer MUST respond with a datagram that starts with the channel number from the sender’s HANDSHAKE message, followed by a VERSION message, then a HANDSHAKE message, whose only payload is a locally unused channel number, followed by any other messages it wants to send.
Peer’s response datagram on the wire:

00000011 10 01
00 00000022 03 00000003

(peer to the initiator: use channel number 0x22 for this transfer and proto version 0x01; I also have first 4 chunks of the file, see Sec. 4.3)

At this point, the initiator knows that the peer really responds; for that purpose channel ids MUST be random enough to prevent easy guessing. So, the third datagram of a handshake MAY already contain some heavy payload. To minimize the number of initialization roundtrips, the first two datagrams MAY also contain some minor payload, e.g. a couple of HAVE messages roughly indicating the current progress of a peer or a HINT (see Sec. 3.6). When receiving the third datagram, both peers have the proof they really talk to each other; three-way handshake is complete.

A peer MAY explicit close a channel by sending a HANDSHAKE message that MUST contain an all 0-zeros channel number.

On the wire:

00 00000000

6.1.5. HAVE

A HAVE message (type 0x03) states that the sending peer has the complete specified bin and successfully checked its integrity:

03 00000003

(got/checked first four kilobytes of a file/stream)

6.1.6. ACK

An ACK message (type 0x02) acknowledges data that was received from its addressee; to facilitate delay-based congestion control, an ACK message contains a timestamp, in particular, a 64-bit microsecond time.

02 00000002 12345678

(got the second kilobyte of the file from you; my microsecond timer was showing 0x12345678 at that moment)

6.1.7. HASH

A HASH message (type 0x04) consists of a four-byte bin number and the cryptographic hash (e.g. a 20-byte SHA1 hash)
6.1.8. DATA

A DATA message (type 0x01) consists of a four-byte bin number and the actual chunk. In case a datagram contains a DATA message, a sender MUST always put the data message in the tail of the datagram. For example:

01 00000000 48656c6c6f20776f726c6421
(This message accommodates an entire file: "Hello world!")

6.1.9. KEEPALIVE

Keepalives do not have a message type on UDP. They are just simple datagrams consisting of a 4-byte channel id only.

On the wire:
00000022

6.1.10. Flow and Congestion Control

Explicit flow control is not necessary in PPSPP-over-UDP. In the case of video-on-demand the receiver will request data explicitly from peers and is therefore in control of how much data is coming towards it. In the case of live streaming, where a push-model may be used, the amount of data incoming is limited to the bitrate, which the receiver must be able to process otherwise it cannot play the stream. Should, for any reason, the receiver get saturated with data that situation is perfectly detected by the congestion control. PPSPP-over-UDP can support different congestion control algorithms, in particular, it supports the new IETF Low Extra Delay Background Transport (LEDBAT) congestion control algorithm that ensures that peer-to-peer traffic yields to regular best-effort traffic [LEDBAT].

6.2. TCP

When run over TCP, PPSPP becomes functionally equivalent to BitTorrent. Namely, most PPSPP messages have corresponding BitTorrent messages and vice versa, except for BitTorrent’s explicit interest declarations and choking/unchoking, which serve the classic implementation of the tit-for-tat algorithm [TIT4TAT]. However, TCP is not well suited for multiparty communication, as argued in Sec. 9.

6.3. RTP Profile for PPSP
In this section we sketch how PPSPP can be integrated into RTP [RFC3550] to form the Peer-to-Peer Streaming Protocol (PPSP) [I-D.ietf-ppsp-reqs] running over UDP. The PPSP charter requires existing media transfer protocols be used [PPSPCHART]. Hence, the general idea is to define PPSPP as a profile of RTP, in the same way as the Secure Real-time Transport Protocol (SRTP) [RFC3711]. SRTP, and therefore PPSPP is considered "a "bump in the stack" implementation which resides between the RTP application and the transport layer. [PPSPP] intercepts RTP packets and then forwards an equivalent [PPSPP] packet on the sending side, and intercepts [PPSPP] packets and passes an equivalent RTP packet up the stack on the receiving side." [RFC3711].

In particular, to encode a PPSPP datagram in an RTP packet all the non-DATA messages of PPSPP such as HINT and HAVE are postfixed to the RTP packet using the UDP encoding and the content of DATA messages is sent in the payload field. Implementations MAY omit the RTP header for packets without payload. This construction allows the streaming application to use of all RTP’s current features, and with a modification to the Merkle tree hashing scheme (see below) meets PPSPP’s atomic datagram principle. The latter means that a receiving peer can autonomously verify the RTP packet as being correct content, thus preventing the spread of corrupt data (see requirement PPSP.SEC-REQ-4).

The use of ACK messages for reliability is left as a choice of the application using PPSP.

6.3.1. Design

6.3.1.1. Joining a Swarm

To commence a PPSP download a peer A must have the swarm ID of the stream and a list of one or more tracker contact points (e.g. host+port). The list of trackers is optional in the presence of a decentralized tracking mechanism. The swarm ID consists of the PPSPP root hash of the content, which is divided into chunks (see Discussion).

Peer A now registers with the PPSP tracker following the tracker protocol [I-D.ietf.ppsp-reqs] and receives the IP address and RTP port of peers already in the swarm, say B, C, and D. Peer A now sends an RTP packet containing a HANDSHAKE without channel information to B, C, and D. This serves as an end-to-end check that the peers are actually in the correct swarm. Optionally A could include a HINT message in some RTP packets if it wants to start receiving content immediately. B and C respond with a HANDSHAKE and HAVE messages.
sends just a HANDSHAKE and omits HAVE messages as a way of choking A.

6.3.1.2. Exchanging Chunks

In response to B and C, A sends new RTP packets to B and C with HINTs for disjunct sets of chunks. B and C respond with the requested chunks in the payload and HAVE messages, updating their chunk availability. Upon receipt, A sends HAVE for the chunks received and new HINT messages to B and C. When e.g. C finds that A obtained a chunk (from B) that C did not yet have, C’s response includes a HINT for that chunk.

D does not send HAVE messages, instead if D decides to unchoke peer A, it sends an RTP packet with HAVE messages to inform A of its current availability. If B or C decide to choke A they stop sending HAVE and DATA messages and A should then rerequest from other peers. They may continue to send HINT messages, or exponentially slowing KEEPALIVE messages such that A keeps sending them HAVE messages.

Once A has received all content (video-on-demand use case) it stops sending messages to all other peers that have all content (a.k.a. seeders).

6.3.1.3. Leaving a Swarm

Peers can implicitly leave a swarm by stopping to respond to messages. Sending peers should remove these peers from the current peer list. This mechanism works for both graceful and ungraceful leaves (i.e., peer crashes or disconnects). When leaving gracefully, a peer should deregister from the tracker following the PPSPP tracker protocol.

More explicit graceful leaves could be implemented using RTCP. In particular, a peer could send a RTCP BYE on the RTCP port that is derivable from a peer’s RTP port for all peers in its current peer list. However, to prevent malicious peers from sending BYEs a form of peer authentication is required (e.g. using public keys as peer IDs [PERMIDS].)

6.3.1.4. Discussion

Using PPSPP as an RTP profile requires a change to the content integrity protection scheme (see Sec. 3.5). The fields in the RTP header, such as the timestamp and PT fields, must be protected by the Merkle tree hashing scheme to prevent malicious alterations.
Therefore, the Merkle tree is no longer constructed from pure content chunks, but from the complete RTP packet for a chunk as it would be transmitted (minus the non-DATA PPSPP messages). In other words, the hash of the leaves in the tree is the hash over the Authenticated Portion of the RTP packet as defined by SRTP, illustrated in the following figure (extended from [RFC3711]). There is no need for the RTP packets to be fixed size, as the hashing scheme can deal with variable-sized leaves.
As a downside, with variable-sized payloads the automatic content size detection of Section 4 no longer works, so content length MUST be explicit in the metadata. In addition, storage on disk is more complex with out-of-order, variable-sized packets. On the upside, carrying RTP over PPSPP allow decryption-less caching.

As with UDP, another matter is how much data is carried inside each packet. An important PPSPP-specific factor here is the resulting number of hash calculations per second needed to verify chunks. Experiments should be conducted to ensure they are not excessive for, e.g., mobile hardware.

At present, Peer IDs are not required in this design.

6.3.2. PPSP Requirements

6.3.2.1. Basic Requirements

- PPSP.REQ-1: The PPSP PEX message can also be used as the basis for a tracker protocol, to be discussed elsewhere.

- PPSP.REQ-2: This draft preserves the properties of RTP.
- PPSP.REQ-3: This draft does not place requirements on peer IDs, IP+port is sufficient.

- PPSP.REQ-4: The content is identified by its root hash (video-on-demand) or a public key (live streaming).

- PPSP.REQ-5: The content is partitioned by the streaming application.

- PPSP.REQ-6: Each chunk is identified by a bin number (and its cryptographic hash.)

- PPSP.REQ-7: The protocol is carried over UDP because RTP is.

- PPSP.REQ-8: The protocol has been designed to allow meaningful data transfer between peers as soon as possible and to avoid unnecessary round-trips. It supports small and variable chunk sizes, and its content integrity protection enables wide scale caching.

6.3.2.2. Peer Protocol Requirements

- PPSP.PP.REQ-1: A GET_HAVE would have to be added to request which chunks are available from a peer, if the proposed push-based HAVE mechanism is not sufficient.

- PPSP.PP.REQ-2: A set of HAVE messages satisfies this.

- PPSP.PP.REQ-3: The PEX_REQ message satisfies this. Care should be taken with peer address exchange in general, as the use of such hearsay is a risk for the protocol as it may be exploited by malicious peers (as a DDoS attack mechanism). A secure tracking / peer sampling protocol like [PUPPETCAST] may be needed to make peer-address exchange safe.

- PPSP.PP.REQ-4: HAVE messages convey current availability via a push model.

- PPSP.PP.REQ-5: Bin numbering enables a compact representation of chunk availability.

- PPSP.PP.REQ-6: A new PPSP specific Peer Report message would have to be added to RTCP.

- PPSP.PP.REQ-7: Transmission and chunk requests are integrated in this protocol.
6.3.2.3. Security Requirements

- **PPSP.SEC.REQ-1**: An access control mechanism like Closed Swarms [CLOSED] would have to be added.

- **PPSP.SEC.REQ-2**: As RTP is carried verbatim over PPSPP, RTP encryption can be used. Note that just encrypting the RTP part will allow for caching servers that are part of the swarm but do not need access to the decryption keys. They just need access to the PPSPP HASHES in the postfix to verify the packet’s integrity.

- **PPSP.SEC.REQ-3**: RTP encryption or IPsec [RFC4303] can be used, if the PPSPP messages must also be encrypted.

- **PPSP.SEC.REQ-4**: The Merkle tree hashing scheme prevents the indirect spread of corrupt content, as peers will only forward chunks to others if their integrity check out. Another protection mechanism is to not depend on hearsay (i.e., do not forward other peers’ availability information), or to only use it when the information spread is self-certified by its subjects.

Other attacks, such as a malicious peer claiming it has content but not replying, are still possible. Or wasting CPU and bandwidth at a receiving peer by sending packets where the DATA doesn’t match the HASHes.

- **PPSP.SEC.REQ-5**: The Merkle tree hashing scheme allows a receiving peer to detect a malicious or faulty sender, which it can subsequently ignore. Spreading this knowledge to other peers such that they know about this bad behavior is hearsay.

- **PPSP.SEC.REQ-6**: A risk in peer-to-peer streaming systems is that malicious peers launch an Eclipse [ECLIPSE] attack on the initial injectors of the content (in particular in live streaming). The attack tries to let the injector upload to just malicious peers which then do not forward the content to others, thus stopping the distribution. An Eclipse attack could also be launched on an individual peer. Letting these injectors only use trusted trackers that provide true random samples of the population or using a secure peer sampling service [PUPPETCAST] can help negate such an attack.

- **PPSP.SEC.REQ-7**: PPSPP supports decentralized tracking via PEX or additional mechanisms such as DHTs [SECDHTS], but self-certification of addresses is needed. Self-certification means for example, that each peer has a public/private key pair [PERMIDS] and creates self-
certified address changes that include the swarm ID and a timestamp, which are then exchanged among peers or stored in DHTs. See also discussion of PPSP.PP.REQ-3 above. Content distribution can continue as long as there are peers that have it available.

- PPSP.SEC.REQ-8: The verification of data via hashes obtained from a trusted source is well-established in the BitTorrent protocol [BITTORRENT]. The proposed Merkle tree scheme is a secure extension of this idea. Self-certification and not using hearsay are other lessons learned from existing distributed systems.

- PPSP.SEC.REQ-9: PPSPP has built-in content integrity protection via self-certified naming of content, see SEC.REQ-5 and Sec. 3.5.1.

7. Security Considerations

As any other network protocol, the PPSPP faces a common set of security challenges. An implementation must consider the possibility of buffer overruns, DoS attacks and manipulation (i.e. reflection attacks). Any guarantee of privacy seems unlikely, as the user is exposing its IP address to the peers. A probable exception is the case of the user being hidden behind a public NAT or proxy.

8. Extensibility

8.1. 32 bit vs 64 bit

While in principle the protocol supports bigger (>1TB) files, all the mentioned counters are 32-bit. It is an optimization, as using 64-bit numbers on-wire may cost ~2% practical overhead. The 64-bit version of every message has typeid of 64+t, e.g. typeid 68 for 64-bit hash message:

```
44 000000000000000E 01234567890ABCDEF1234567890ABCDEF1234567
```

8.2. IPv6

IPv6 versions of PEX messages use the same 64+t shift as just mentioned.

8.3. Congestion Control Algorithms

Congestion control algorithm is left to the implementation and may even vary from peer to peer. Congestion control is entirely implemented by the sending peer, the receiver only provides clues, such as hints, acknowledgments and timestamps. In general, it is
expected that servers would use TCP-like congestion control schemes such as classic AIMD or CUBIC [CUBIC]. End-user peers are expected to use weaker-than-TCP (least than best effort) congestion control, such as [LEDBAT] to minimize seeding counter-incentives.

8.4. Chunk Picking Algorithms

Chunk (or piece) picking entirely depends on the receiving peer. The sender peer is made aware of preferred chunks by the means of HINT messages. In some scenarios it may be beneficial to allow the sender to ignore those hints and send unrequested data.

The chunk picking algorithm is external to the PPSPP protocol and will generally be a pluggable policy that uses the mechanisms provided by PPSPP. The algorithm will handle the choices made by the user consuming the content, such as seeking, switching audio tracks or subtitles.

8.5. Reciprocity Algorithms

Reciprocity algorithms are the sole responsibility of the sender peer. Reciprocal intentions of the sender are not manifested by separate messages (as BitTorrent’s CHOKE/UNCHOKE), as it does not guarantee anything anyway (the "snubbing" syndrome).

8.6. Different crypto/hashing schemes

Once a flavor of PPSPP will need to use a different crypto scheme (e.g., SHA-256), a message should be allocated for that. As the root hash is supplied in the handshake message, the crypto scheme in use will be known from the very beginning. As the root hash is the content's identifier, different schemes of crypto cannot be mixed in the same swarm; different swarms may distribute the same content using different crypto.

9. Rationale

Historically, the Internet was based on end-to-end unicast and, considering the failure of multicast, was addressed by different technologies, which ultimately boiled down to maintaining and coordinating distributed replicas. On one hand, downloading from a nearby well-provisioned replica is somewhat faster and/or cheaper; on the other hand, it requires to coordinate multiple parties (the data source, mirrors/CDN sites/peers, consumers). As the Internet
progresses to richer and richer content, the overhead of peer.replica
coordination becomes dwarfed by the mass of the download itself.
Thus, the niche for multiparty transfers expands. Still, current,
relevant technologies are tightly coupled to a single use case or
even infrastructure of a particular corporation. The mission of our
project is to create a generic content-centric multiparty transport
protocol to allow seamless, effortless data dissemination on the Net.

TABLE 1. Use cases.
<table>
<thead>
<tr>
<th>mirror-based</th>
<th>peer-assisted</th>
<th>peer-to-peer</th>
</tr>
</thead>
<tbody>
<tr>
<td>data</td>
<td>SunSITE</td>
<td>CacheLogic</td>
</tr>
<tr>
<td></td>
<td></td>
<td>VelociX</td>
</tr>
<tr>
<td>VoD</td>
<td>YouTube</td>
<td>Azureus(+seedboxes)</td>
</tr>
<tr>
<td>live</td>
<td>Akamai Str.</td>
<td>Octoshape, Joost</td>
</tr>
</tbody>
</table>

The protocol must be designed for maximum genericity, thus focusing
on the very core of the mission, contain no magic constants and no
hardwired policies. Effectively, it is a set of messages allowing to
securely retrieve data from whatever source available, in parallel.
Ideally, the protocol must be able to run over IP as an independent
transport protocol. Practically, it must run over UDP and TCP.

9.1. Design Goals

The technical focus of the PPSPP protocol is to find the simplest
solution involving the minimum set of primitives, still being
sufficient to implement all the targeted usecases (see Table 1),
suitable for use in general-purpose software and hardware (i.e. a web
browser or a set-top box). The five design goals for the protocol
are:

1. Embeddable kernel-ready protocol.
2. Embrace real-time streaming, in- and out-of-order download.
3. Have short warm-up times.
4. Traverse NATs transparently.
5. Be extensible, allow for multitude of implementation over
diverse mediums, allow for drop-in pluggability.

The objectives are referenced as (1)-(5).

The goal of embedding (1) means that the protocol must be ready to
function as a regular transport protocol inside a set-top box, mobile
device, a browser and/or in the kernel space. Thus, the protocol must
have light footprint, preferably less than TCP, in spite of the
necessity to support numerous ongoing connections as well as to
constantly probe the network for new possibilities. The practical
overhead for TCP is estimated at 10KB per connection [HTTP1MLN]. We aim at <1KB per peer connected. Also, the amount of code necessary to make a basic implementation must be limited to 10KLoC of C. Otherwise, besides the resource considerations, maintaining and auditing the code might become prohibitively expensive.

The support for all three basic usecases of real-time streaming, in-order download and out-of-order download (2) is necessary for the manifested goal of THE multiparty transport protocol as no single usecase dominates over the others.

The objective of short warm-up times (3) is the matter of end-user experience; the playback must start as soon as possible. Thus any unnecessary initialization roundtrips and warm-up cycles must be eliminated from the transport layer.

Transparent NAT traversal (4) is absolutely necessary as at least 60% of today’s users are hidden behind NATs. NATs severely affect connection patterns in P2P networks thus impacting performance and fairness [MOLNAT,LUCNAT].

The protocol must define a common message set (5) to be used by implementations; it must not hardwire any magic constants, algorithms or schemes beyond that. For example, an implementation is free to use its own congestion control, connection rotation or reciprocity algorithms. Still, the protocol must enable such algorithms by supplying sufficient information. For example, trackerless peer discovery needs peer exchange messages, scavenger congestion control may need timestamped acknowledgments, etc.

9.2. Not TCP

To large extent, PPSPP’s design is defined by the cornerstone decision to get rid of TCP and not to reinvent any TCP-like transports on top of UDP or otherwise. The requirements (1), (4), (5) make TCP a bad choice due to its high per-connection footprint, complex and less reliable NAT traversal and fixed predefined congestion control algorithms. Besides that, an important consideration is that no block of TCP functionality turns out to be useful for the general case of swarming downloads. Namely,

1. in-order delivery is less useful as peer-to-peer protocols often employ out-of-order delivery themselves and in either case out-of-order data can still be stored;
2. reliable delivery/retransmissions are not useful because the same data might be requested from different sources; as in-order delivery is not required, packet losses might be patched up lazily, without stopping the flow of data;
3. flow control is not necessary as the receiver is much less likely to be saturated with the data and even if so, that situation is perfectly detected by the congestion control;
4. TCP congestion control is less useful as custom congestion control is often needed [LEDBAT].

In general, TCP is built and optimized for a different usecase than we have with swarming downloads. The abstraction of a "data pipe" orderly delivering some stream of bytes from one peer to another turned out to be irrelevant. In even more general terms, TCP supports the abstraction of pairwise _conversations_, while we need a content-centric protocol built around the abstraction of a cloud of participants disseminating the same _data_ in any way and order that is convenient to them.

Thus, the choice is to design a protocol that runs on top of unreliable datagrams. Instead of reimplementing TCP, we create a datagram-based protocol, completely dropping the sequential data stream abstraction. Removing unnecessary features of TCP makes it easier both to implement the protocol and to verify it; numerous TCP vulnerabilities were caused by complexity of the protocol’s state machine. Still, we reserve the possibility to run PPSPP on top of TCP or HTTP.

Pursuing the maxim of making things as simple as possible but not simpler, we fit the protocol into the constraints of the transport layer by dropping all the transmission’s technical metadata except for the content’s root hash (compare that to metadata files used in BitTorrent). Elimination of technical metadata is achieved through the use of Merkle [MERKLE,ABMRKL] hash trees, exclusively single-file transfers and other techniques. As a result, a transfer is identified and bootstrapped by its root hash only.

To avoid the usual layering of positive/negative acknowledgment mechanisms we introduce a scale-invariant acknowledgment system (see Sec 4.4). The system allows for aggregation and variable level of detail in requesting, announcing and acknowledging data, serves in-order and out-of-order retrieval with equal ease. Besides the protocol’s footprint, we also aim at lowering the size of a minimal useful interaction. Once a single datagram is received, it must be checked for data integrity, and then either dropped or accepted, consumed and relayed.

9.3. Generic Acknowledgments

Generic acknowledgments came out of the need to simplify the data addressing/requesting/acknowledging mechanics, which tends
to become overly complex and multilayered with the conventional approach. Take the BitTorrent+TCP tandem for example:

1. The basic data unit is a byte of content in a file.
2. BitTorrent’s highest-level unit is a "torrent", physically a byte range resulting from concatenation of content files.
3. A torrent is divided into "pieces", typically about a thousand of them. Pieces are used to communicate progress to other peers. Pieces are also basic data integrity units, as the torrent’s metadata includes a SHA1 hash for every piece.
4. The actual data transfers are requested and made in 16KByte units, named "blocks" or chunks.
5. Still, one layer lower, TCP also operates with bytes and byte offsets which are totally different from the torrent’s bytes and offsets, as TCP considers cumulative byte offsets for all content sent by a connection, be it data, metadata or commands.
6. Finally, another layer lower, IP transfers independent datagrams (typically around 1.5 kilobyte), which TCP then reassembles into continuous streams.

Obviously, such addressing schemes need lots of mappings; from piece number and block to file(s) and offset(s) to TCP sequence numbers to the actual packets and the other way around. Lots of complexity is introduced by mismatch of bounds: packet bounds are different from file, block or hash/piece bounds. The picture is typical for a codebase which was historically layered.

To simplify this aspect, we employ a generic content addressing scheme based on binary intervals, or "bins" for short.

Acknowledgements

Arno Bakker and Victor Grishchenko are partially supported by the P2P-Next project (http://www.p2p-next.org/), a research project supported by the European Community under its 7th Framework Programme (grant agreement no. 216217). 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 P2P-Next project or the European Commission.

The PPSPP protocol was designed by Victor Grishchenko at Technische Universität Delft. The authors would like to thank the following people for their contributions to this draft: Mihai Capota, Raul Jiminez, Flutra Osmani, Riccardo Petrocco, Johan Pouwelse, and Raynor Vliegendhart.

A. Bakker
Expires August 2, 2012
References


http://datatracker.ietf.org/wg/ppsp/charter/


http://doi.acm.org/10.1145/1877891.1877898


Authors’ addresses

A. Bakker

A. Bakker  Expires August 2, 2012
Appendix A. Revision History

-00 2011-12-19 Initial version.
-01 2010-10-27 Minor text revision:
  * Changed heading to "A. Bakker"
  * Changed title to *Peer* Protocol, and abbreviation PPSPP.
  * Replaced swift with PPSPP.
  * Removed Sec. 6.4. "HTTP (as PPSP)".
  * Renamed Sec. 8.4. to "Chunk Picking Algorithms".
  * Resolved Ticket #3: Removed sentence about random set of peers.
  * Resolved Ticket #6: Added clarification to "Chunk Picking Algorithms" section.
  * Resolved Ticket #11: Added Sec. 3.12 on Storage Independence
  * Resolved Ticket #13: Added clarification to "Automatic Size Detection" section.
  * Resolved Ticket #15: Operation section now states it shows example behaviour for a specific set of policies and schemes.
  * Resolved Ticket #30: Explained why multiple HINTs in one datagram.
  * Resolved Ticket #31: Renamed PEX_ADD message to PEX_RES.
  * Resolved Ticket #32: Renamed Sec 3.8. to "Keep Alive Signaling", and updated explanation.
  * Resolved Ticket #33: Explained NAT hole punching via only PPSPP messages.
  * Resolved Ticket #34: Added section about limited overhead of the Merkle hash tree scheme.
Abstract

Peer-to-Peer (P2P for short) streaming systems show more and more popularity in current Internet with proprietary protocols. This document identifies problems of the proprietary protocols, proposes a Peer to Peer Streaming Protocol (PPSP) including tracker and peer signaling components, and discusses the scope, uses cases and requirements of PPSP.
Status of this Memo

This Internet-Draft is submitted in full conformance with the provisions of BCP 78 and BCP 79.

Internet-Drafts are working documents of the Internet Engineering Task Force (IETF), its areas, and its working groups. Note that other groups may also distribute working documents as Internet-Drafts.

Internet-Drafts are draft documents valid for a maximum of six months and may be updated, replaced, or obsoleted by other documents at any time. It is inappropriate to use Internet-Drafts as reference material or to cite them other than as "work in progress."

The list of current Internet-Drafts can be accessed at http://www.ietf.org/ietf/1id-abstracts.txt

The list of Internet-Draft Shadow Directories can be accessed at http://www.ietf.org/shadow.html

This Internet-Draft will expire on August 27, 2012.

Copyright Notice

Copyright (c) 2012 IETF Trust and the persons identified as the document authors. All rights reserved.

This document is subject to BCP 78 and the IETF Trust’s Legal Provisions Relating to IETF Documents (http://trustee.ietf.org/license-info) in effect on the date of publication of this document. Please review these documents carefully, as they describe your rights and restrictions with respect to this document. Code Components extracted from this document must include Simplified BSD License text as described in Section 4.e of the Trust Legal Provisions and are provided without warranty as described in the Simplified BSD License.
Table of Contents

1. Introduction ................................................ 4
2. Terminology and concepts .................................... 5
3. Problem statement ........................................... 7
   3.1. Traffic issue and difficulties for ISPs in deploying P2P caches ................................................................. 7
   3.2. Efficiency issue and difficulties in building open streaming delivery infrastructure ............................................. 7
   3.3. Extended applicability issue and difficulties in mobile and wireless environment ......................................................... 8
4. PPSP: Standard peer to peer streaming protocols .......... 10
5. Use cases of PPSP ........................................... 12
   5.1. Worldwide provision of live/VoD streaming ................ 12
   5.2. PPSP supporting cross-screen streaming in heterogeneous environment ................................................................. 14
   5.3. Cache service supporting P2P streaming .................. 15
6. Security Considerations ..................................... 17
7. Requirements of PPSP ........................................ 18
   7.1. Basic Requirements ..................................... 18
   7.2. PPSP Tracker Protocol Requirements ..................... 19
   7.3. PPSP Peer Protocol Requirements ........................ 21
   7.4. Security Requirements .................................. 22
8. IANA Considerations ......................................... 24
9. Acknowledgments ............................................. 25
10. Informative References ..................................... 26
1. Introduction

Streaming traffic is among the largest and fastest growing traffic on the Internet [Cisco], where peer-to-peer (P2P) streaming contribute a lot. With the advantage of high scalability and fault tolerance against single point of failure, P2P streaming applications are able to distribute large-scale, live and VoD streaming programs to millions of audience with only a handful of servers.

What’s more, along with the new players like CDN providers joining in the effort of using P2P technologies in distributing their serving streaming content, there are more and more various players in P2P streaming ecosystem.

Given the increasing integration of P2P streaming into the global content delivery infrastructure, the lack of an open, standard P2P streaming signaling protocol suite becomes a major missing component in the protocol stack. Almost all of existing systems use their proprietary protocols. Multiple, similar but proprietary protocols result in repetitious development efforts for new systems, and the lock-in effects lead to substantial difficulties in their integration with other players like CDN. For example, in the enhancement of existing caches and CDN systems to support P2P streaming, proprietary protocols may increase the complexity of the interaction with different P2P streaming applications.

In this document we propose an open P2P Streaming Protocol, which is defined as PPSP, to standardize signaling operations on two important components, peer and tracker in P2P streaming systems for information exchange. Note that using PPSP would not hurt current P2P streaming vendors: Firstly, the openness of signaling interaction would make it easy to integrate them with some infrastructural components like ISP’s caches or CDNs for better user experience, say, smaller delay of the play. Secondly, different applications could use the same PPSP for signaling, but implement system specific mechanisms on top of that. That is to say, different P2P streaming systems compete on "on top" things, like scheduling algorithms, which is independent of the proposed protocols.
2. Terminology and concepts

Chunk: A chunk is a basic unit of data block organized in P2P streaming for storage, scheduling, advertisement and exchange among peers [VoD]. A chunk size varies from several KB to several MB in different systems. In case of MB size chunk scenario, a sub-chunk structure named piece is often defined to fit in a single transmitted packet. A streaming system may use different granularities for different usage, e.g., using chunks during data exchange, and using a larger unit such as a set of chunks during advertisement.

Content Distribution Network (CDN): A CDN node refers to a network entity that is deployed in the network (e.g., at the network edge or data centers) to store content provided by the original servers, and serves content to the clients located nearby topologically.

Client: A client refers to the service requester in client/server computing paradigm. In this draft a client refers to a participant in a P2P streaming system that only receives streaming content. In some cases the node is not eligible to be a peer without enough computing and storage capability acting as a client. It can be viewed as a specific kind of peer.

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

P2P cache: A P2P cache refers to a network entity that caches P2P traffic in the network, and either transparently or explicitly as a peer distributes content to other peers.

Peer: A peer refers to a participant in a P2P streaming system that not only receives streaming content, but also stores and uploads streaming content to other participants.

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

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.
Tracker: A tracker refers to a directory server which maintains a list of peers which participate in a specific video channel or in the distribution of a streaming file, and answers queries from peers for peer lists. The tracker is a logical component which can be centralized or distributed.

Video-on-demand (VoD): It refers to a scenario where different clients may watch different parts of the same recorded media with downloaded content.

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 either tracker or other peers to know which peers have the required streaming content.

Peer ID: An identifier of a peer such that other peers or tracker can refer the ID for the peer.

Swarm ID: An identifier of a swarm containing a group of peers sharing a same streaming content.

Chunk ID: An identifier of a chunk in a streaming content.
3. Problem statement

The problems imposed by proprietary protocols for P2P streaming applications are listed as follows.

3.1. Traffic issue and difficulties for ISPs in deploying P2P caches

Facing with many P2P streaming applications, ISPs are witnessing a big traffic tension on their backbone and inter-networking points. P2P caches are used for ISPs to reduce the traffic by dynamically storing the frequently accessed streaming content (in chunk or in file granularity).

However, unlike the Web where all kinds of the infrastructure devices have been already equipped with standard HTTP protocol, cache systems have to build a matching library to identify different P2P streaming protocols firstly. Multiple ever changing proprietary protocols require the cache system updating its matching library constantly. This increases the operator's cost dramatically.

With PPSP, P2P caches needn't learn new proprietary protocols any longer, which would reduce the ISP workload much.

3.2. Efficiency issue and difficulties in building open streaming delivery infrastructure

Another problem for P2P streaming applications is the efficiency issue. P2P streaming is often criticized by longer delays (e.g., startup delay, seek delay and channel switch delay). Hybrid CDN/P2P is a good means to solve this problem for operators [Hybrid CDN/P2P].

In such design, CDN takes two roles: one is for media streaming server and the other is for P2P tracker. Consider a CDN vendor serving for various P2P streaming, similar to the P2P cache issue in Section 3.1, proprietary P2P streaming protocols introduce interaction complexity between the peer and tracker, and increase deployment cost of CDN nodes.

With PPSP, CDN nodes acting as the P2P tracker can be designed to inter-operate with other devices by only standard protocols, reducing the case by case negotiation. On the other side, the interface between edged CDN nodes and user peers could be either via something like traditional HTTP, or via PPSP peer protocol.
3.3. Extended applicability issue and difficulties in mobile and wireless environment

Mobility and wireless are becoming increasingly important in today’s Internet, where streaming service is a major usage. In Korea the number of mobile TV subscriber has reached seventeen million, accounting for one third of the mobile subscribers. There are multiple prior studies exploring P2P streaming in mobile and wireless networks [Mobile Streaming1] [Mobile Streaming2].

However it’s difficult to copy current P2P streaming protocols (even we suppose we can re-use the proprietary ones) in mobile and wireless networks. Current protocols are designed mainly for fixed Internet. Although smart handsets are more eligible to be peers with much higher bandwidth and CPU frequency, larger storage and memory than before, peer selection becomes more challenging which needs more information to exchange during the tracker/peer and peer/peer communications:

First, the connections are unsteady, lower rate, and costly in terms of energy consumption and transmission (esp. in uplink). The trackers and peers may need more information like packet loss rate, peer battery status and processing capability for peer selection.

Second, current practices often use a "bitmap" message to exchange chunk availability among peers/trackers. The message is often of some kilobytes size and exchanged relatively frequently, say, some seconds. In a scarce bandwidth resource environment, a reasonable optimization is to reduce the message size, which may require alternative methods for expressing and distributing bitmap information.

Third, when a peer is moving and if the IP address changes, the ongoing connection and transmission between peers may be affected. Such information should be reported in time.

Fourth, for a resource constraint peer like mobile handsets or set-top boxes (STB), the limited resource and the requirements for installing various applications form big conflicts. On one side, the limited CPU, storage and memory often limit the total number of concurrent threads and processes. One the other side, the proprietary protocols require the user to install many different applications for different usage, for instance, some applications have rich resources on TV series or movies while others may offer rich broadcasting sports program. What’s worse, for many P2P applications, even they are not used by the users right now, the background program may be invoked to facilitate other peers for free data delivery assistance. That is to say, there will be multiple background programs running at
the same time. But it may be difficult to invoke multiple programs in such a resource constraint peer. PPSP should investigate these factors and help to reduce the resource consumption in a converged network.
4. PPSP: Standard peer to peer streaming protocols

The objective of the PPSP working group is to design a unified peer-to-peer streaming protocol (PPSP) to address the problems discussed in the preceding sections.

There are basically two kinds of P2P streaming systems, pull-based and push-based.

In pull-based P2P streaming systems, a centralized tracker or distributed trackers maintains information about which peers are in which swarms and answers the peers' query on such information with a peer-list. After receiving the message, the peer can connect with the candidates in a swarm, exchange its content availability in its memory or storage (depending on it is real-time or VoD streaming) with other peers and then retrieve the wanted streaming data. The swarm is a mesh topology. Most of the current practices belong to this genre. The advantages of pull-based mode are its robustness to the peer churn and acceptable latency for a smooth play. Most commercial systems either live streaming and VoD use this mode.

In push-based P2P streaming systems, there is a head node maintaining the topology, e.g., a tree. The head node acts similar to a tracker. The peers in this topology share the same interest on content. The signaling and data distribution are both based on this topology. For one program or video file, the peer queries the head node by offline or pre-set head node address information for its location to join and the head node replies with a peer-list (potentially in a recommended order). After receiving this peer-list, the peer can connect with the candidates for being a node in certain place of the topology and receive the data along this topology without the need of exchanging content availability with its siblings, as done in pull-based mode. In this sense the head node is acting as the tracker in the pull-based mode. The push mode has the advantages of lower latency but the topology is fragile to the peer churn and is hard to deal with the VoD scenario. This makes it less robust in practical running. Few commercially deployed systems use this mode.

PPSP is targeted to standardize signaling protocols for tracker-based architectures feasible for both modes above that support either live or offline streaming.

The PPSP design includes a protocol for signaling between trackers and peers (the PPSP "tracker protocol") and a signaling protocol for communication among the peers (the PPSP "peer protocol") as shown in
The two protocols enable peers to receive streaming data within the time constraints required by specific content items. The tracker protocol handles the initial and periodic exchange of meta information between trackers and peers, such as peer-list and content information. The peer protocol controls the advertising and exchange of media data between the peers.

Note that in the pull mode, both tracker protocol and peer protocol can be used; while in the push mode, only tracker protocol is used.
5. Use cases of PPSP

5.1. Worldwide provision of live/VoD streaming

The content provider can easily expand the broadcasting/VoD scale to utilize the cooperative content providers’ distribution networks or third party CDN networks with PPSP.

Figure 2 shows the case that provider A broadcasts the program with the help of provider B and C for a wider coverage. Without PPSP, when users in B or C’s domain (outside A’s main serving zone) requests A’s programs, the returned peer-list may include few local peers, which may decrease the user viewing experience. With PPSP more local resources from cooperative vendors may be utilized. The content providers often deploy in-network peers called super-nodes (SN for short) with better stability and higher storage and bandwidth for better QoS. With tracker protocol, vendor A’s tracker can returns user request with vendor B and vendor C’s SNs in the peer-list. Users in B and C’s domain can exchange data (availability) with these SNs using peer protocol for better QoS. In this way vendor B and vendor C’s SNs resources are shared with vendor A and vendor A expands its serving scale with acceptable QoS.
Figure 2 Cooperative Vendors Interaction

Figure 3 is similar to Figure 2 except that the intermediate SNs are replaced by 3rd party CDN surrogates with PPSP. The P2P streaming vendors A and B can rent CDN surrogates to provide higher QoS for VIP users. The CDN nodes talk with the different vendors (including the peers inside) with the uniform protocols. For users who use browser equipped with HTTP, scalable streaming is also achieved. The internal interaction of CDN nodes can be executed by either existing protocol or peer protocol. The latter is used when building a new CDN system supporting streaming applications with lower cost.
5.2. PPSP supporting cross-screen streaming in heterogeneous environment

In this scenario PC, STB/TV and mobile terminals from both fixed network and mobile/wireless network share the streaming content. With PPSP, peers can identify the types of access networks, average load, peer abilities and get to know what content other peers have (potentially with the conversion of the content availability expression in different networks) even in different network conditions as shown in Figure 4.

These information will play an important role on selecting suitable peers, e.g., a PC or STB is more likely to be selected to provide stable content for mobile nodes; a mobile peer within a high-load base station is unlikely to be selected, which may lead to higher load on the base station.
5.3. Cache service supporting P2P streaming

In Figure 5, when peers request the P2P streaming data, the cache nodes intercept the requests and ask for the frequently visited content (or part of) on behalf of the user peers. To do this, it requests peer-list to the tracker and the tracker replies with (outward) peers in the peer-list. After the cache nodes exchange data with these peers, it can report what it cache to the tracker like a normal peer and serve other requesting peers inside. This operation greatly decreases the inter-network traffic and increase user experience in P2P streaming services for an ISP.

The cache nodes needn’t update their library when new applications supporting PPSP are introduced, which enable the cache nodes spend less cost to support more applications.
Figure 5 Cache Service Supporting Streaming with PPSP
6. Security Considerations

This document discusses the problem statement around Peer-to-Peer streaming protocols without specifying the protocols. The protocol specification is deferred to other documents under development in the PPSP working group. However we believe it is important for the reader to understand areas of security caused by the P2P nature of the proposed solution. The main issue is the usage of un-trusted entities (peers) for service provisioning.

Malicious peers may, for example:

- Issue denial of service (DOS) attacks to the trackers by sending large amount of requests with the tracker protocol;

- Issue fake information on behalf of other peers;

- Issue fake information about available content;

- Issue fake information about chunk availability;

Malicious peers/trackers may, for example:

- Issue reply instead of the regular tracker (man in the middle attack).

The PPSP protocol specifications, e.g., the tracker protocol and the peer protocol, will document the expected threats and how they will be mitigated for each protocol, but also considerations on threats and mitigations when combining both protocols in an application. This will include privacy of the users, protection of the content distribution, but not protection of the content by Digital Rights Management (DRM).
7. Requirements of PPSP

This section enumerates the requirements for the PPSP, which should be considered when designing PPSP.

7.1. Basic Requirements

PPSP.REQ-1: The tracker and the peer protocols SHOULD be as similar as possible, in terms of design, message formats and flows.

It is desirable that the peer protocol would be an extension to the tracker protocol by adding a few message types, or vice versa.

PPSP.REQ-2: The tracker protocol and the peer protocol SHOULD enable peers to receive streaming content within the required time constraints, i.e., fulfill streaming feature.

PPSP.REQ-3: Each peer MUST have a unique ID (i.e. peer ID) in a swarm.

It’s a basic requirement for a peer to be uniquely identified in a swarm that other peers or tracker can refer to the peer by ID.

PPSP.REQ-4: The streaming content MUST be uniquely identified by a swarm ID.

A swarm refers to a group of peers sharing the same streaming content. A swarm ID uniquely identifies a swarm. The swarm ID can be used in two cases: 1) a peer requests the tracker for the peer list indexed by a swarm ID; 2) a peer tells the tracker about the swarms it belongs to.

PPSP.REQ-5: The streaming content MUST allow to be partitioned into chunks.

A key characteristic of P2P streaming system is allowing the data fetching from different peers concurrently. Therefore, the whole streaming content must allow to be partitioned into small pieces or chunks for transmission between peers.

PPSP.REQ-6: Each chunk MUST have a unique ID (i.e. chunk ID) in the swarm.

Each chunk must have a unique ID in the swarm such as the peer can understand which chunks are stored in which peers and which chunks
are requested by other peers. An example for generating the chunk ID is the buffer map approach [I-D.ietf-ppsp-survey].

PPSP.REQ-7: The tracker protocol and peer protocol are recommended to be carried over TCP (or UDP, when delivery requirements cannot be met by TCP).

PPSP.REQ-8: The tracker and peer protocol together MUST facilitate acceptable QoS (e.g. low startup delay, low channel/content switching time and minimal end-to-end delay) for both on-demand and live streaming, even for very popular content. The tracker and peer protocol do not include the algorithm required for scalable streaming. However, the tracker and peer protocol SHALL NOT restrict or place limits on any such algorithm.

There are basic QoS requirements for streaming system. Setup time to receive a new streaming channel or to switch between channels should be reasonable small. End to end delay (time between content generation, e.g. camera and content consumption, e.g. user side monitor) will become critical in case of live streaming. Especially in provisioning of sports events, end to end delay of 1 minute and more are not acceptable.

For instance, the tracker and peer protocols can support carrying QoS related parameters (e.g. video quality, delay requirements) together with the priorities of these parameters, and QoS situation (e.g., performance, available uplink bandwidth) of content providing peers.

There are also some other possible mechanisms, e.g. addition of super peers, in-network storage, request of alternative peer addresses, and the usage of QoS information for an advanced peer selection.

7.2. PPSP Tracker Protocol Requirements

The tracker protocol defines how the peers report and request information to/from the tracker and how the tracker replies to the requests. The tracker discovery and the possible communication between trackers are out of the scope of tracker protocol.

PPSP.TP.REQ-1: The tracker MUST implement the tracker protocol for receiving queries and periodical peer status reports/updates from the peers and for sending the corresponding replies.
PPSP.TP.REQ-2: The peer MUST implement the tracker protocol for sending queries and periodical peer status reports/updates to the tracker and receiving the corresponding replies.

PPSP.TP.REQ-3: The tracker request message MUST allow the requesting peer to solicit the peer list from the tracker with respect to a specific swarm ID.

The tracker request message may also include the requesting peer’s preference parameter, e.g. preferred number of peers in the peer list, or preferred downloading bandwidth. The track will then be able to select an appropriate set of peers for the requesting peer according to the preference.

PPSP.TP.REQ-4: The tracker reply message MUST allow the tracker to offer the peer list to the requesting peer with respect of a specific swarm ID.

PPSP.TP.REQ-5: The tracker SHOULD support generating the peer list with the help of traffic optimization services, e.g. ALTO [I-D.ietf-alto-protocol].

PPSP.TP.REQ-6: The peer status report/update MUST have the ability to inform the tracker about the peer’s activity in the swarm.

PPSP.TP.REQ-7: The chunk availability information of the peer SHOULD be reported to tracker when tracker needs such information to steer peer selection. The chunk information MUST at least contain the chunk ID.

PPSP.TP.REQ-8: The chunk availability information between peer and tracker MUST be as expressed as compactly as possible.

The peers may report CHUNK AVAILABILITY DIGEST information (i.e., compact expression of chunk availability) to the tracker when possible to decrease the bandwidth consumption for messages in bandwidth constraint environment like mobile network. For example, if a peer has a bitmap like 1111111...1 (100 continuous 1)xxx..., the 100 continuous "1" can be expressed by one byte with seven bits representing 100 and one bit representing "1". In this example, 100-8=92 bits are saved. Considering the frequency of exchange of CHUNK AVAILABILITY and the fact that many bitmaps have quite a long length of continuous "1" or "0", such compression makes sense.

PPSP.TP.REQ-9: The status of the peer SHOULD be reported to the tracker when tracker needs such information to steer peer selection.
For example, peer status can be online time, physical link status including DSL/WIFI/etc, battery status, processing capability, and other capabilities of the peer. Therefore, the tracker is able to select better candidate peers for streaming.

7.3. PPSP Peer Protocol Requirements

The peer protocol defines how the peers advertise streaming content availability and exchange status with each other. The peer protocol also defines the requests and responses of the chunks among the peers. The first task for this WG will be to decide which signaling and media transfer protocols will be used. The WG will consider existing protocols and, if needed, identify potential extensions to these protocols.

PPSP.PP.REQ-1: The streaming content availability request message MUST allow the peer to solicit the chunk information from other peers in the peer list. The chunk information MUST at least contain the chunk ID. This chunk availability information MUST NOT be passed on to other peer, unless validated (e.g. prevent hearsay and DoS).

PPSP.PP.REQ-2: The streaming content availability reply message MUST allow the peer to offer the information of the chunks in its content buffer. The chunk information MUST at least contain the chunk ID.

PPSP.PP.REQ-3: The streaming content availability request message SHOULD allow the peer to solicit an additional list of peers to that received from the tracker – with the same swarm ID. The reply message MUST contain swarm-membership information of the peers that have explicitly indicated they are part of the swarm, verifiable by the receiver. This additional list of peers MUST only contain peers which have been checked to be valid and online recently (e.g. prevent hearsay and DoS).

It is possible that a peer may need additional peers for certain streaming content. Therefore, it is allowed that the peer communicates with the peers in the current peer list to obtain an additional list of peers in the same swarm.

PPSP.PP.REQ-4: Streaming content availability update message among the peers MUST be supported by peer protocol. In the push based model, where peers advocate their own chunk availability proactively, the content availability request message described in PP.REQ-1 is not needed. The peer protocol MUST implement either pull-based, push-based or both.
Due to the dynamic change of the buffered streaming content in each peer and the frequent join/leave of peers in the swarm, the streaming content availability among a peer’s neighbors (i.e. the peers known to a peer by getting the peer lists from either tracker or peers) always changes and thus requires being updated on time. This update should be done at least on demand. For example, when a peer requires finding more peers with certain chunks, it sends a message to some other peers in the swarm for streaming content availability update. Alternatively, each peer in the swarm can advertise its streaming content availability to some other peers periodically. However, the detailed mechanisms for this update such as how far to spread such update message, how often to send this update message, etc should leave to peer algorithms, rather than protocol concerns.

PPSP.PP.REQ-5: The chunk availability information between peers MUST be as expressed as compactly as possible.

In PP.REQ-1/2/4, the peers may exchange CHUNK AVAILABILITY DIGEST information (i.e. compact expression of chunk availability) to with other peers when possible to decrease the bandwidth consumption for messages in bandwidth constraint environment like mobile network.

PPSP.PP.REQ-6: The peer status report/update SHOULD be advertised among the peers to reflect the status of the peer.

Peer status information should be advertised among the peers via the peer status report/update message. For example, peer status can be online time, physical link status including DSL/WIFI/etc, battery status, processing capability, and other capabilities of the peer. With this information, a peer can select more appropriate peers for streaming.

PPSP.PP.REQ-7: The peers MUST implement the peer protocol for chunk data (not availability information) requests and responses among the peers before the streaming content is transmitted.

7.4. Security Requirements

PPSP.SEC.REQ-1: PPSP MUST support closed swarms, where the peers are authenticated.

This ensures that only the authenticated users can access the original media in the P2P streaming system. This can be achieved by security mechanisms such as user authentication and/or key management scheme.
PPSP.SEC.REQ-2: Confidentiality of the streaming content in PPSP SHOULD be supported and the corresponding key management scheme SHOULD scale well in P2P streaming system.

PPSP.SEC.REQ-3: PPSP MUST provide an option to encrypt the data exchange among the PPSP entities.

PPSP.SEC.REQ-4: PPSP MUST have mechanisms to limit potential damage caused by malfunctioning and badly behaving peers in the P2P streaming system.

Such an attack will degrade the quality of the rendered media at the receiver. For example, in a P2P live video streaming system a polluter can introduce corrupted chunks. Each receiver integrates into its playback stream the polluted chunks it receives from its other neighbors. Since the peers forwards chunks to other peers, the polluted content can potentially spread through much of the P2P streaming network.

PPSP.SEC.REQ-5: PPSP SHOULD support identifying badly behaving peers, and exclude or reject them from the P2P streaming system.

PPSP.SEC.REQ-6: PPSP MUST prevent peers from DoS attacks which will exhaust the P2P streaming system’s available resource.

Given the prevalence of DoS attacks in the Internet, it is important to realize that a similar threat could exist in a large-scale streaming system where attackers are capable of consuming a lot of resources with just a small amount of effort.

PPSP.SEC.REQ-7: PPSP SHOULD be robust, i.e., when centralized tracker fails the P2P streaming system SHOULD still work by supporting distributed trackers.

PPSP.SEC.REQ-8: Existing P2P security mechanisms SHOULD be re-used as much as possible in PPSP, to avoid developing new security mechanisms.

PPSP.SEC.REQ-9: Integrity of the streaming content in PPSP MUST be supported to provide a peer with the possibility to identify inauthentic media content (undesirable modified by other entities rather than its genuine source). The corresponding checksum distribution and verification scheme SHOULD scale well in P2P streaming system and be robust against distrustful trackers/peers.
8. IANA Considerations

This document has no actions for IANA.
9. Acknowledgments

Thank you to J.Seng for contribution to many sections of this draft. Thank you to C. Williams and L. Xiao for contributions to PPSP requirements section.

We would like to acknowledge the following people who provided review, feedback and suggestions to this document: M. Stiemerling; C. Schmidt; D. Bryan; E. Marocco; V. Gurbani; R. Even; H. Zhang; V. Pasual; D. Zhang; J. Lei; Y.Gu; H.Song; X.Jiang; J.Seedorf; D.Saumitra; A.Rahman; L.Deng; J.Pouwelse; A.Bakker and W.Eddy.

This document was prepared using 2-Word-v2.0.template.dot.
10. Informative References


[VoD] Yan Huang et al, Challenges, "Design and Analysis of a Large-scale P2P-VoD System", Sigcomm08.


Authors’ Addresses

Yunfei Zhang
China Mobile Communication Corporation
zhangyunfei@chinamobile.com

NingZong
Huawei Technologies Co., Ltd.
zongning@huawei.com

Gonzalo Camarillo
Ericsson
Gonzalo.Camarillo@ericsson.com

Richard Yang
Yale University
yry@cs.yale.edu

Victor Pascual
Acme packet
Anabel Segura 10, Madrid 28108, Spain
Vpascual@acmepacket.com