PPSP
Internet-Draft
Intended status: Standards Track
Expires: May 3, 2012

Y. Gu
J. Xia
Huawei
R. Cruz
M. Nunes
IST/INESC-ID/INOV
David A. Bryan
Polycom
J. Taveira
ID/INOV

Oct 31, 2011

Peer Protocol
draft-gu-ppsp-peer-protocol-03

Abstract

This document presents the architecture of the PPSP Peer protocol outlining the functional entities, message flows and message processing instructions, with the respective parameters. The PPSP Peer Protocol proposed in this document extends the capabilities of PPSP to support adaptive and scalable video and 3D video, for Video On Demand (VoD) and Live video services. The protocol messages formal syntax and semantics, methods, and formats are presented for both Binary and HTTP/XML encoded formats.

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). Note that other groups may also distribute working documents as Internet-Drafts. The list of current Internet-Drafts is at http://datatracker.ietf.org/drafts/current/.

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."

This Internet-Draft will expire on May 3, 2012.

Table of Contents

1. Introduction

The P2P Streaming Protocol (PPSP) is composed of two protocols: the PPSP Tracker Protocol and the PPSP Peer Protocol [I-D.ietf-ppsp-problem-statement].

The PPSP architecture requires PPSP peers able to communicate with a Tracker in order to participate in a particular swarm. This centralized Tracker service is used for peer and content registration and location. Content indexes (Media Presentation Descriptions) are also stored in the Tracker system allowing the association of content location information to the active peers in the swarm sharing the content.

The PPSP Tracker Protocol provides communication between Trackers and Peers and outlines how a peer is able to communicate with a tracker in order to exchange meta information about the location of other peers contributing with a specific stream (swarm) the peer interested in, as well as to report streaming status. The Peer can also apply to be a contributor for several streams (swarms), periodically reporting its status to the Tracker, allow it to estimate whether the peer is a competent contributor.

The PPSP Peer protocol outlines how a peer is able to communicate with other peers in order to control the advertising and exchange of media data, directly between peers, for a specific stream (swarm), as described in [I-D.ietf-ppsp-problem-statement].

The process used for media streaming distribution assumes a segment transfer scheme whereby the original content (that can be encoded using adaptive or scalable techniques) is chopped into small segments (and subsegments). For simplicity, in this document the segments (and subsegments) of media are named Chunks. The media streaming process has the following representations:

1. Adaptive - alternate representations with different qualities and bitrates; a single represention 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 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.

The information that should be exchanged between peers using this Peer Protocol includes:

1. ChunkMap indicating which chunks a peer possesses.

2. Required ChunkIDs

3. Peer preferences and status information

4. Signalling and Data Transport protocol negotiation

5. Information that can help improve the performance of PPSP.

In this document, a set of concrete information that needs to be exchanged between peers is introduced, together with the messages to convey such information.

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

This PPSP Peer Protocol proposal presents an early sketch for an extensible protocol that extends the capabilities of PPSP to support adaptive and scalable video.

2. Document Conventions

2.1. Notational Conventions

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].

2.2. Terminology

The draft uses the terms defined in [I-D.ietf-ppsp-problem-statement], [I-D.gu-ppsp-tracker-protocol] and [I-D.cruz-ppsp-http-peer-protocol]. Additionally, This document uses the following acronyms and definitions frequently in itself:

Peer-Peer Messages

The Peer Protocol messages enable each Peer to exchange content availability with other Peers and request other Peers for content.

Tracker-Peer Messages

The Tracker Protocol messages provide communication between Peers and Trackers, by which Peers provide content availability, report streaming status and request candidate Peer lists from Trackers.

Connection Tracker

The Tracker Node to which the PPSP Peer will connect when it wants to join the PPSP system.

Sender Peer

A peer that contains the corresponding chunk files requested by leech peer is the Sender peer. Many peers can contain the content, but only one who is contributing the content to the leech peer can be named as Sender peer.

Leech Peer

A peer that requests the specific media content from other peers. Note that the leech peer can also contribute the downloaded media content (i.e., chunks) even the swarm is not completed, in such case, the leech peer will take on the role of sender peer for downloaded chunks.

Chunk Map

A peer list that indicates which chunks can be available for leech peer to playback smoothly.

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.

Video-on-demand (VoD)

The scenario where all clients are allowed to select and watch video content on demand.

Adaptive Streaming

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

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.

Base Layer

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

Enhancement Layer

Enhancement differential quality level 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.

Continuous Media

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

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.

3. Protocol Overview

3.1. Protocol Architecture

The functional entities involved in the PPSP Peer Protocol are 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 contacts a 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/exchanging chunks for a specific Live media channel or VoD media streaming content, answers queries from peers and collects information on the activity of peers. A simplified network diagram showing this interaction of tracker and peers is depicted in Figure 1.

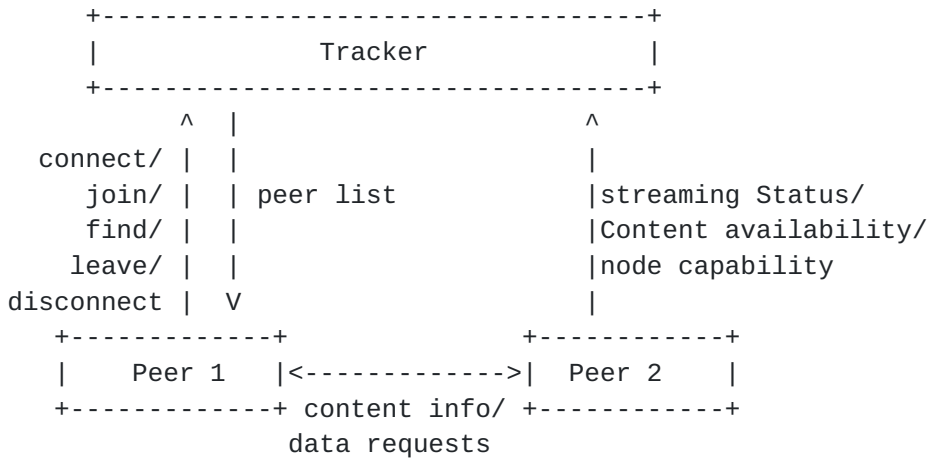

Figure 1: A PPSP streaming process

The signaling between PPSP Peers and trackers is done using a request/reply mechanism as defined in PPSP Tracker protocol [I-D.gu-ppsp-tracker-protocol].

This protocol can be used to connect peers that are sharing real-time streams of video or offline video, segmented in chunks. As for the streams of video, they can correspond to Live or Video on Demand streaming modes.

There are some significant differences between the details of these

scenarios, i.e., Live streaming, VoD and offline video. From a high level perspective the overall structure is quite similar. The optimal signaling flow for the different scenarios could also be different, but it depends on the real situation and on the implementer's choice

This draft defines a PULL based streaming signaling, as mandatory. However, a PUSH based or hybrid streaming signaling can optionally be considered.

For a PULL based Peer Protocol, the steps of signaling for a peer wishing to participate either in a Live streaming or a VoD or offline video is as follows (assuming the leech peer has already obtained from the Tracker a list of peers) and that, in case of traversing a NAT, performed ICE connectivity checks [I-D.li-ppsp-nat-traversal] with candidate peers using PPSP's own authentication method, as described in [I-D.gu-ppsp-tracker-protocol]:

1. The leech peer using PPSP Peer Protocol messages, establishes a connection to at least one of the peers in the Peerlist, based on the known PeerID and Peer IP address.

2. The peer sends request to candidate peers and the request could include one or more of the information described in below:

   * Request for the data availability of the candidate peer;

   * Notify its data availability to the candidate peer;

   * Request for the peer status of the candidate peer;

   * Notify its peer status to the candidate peer;

   * Request for additional peerlist;

   * Transport negotiation, wherein the requesting peer can have two choices:

     + Only support Mandatory Tranport Protocol;

     + Providing a list of supported Transport protocol.

3. Finally, the peers exchange the actual chunks of data, using the mechanism/protocol negotiated in the previous step.

In terms of Data Transport protocol negotiation, the leech peer can either inform the candidate that it supports a Mandatory Tranport Protocol or provides a list of supported Transport protocols. That

there are several options here to negotiate the connection model. The PPSP Peer Protocol may include new mechanisms to negotiate the protocol used to exchange data, or the offer-answer mechanism in SIP [RFC3261] (the IETF protocol for session establishment) along with SDP [RFC4566].

Note also that these mechanisms are not new protocols defined in PPSP, but existing protocols, and would eventually differ between an offline and a Live streaming scenario. Mechanisms such as flow control are handled in the negotiated Data Transport mechanism, not in the Peer Protocol itself.

3.2. Example Call Flow

This is a very high-level example of a session in which a leech peer joins a swarm, and retrieves some data (either via blocks or by streaming). The protocol used is indicated for each transaction. Note that not all of the communication shown in this figure are in scope of Peer Protocol, only those request/response followed by Peer Protocol are in scope.

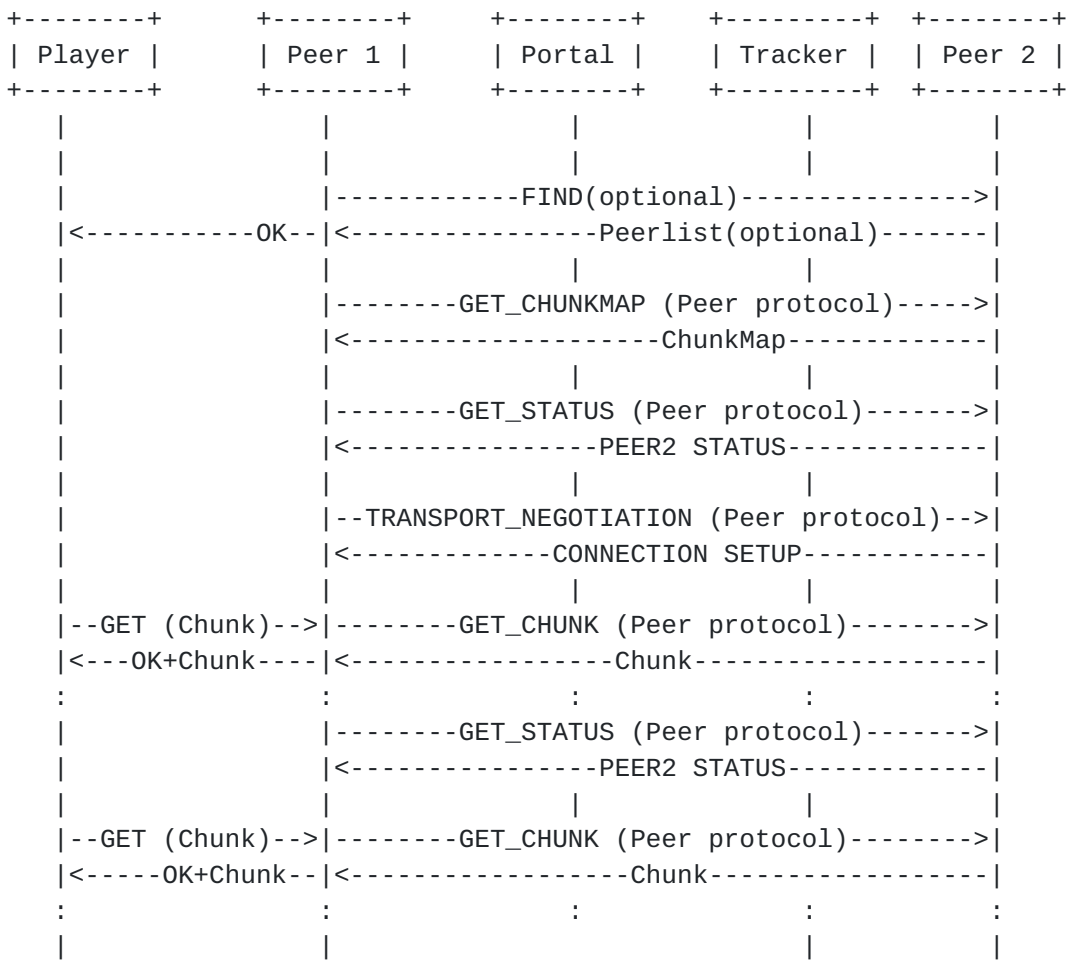

Figure 2: Example Call Flow

3.3. Chunk Scheduling

The goal of chunk trading is receiving the stream smoothly (and with small delay) and to cooperate in the distribution procedure. Peers need to exchange information about their current status to enable scheduling decisions. The information exchanged refers to the state of the peer with respect to the flow, i.e., a map of which chunks are needed by a peer to smoothly playback the stream.

This task means:

1. sending chunk maps to other nodes with the proper timing,

2. receiving chunk maps from other nodes and merging the information in the local buffer map.

3. besides chunk map exchange, the signaling includes Status/ Request/Select primitives used to trade chunks.

The core of the scheduler, not described in this specification, is the algorithm used to choose the chunks to be exchanged and the peers to communicate with.

4. Protocol Architecture

The PPSP Peer Protocol is a request-response protocol. Requests are sent, and responses returned to these requests. A single request generates a single response (neglecting fragmentation of messages).

As shown in example call flow depicted in Figure 2, the Peer protocol only provides signaling messages for obtaining additional peerlist (optionally), query for content availability and negotiation for transfer protocol. Peer protocol may also provide communication for peers to exchange information that can improve system performance.

The encoding for the signaling messages is not yet decided. Two encodings are proposed, a Text-based (HTTP/XML) and a Binary-based, described in Appendixes A and B. The authors will raise more discussion on the encoding, and will move the one that gets rough consensus of the PPSP WG to the draft text. In the Appendixes, some considerations are provided on each encoding based on the Mail List discussions.

The specific PPSP signaling messages are listed as following:

GET_PEERLIST:

The GET_PEERLIST message is sent from a leech peer to one or more remote peers in order for a peer to refresh/update the list of active peers in the swarm.

When receiving the GET_PEERLIST message, and if the message is well formed and accepted, the peer will search for the requested data and will respond to the leech peer with the peer list with PeerIDs (and respective IP Addresses) of sender peers that can provide the specific content.

GET_CHUNKMAP:

The GET_CHUNKMAP message is sent from a leech peer to one or more remote peers in order to receive the map of chunks of a content (of a swarm identified by SwarmID) the other peer presently stores. The chunk map returned by the other peer lists ranges of chunks.

When receiving the GET_CHUNKMAP message, and if the message is well formed and accepted, the peer will search for the requested data and will respond to the leech peer with the map of chunks it currently stores of the specific content.

GET_CHUNK:

The GET_CHUNK message is sent from a leech peer to sender peer in order to request the delivery of media content chunks.

When receiving the GET_CHUNK message, and if the message is well formed and accepted, the peer will search for the requested data and will respond to the leech peer with the specific chunks the leech peer requested.

GET_STATUS:

The GET_ STATUS message is sent from a leech peer to one or more remote peers in order to request the corresponding properties of the sender peers. The corresponding properties are enumerated in [draft-gu-ppsp-tracker-protocol], e.g., Caching Size, Bandwidth etc.

When receiving the GET_STATUS message, and if the message is well formed and accepted, the peer will search for the requested data and will respond to the leech peer with the specific parameters to the properties the leech peer requested.

TRANSPORT_NEGOTIATION:

The TRANSPORT_NEGOTIATION message is sent from a leech peer to a sender peer in order to negotiate the underlying transport protocol. Leech peer provide a set of transport protocols it supported to sender peer, and leave send peer to choose its preference. Reusing existing transport protocol to transfer data is recommended.

When receiving the TRANSPORT_NEGOTIATION message, and if the message is well formed and accepted, the sender peer will decide the transport protocol and will respond to the leech peer with the specific transport protocol the sender peer preferred.

5. Security Consideration

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 peer-peer communication even in the face of a large number of malicious peers.

As in typical Peer to Peer network, the most significant security issue is that the peers are untrusted. A peer may announce that it has a specific content, but the content might be just noise or it could be poisoned. A peer could also download a large number of chunks but upload very few of them. This problem can be alleviated by incentive mechanism, the goal of which is to reward honest peers and degrade dishonest peers.

5.1. Authentication

To protect the PPSP 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 PeerID 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 draft.

The authetication mechanism MUST allow the means for negotiating data security layer mechanisms to provide data integrity, data confidentiality, and other services, subject to local policies and security requirements.

5.2. Content Integrity Protection Against Polluting Peers/Trackers

Malicious peers may declaim ownership of popular content to the Tracker but serve polluted (i.e., decoy content or even virus/trojan infected contents) to other peers. This kind of pollution can be detected by incorporating a checksum distribution scheme for published sharing content. As content chunks of the same content are

transferred independently and concurrently, correspondent chunk-level checksums MUST be distributed from an authentic origin.

5.3. 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 a peer honestly report its status to the Tracker, or server authentic content to other peers as it claims 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 partner 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.

5.4. Pro-incentive Parameter Trustfulness

Properties for PEER_STATUS messages will consider pro-incentive parameters, which can enable the improvement of 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, ChunkMap is essential, and needs to be accurate. The P2P system should be designed in a way such that a peer will have the incentive to report truthfully its ChunkMap (otherwise it may penalize itself).

Furthermore, both the amount of upload and download 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.

6. References

6.1. Normative References

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

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

[RFC4566] Handley, M., Jacobson, V., and C. Perkins, "SDP: Session Description Protocol", RFC 4566, July 2006.

[RFC2616] Fielding, R., Gettys, J., Mogul, J., Frystyk, H., Masinter, L., Leach, P., and T. Berners-Lee, "Hypertext Transfer Protocol -- HTTP/1.1", RFC 2616, June 1999.

[draft-ietf-p2psip-base]
Jennings, C., Lowekamp, B., Ed., Rescorla, E., and H. Schulzrinne, "draft-ietf-p2psip-base-07", February 2010, <draft-ietf-p2psip-base>.

6.2. Informative References

[I-D.ietf-ppsp-problem-statement]
Zhang, Y., Zong, N., Camarillo, V., Seng, J., and R. Yang, "Problem Statement of P2P Streaming Protocol (PPSP)", Januray 2011, <I-D.ietf-ppsp-problem-statement>.

[I-D.ietf-ppsp-reqs]
Zong, N., Zhang, Y., Pascual, V., and C. Williams, "P2P Streaming Protocol (PPSP) Requirements", February 2011, <I-D.ietf-ppsp-reqs>.

[I-D.ietf-ppsp-survey]
Gu, Y., Zong, N., Zhang, H., Zhang, Y., Camarillo, G., and Y. Liu, "Survey of P2P Streaming Applications", March 2011, <I-D.ietf-ppsp-survey>.

[I-D.gu-ppsp-tracker-protocol]
Cruz, R., Nunes, M., Gu, Y., Xia, J., Bryan, D., Taveira, J., and D. Deng, "PPSP Tracker Protocol", March 2011, <I-D.gu-ppsp-tracker-protocol>.

[I-D.cruz-ppsp-http-peer-protocol]
Gu, Y., Xia, J., Cruz, R., Nunes, M., Bryan, D., and J. Taveira, "PPSP HTTP-Based Peer Protocol", March 2011, <I-D.cruz-ppsp-http-peer-protocol>.

[I-D.li-ppsp-nat-traversal]
Li , L., Wang , J., and W. Chen , "PPSP NAT Traversal", March 2011, <I-D.li-ppsp-nat-traversal>.

[BittorrentSpecification]
"Bittorrent Protocol Specification v1.0", February 2010, <Bittorrent Specification>.

Appendix A. Binary Encoding

Binary Encoding is an encoding of data in plain text. More precisely, it is an encoding of binary data in a sequence of ASCII-printable characters. Binary Encoding is necessary for transmission of data when the channel or the protocol only allows ASCII-printable characters.

The PPSP Peer protocol can be carried on top of IP, UDP, RTP or TCP. But using which layer to carry peer protocol is out of scope in current stage.

The peer message header has the following format:

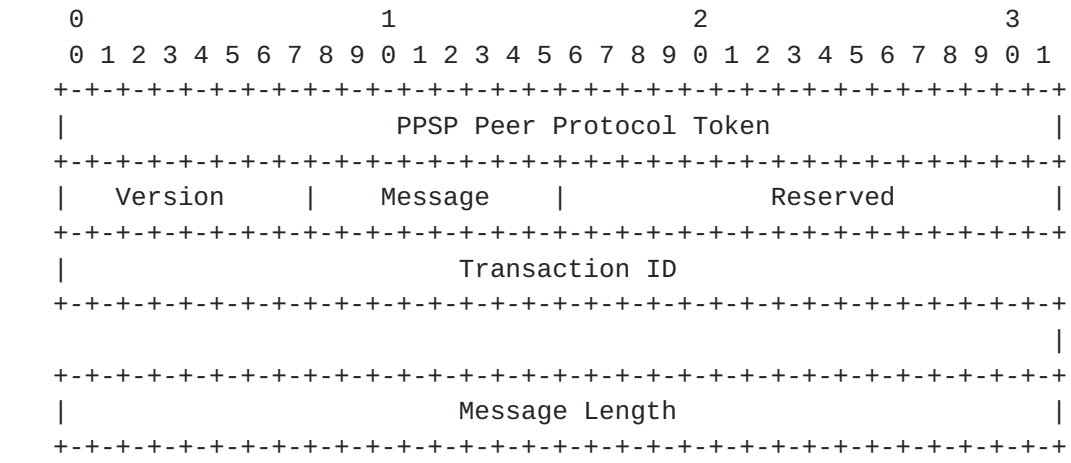

Figure 3: PPSP Peer message header

The fields have the following meaning:

PPSP Peer Protocol Token: 32 bits

A fixed token indicating to the receiver this message is a PPSP Peer Protocol message. The token field is four bytes long. This value MUST be set to 0x50505350, the string "PPSP".

Version: 8 bits The version of the PPSP peer protocol being used in the form of a fixed point integer between 0.1 and 25.4. For the version of the protocol described in this document, this field MUST be set to 0.1. The version field is one byte long.

Message Types: 8 bits

Message types currently have two kinds of value: Request and Response.

Reserved: 16 bits

Not to be assigned. Reserved values are held for special uses, such as to extend the namespace when it becomes exhausted. Reserved values are not available for general assignment.

Transaction ID: 64 bits

Identifies the transaction and also allows receivers to disambiguate transactions which are otherwise identical. Responses use the same Transaction ID as the request they correspond to. Transaction IDs are also used for fragment reassembly.

Message Length: 32 bits:

The length of the message, including header, in bytes. Note if the message is fragmented, this is the length of this message, not the total length of all assembled fragments.

A.1. Methods in Peer messages

To improve the compatibility of the peer methods, the method fields in message extension MUST be encoded as TLV elements as described below and sketched in Figure 4:

To improve the compatibility of the peer methods, the method fields in message extension MUST be encoded as TLV elements as described below and sketched in Figure 4:

- Type: A single-octet identifier that defines the type of the parameter represented in this TLV element.
- Length: A two-octet field that indicates the length (in octets) of the TLV element excluding the Type and Length fields, and the 8-bit Reserved field between them. Note that this length does not include any padding that is required for alignment.

o Value: Variable-size set of octets that contains the specific value for the parameter.

In the extensions, the Reserved field SHALL be set to zero and ignored. If a TLV element does not fall on a 32-bit boundary, the last word MUST be padded to the boundary using further bits set to zero.

In a peer message, any method extension MUST be placed after the mandatory message header. The extensions MAY be placed in any order.

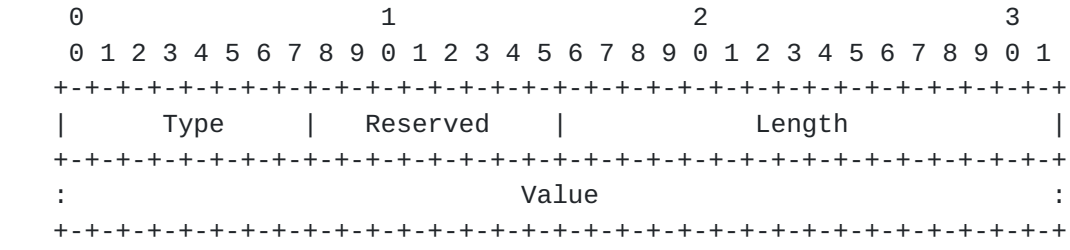

Figure 4: Structure of a TLV element

Method Type: 8 bits

Indicates the method type for the message. There are five method types: GET_PEERLIST, GET_CHUNKMAP, GET_CHUNK, GET_ PROPERTY and TRANSPORT_NEGOTIATION. They are counted from 1 to 5.

Method Body Length: 24 bits

The length of the method body in bytes.

A.1.1. GET_PEERLIST

Peerlist is composed of several pairs of Peer ID and Peer IP. Peer ID is a 128 bit integer that is unique in the P2P streaming system. That's no matter there is a centralized tracker or several distributed trackers in the streaming system, a peer ID should be unique.

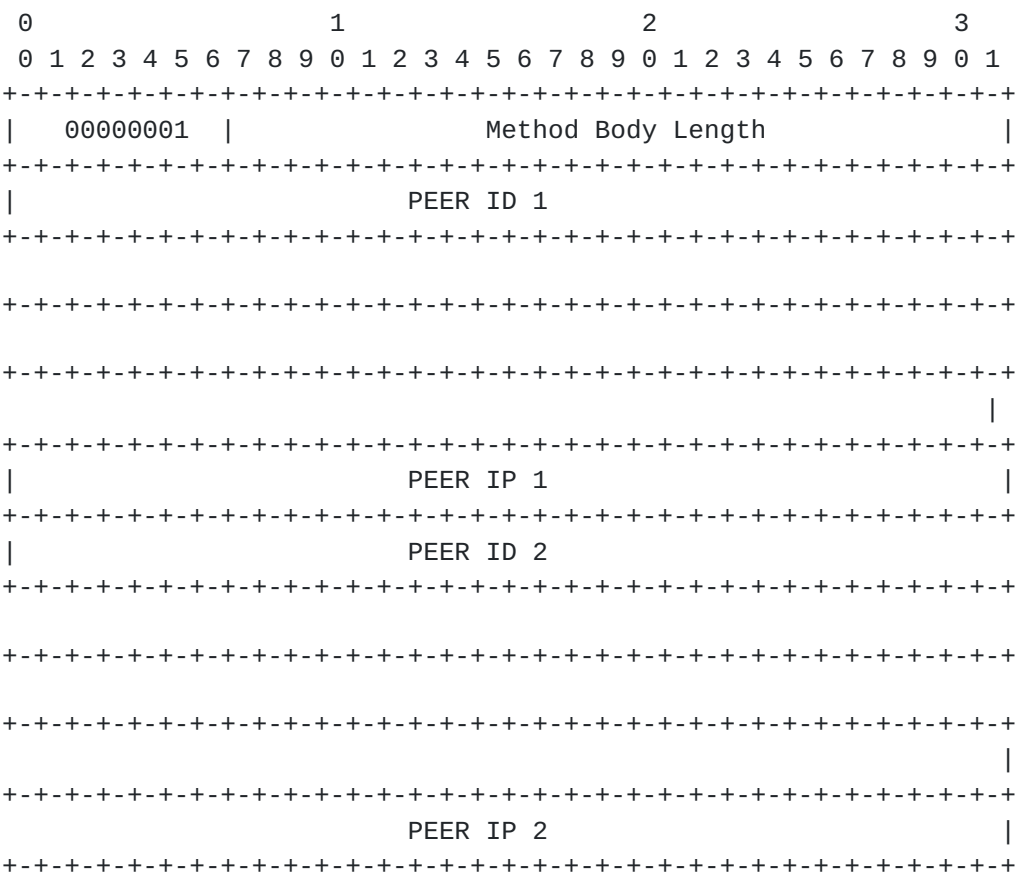

Figure 5: GET_PEERLIST Method Body

A.1.2. GET_CHUNKMAP

Chunkmap of a content (a swarm identified by SwarmID)

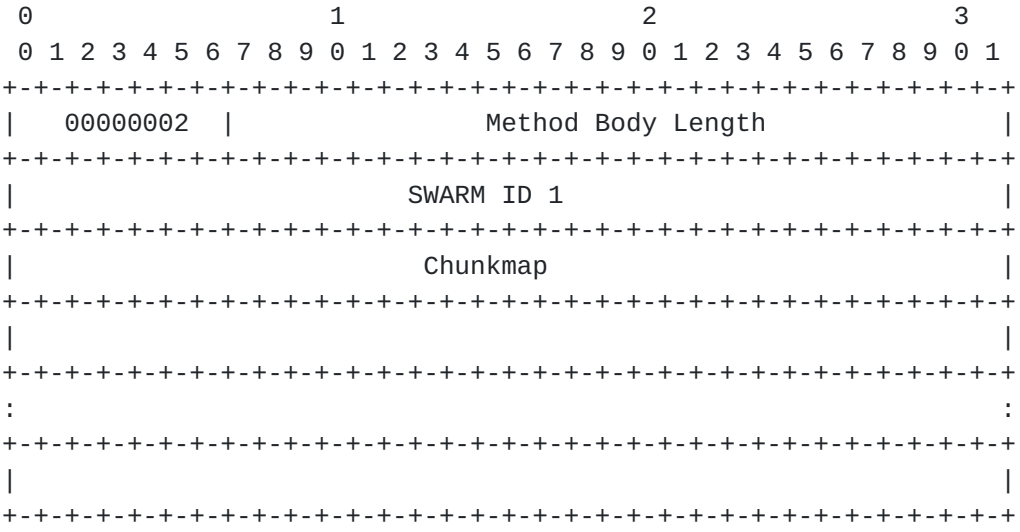

Figure 6: GET_CHUNKMAP Method Body

A.1.3. GET_CHUNK

[TBD]

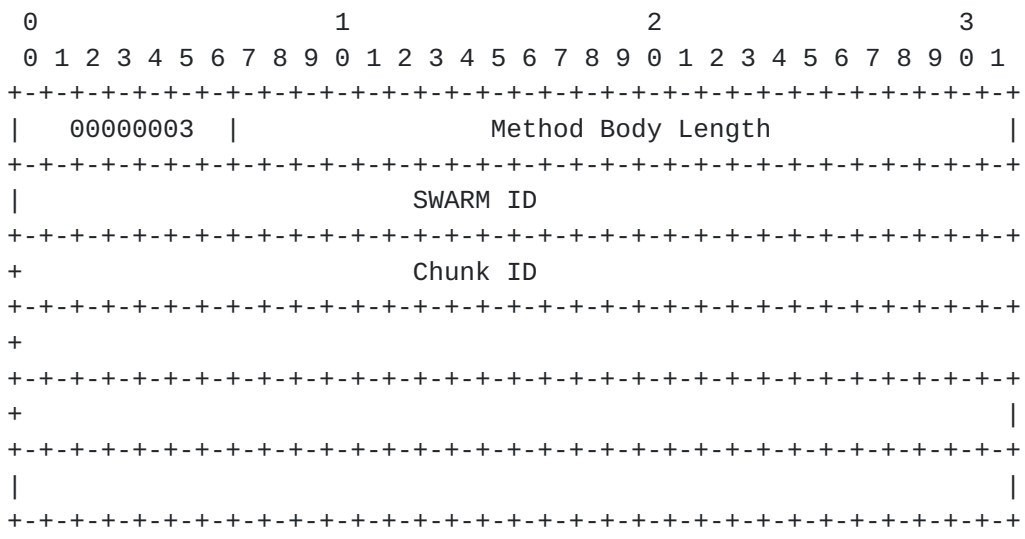

Figure 6: GET_CHUNKMAP Method Body

A.1.4. GET_STATUS

Several property types are defined in I-D.gu-ppsp-tracker-protocol. But not all of the property types are reasonable to be used in peer protocol. So we just list the following property types. New types can be easily added.

PROPERTY	Description	Code
CachingSize	Caching size: available size for caching	0x01
Bandwidth	Bandwidth: available bandwidth	0x02
LinkNumber	Link number: acceptable links for remote peer	0x03
Certificate	Certificate: certificate of the peer	0x04

Table 1: Status changed between peers

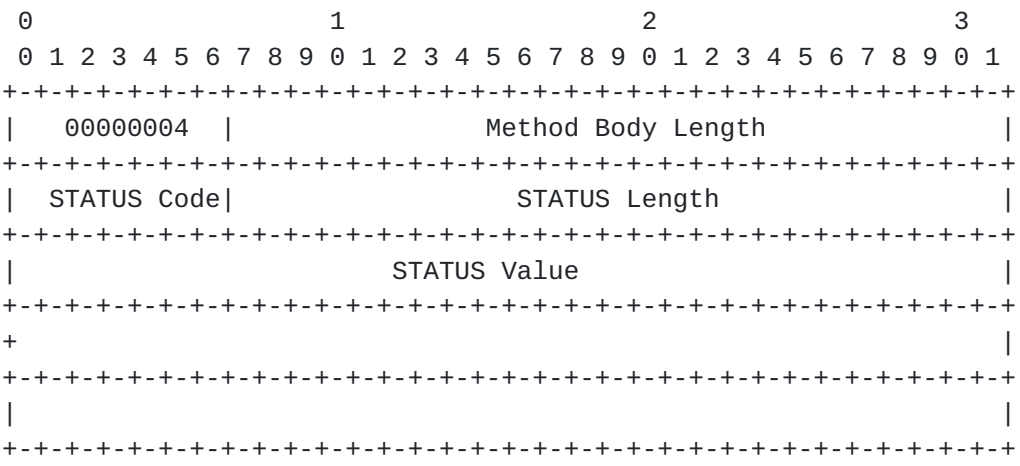

Figure 6: GET_STATUS Method Body

A.1.5. TRANSPORT_NEGOTIATION

To Do: Define mandatory transport protocol and some optional transport protocol.

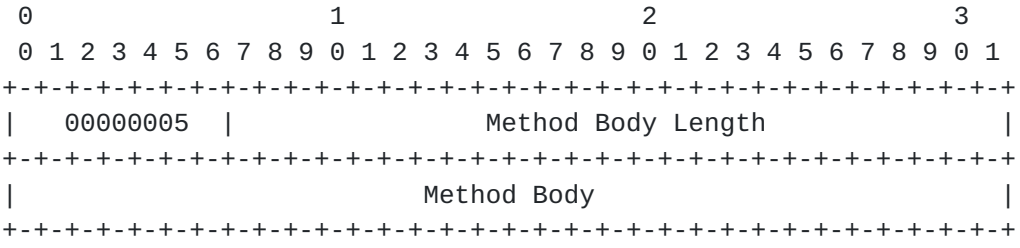

Figure 7: TRANSPORT_NEGOTIATION Method Body

Appendix B. HTTP/XML Encoding

The PPSP Peer Protocol HTTP/XML encoding messages follow the request and response standard formats for HTTP Request and Response messages [RFC2616].

B.1. HTTP/XML Encoding

A Request message is a standard HTTP Request generated by the HTTP Client Peer with the following syntax:

```
<Method> /<Resource> HTTP/1.1
Host: <Host>
```

The HTTP Method and URI path (the Resource) indicates the operation requested. The current proposal uses only HTTP POST as a mechanism for the request messages.

The Host header follows the standard rules for the HTTP 1.1 Host Header.

The Response message is also a standard HTTP Response generated by the HTTP Serving Peer with the following syntax:

```
HTTP/1.1 <StatusCode> <StatusMsg>
Content-Lenght: <ContentLenght>
Content-Type: <ContentType>
Content-Encoding: <ContentCoding>
<Response_Body>
```

The body for both Request and Response messages are encoded in XML for all the PPSP Peer Protocols messages, with the following schema (the XML information being method specific):

```
<?xml version="1.0" encoding="utf-8"?>
<ProtocolName version="#.#">
   <Method>***</Method>     <!-- for the Request method -->
   <Response>***</Response> <!-- for the Response method -->
   <TransactionID>###</TransactionID>
     ...XML information specific of the Method...
</ProtocolName>
```

In the XML body, the *** represents alphanumeric data and ### represents numeric data to be inserted. The <Method> corresponds to the method type for the message, the <Response> corresponds to the response method type of the message and the element <TransactionID> uniquely identifies the transaction.

The Response message MAY use Content-Encoding entity-header with "gzip" compression scheme [RFC2616] for faster transmission times and less network bandwidth usage.

B.2. Method Fields

Table B 1 and Table B 2 define the valid string representations for the requests and responses, respectively. These values MUST be treated as case-insensitive.

PPSP Request	XML Request Value String
GET_PEERLIST	GET_PEERLIST
GET_CHUNKMAP	GET_CHUNKMAP
GET_CHUNK	GET_CHUNK
PEER_STATUS	PEER_STATUS
TRANSPORT_NEGOTIATION	TRANSP_NEGO

Table B 1: Valid Strings for Requests

Response Method Name	HTTP Response Mechanism	XML Response Value
SUCCESSFUL (OK)	200 OK	OK
INVALID SYNTAX	400 Bad Request	INVALID SYNTAX
VERSION NOT SUPPORTED	400 Bad Request	VERSION NOT SUPPORTED
AUTHENTICATION REQUIRED	401 Unauthorized	AUTHENTICATION REQUIRED
MESSAGE FORBIDDEN	403 Forbidden	MESSAGE FORBIDDEN
OBJECT NOT FOUND	404 Not Found	OBJECT NOT FOUND
INTERNAL ERROR	500 Internal Server Error	INTERNAL ERROR
TEMPORARILY OVERLOADED	503 Service Unavailable	TEMPORARILY OVERLOADED

Table B 2: Valid Strings for Responses

B.3. Message Processing

When a PPSP Peer Protocol message is received in a peer, 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. If the message is found to be incorrectly formed or the length does not match the length encoded in the header, the receiver MUST reply with an HTTP 400 response with a PPSP XML body with the Response method set to INVALID SYNTAX.

B.4. GET_PEERLIST Message

The GET_PEERLIST message is sent from a client peer to a selected serving peer in order for a peer to refresh/update the list of active peers in the swarm.

The Request message uses a HTTP POST method with the following body:

```
<?xml version="1.0" encoding="utf-8"?>
<PPSPPeerProtocol version="#.#">
   <Method>GET_PEERLIST</Method>
   <PeerID>***</PeerID>
   <SwarmID>***</SwarmID>
   <TransactionID>###</TransactionID>
</PPSPPeerProtocol>
```

The sender MUST properly form the XML body, MUST set the Method string to GET_PEERLIST, MUST set the PeerID to the PeerID of the peer, MUST set the SwarmID to the joined swarm identifier and randomly generate and set the TransactionID value.

When receiving the GET_PEERLIST message, and if the message is well formed and accepted, the peer will search for the requested data and will respond to the requesting peer with an HTTP 200 OK message response with a PPSP XML payload SUCCESSFUL, as well as the peer list with PeerIDs (and respective IP Addresses) of peers that can provide the specific content.

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

An example of the Response message structure is the following:

```
<?xml version="1.0" encoding="utf-8"?>
<PPSPPeerProtocol version="#.#">
   <Response>OK</Response>
   <SwarmID>***</SwarmID>
   <TransactionID>###</TransactionID>
   <PeerInfoList>
      <PeerInfo>
         <PeerID>***</PeerID>
         <PeerType>***</PeerType>
         <PeerAddresses>
            <PeerAddress ip="##.##.##.##"
                         port="###" />
            <PeerAddress ip="hh:hh:hh:hh:hh:hh:hh:hh"
                         port="###" />
         </PeerAddresses>
         <PeerLocation>****</PeerLocation>
         <ConnectionType>***</ConnectionType>
         <EndPointRankCost>###</EndPointRankCost>
      </PeerInfo>
      <PeerInfo>
         <PeerID>***</PeerID>
         <PeerType>***</PeerType>
         <PeerAddresses>
            <PeerAddress ip="##.##.##.##"
                         port="###" />
            <PeerAddress ip="hh:hh:hh:hh:hh:hh:hh:hh"
                         port="###" />
         </PeerAddresses>
         <PeerLocation>****</PeerLocation>
         <ConnectionType>***</ConnectionType>
         <EndPointRankCost>###</EndPointRankCost>
      </PeerInfo>
   </PeerInfoList>
</PPSPPeerProtocol>
```

The element <PeerInfoList> MAY contain multiple <PeerInfo> child elements.

The element <PeerAddresses> MAY contain multiple <PeerAddress> child elements with attributes "ip" and "port" corresponding to each of the network interfaces of the peer. The "ip" attribute can be expressed in dotted decimal format for IPv4 or 16-bit hexadecimal values (hh) separated by colons (:) for IPv6.

The elements <PeerLocation> and <ConnectionType> have a string format, and together with the element <EndPointRankCost> of numerical integer format, form a set of information related to peer location.

B.5. GET_CHUNKMAP Message

The GET_CHUNKMAP message is sent from a client peer to a selected serving peer in order to receive the map of chunks of a content (of a swarm identified by SwarmID) the other peer presently stores. The chunk map returned by the other peer lists ranges of chunks. The Request message uses a HTTP POST method with the following body:

```
<?xml version="1.0" encoding="utf-8"?>
<PPSPPeerProtocol version="#.#">
   <Method>GET_CHUNKMAP</Method>
   <PeerID>***</PeerID>
   <SwarmID>***</SwarmID>
   <TransactionID>###</TransactionID>
</PPSPPeerProtocol>
```

The sender MUST properly form the XML body, MUST set the Method string to GET_CHUNKMAP, MUST set the PeerID to the PeerID of the peer, MUST set the SwarmID to the joined swarm identifier and randomly generate and set the TransactionID value.

When receiving the GET_CHUNKMAP message, and if the message is well formed and accepted, the peer will search for the requested data and will respond to the requesting peer with an HTTP 200 OK message response with a PPSP XML payload SUCCESSFUL, as well as the map of chunks it currently stores of the specific content.

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

The Response message is an HTTP 200 OK message with the following body, exemplified for a video component of a media clip:

```
<?xml version="1.0" encoding="utf-8"?>
<PPSPPeerProtocol version="#.#">
   <Response>OK</Response>
   <TransactionID>###</TransactionID>
   <StreamInfo>
      <SwarmID>***</SwarmID>
      <Clip>
         <Name>***</Name>
         <ChunkSegments type="video/audio/etc">
            <ChunkSegment from="###" to="###"
                          bitmapSize="###">
               ...(base64 string)...
            </ChunkSegment>
         </ChunkSegments>
      </Clip>
   </StreamInfo>
```

```
      </PPSPPeerProtocol>
```

The element <StreamInfo> MAY contain multiple <Clip> child elements.

The element <ChunkSegments> has an attribute "type" that indicates the type of media of the corresponding chunks.

A <ChunkSegments> element MAY contain multiple <ChunkSegment> child elements with attributes "from" and "to" corresponding to ranges of contiguous chunks. The "from", "to", and "bitmapSize" attributes are expressed as integer number string format. The <ChunkSegment> content corresponds to the chunk map, and is represented as base64 encoded string.

B.6. GET_CHUNK Message

The GET_CHUNK message is sent from a client peer to a serving peer in order to request the delivery of media content chunks. The Request message uses a HTTP POST method with the following body:

```
      <?xml version="1.0" encoding="utf-8"?>
      <PPSPPeerProtocol version="#.#">
         <Method>GET_CHUNK</Method>
         <PeerID>***</PeerID>
         <SwarmID>***</SwarmID>
         <TransactionID>###</TransactionID>
      </PPSPPeerProtocol>
```

The sender MUST properly for the HTTP request for a POST method including the URI path (the Resource) of the chunk. The sender MUST also properly form the XML body, MUST set the Method string to GET_CHUNK, MUST set the PeerID to the PeerID of the peer, MUST set the SwarmID to the joined swarm identifier and randomly generate and set the TransactionID value.

Figure B 1: Example of GET_CHUNK message sequence (simplified)

When receiving the GET_CHUNK message, and if the message is well formed and accepted, the peer will search for the requested data and will respond to the requesting peer with an HTTP 200 OK message response with a PPSP XML payload SUCCESSFUL.

The Response message is an HTTP 200 OK message. If The Data Transport Protocol negotiated is also HTTP/XML, the body of the response to GET_CHUNK can be immediately followed by the chunk data transfer, as shown in Figure B 1.

The response MUST have the same TransactionID as the request.

B.7. PEER_STATUS Message

The PEER_STATUS message is sent from a serving peer to a client peer to indicate its participation status. The information conveyed may include information related to chunk trading like "choke" (to inform the other peer of the intention to stop sending data to it) and "unchoke" (to inform the other peer of the intention to start/re-start sending data to it).

The Request message uses a HTTP POST method with the following body:

```
<?xml version="1.0" encoding="utf-8"?>
<PPSPPeerProtocol version="#.#">
   <Method>PEER_STATUS</Method>
   <PeerID>***</PeerID>
   <SwarmID>***</SwarmID>
   <TransactionID>###</TransactionID>
   <Status>(choke/unchoke)</Status>
</PPSPPeerProtocol>
```

The sender MUST properly form the XML body, MUST set the Method string to PEER_STATUS, MUST set the PeerID to the PeerID of the peer, MUST set the SwarmID to the joined swarm identifier and randomly generate and set the TransactionID value.

When receiving the PEER_STATUS message, and if the message is well formed and accepted, the peer will respond to the requesting peer with an HTTP 200 OK message response with a PPSP XML payload SUCCESSFUL.

If the status signal received is "choke" the peer will stop requesting chunks from the other peer until receiving an "unchoke" status signal.

The only element currently defined in the request message is <Status>, assuming values of "choke" and "unchoke", but, in future, other values may be added.

The Response message is an HTTP 200 OK message with the following body.

```
<?xml version="1.0" encoding="utf-8"?>
<PPSPPeerProtocol version="#.#">
   <Response>OK</Response>
   <TransactionID>###</TransactionID>
</PPSPPeerProtocol>
```

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

The only element currently defined in the response message is the <TransactionID>, but, in future, other elements may be added, for example, containing statistical data or other primitives for chunk trading negotiation.

B.8. TRANSPORT_NEGOTIATION Message

To Do: Define message format, mandatory transport protocol and some optional transport protocols.

Authors' Addresses

Yingjie Gu
Huawei
No.101 Software Avenue
Nanjing, Jiangsu Province 210012
P.R.China

Phone: +86-25-56622638
Email: guyingjie@huawei.com

Jinwei Xia
Huawei
Software No.101
Nanjing, Yuhuatai District 210012
China

Phone: +86-025-86622310
Email: xiajinwei@huawei.com

Rui Santos Cruz
IST/INESC-ID/INOV

Phone: +351.939060939
Email: rui.cruz@ieee.org

Mario Serafim Nunes
IST/INESC-ID/INOV
Rua Alves Redol, n.9
1000-029 LISBOA
Portugal

Phone: +351.213100256
Email: mario.nunes@inov.pt

David A. Bryan
Polycom
San Jose, CA, USA,
USA

Phone:
Email: dbryan@ethernot.org

Joao P. Taveira
ID/INOV

Email: joao.silva@inov.pt