Y. Gu
N. Zong
Huawei
Hui. Zhang
NEC Labs America.
Yunfei. Zhang
China Mobile
Gonzalo. Camarillo
Ericsson
Yong. Liu
Polytechnic University
October 22, 2009

Survey of P2P Streaming Applications
draft-gu-ppsp-survey-01

Status of this Memo

Copyright Notice

Abstract

This document presents a survey of popular Peer-to-Peer streaming applications on the Internet. We focus on the Architecture and Peer Protocol/Tracker Signaling Protocol description in the presentation, and study a selection of well-known P2P streaming systems, including Joost, PPlive, and more. Through the survey, we summarize a common P2P streaming process model and the correspondent signaling process for P2P Streaming Protocol standardization.

Table of Contents

1. Introduction

Toward standardizing the signaling protocols used in today's Peer-to-Peer (P2P) streaming applications, we surveyed several popular P2P streaming systems on their architectures and signaling protocols between peers, as well as, between peers and trackers. The studied P2P streaming systems, running worldwide or domestically, include PPLive, Joost, Cybersky-TV, Octoshape, and more. This document does not intend to cover all design options of P2P streaming applications. Instead, we choose a representative set of applications and focus on the respective signaling characteristics of each kind. Through the survey, we generalize a common streaming process model from those P2P streaming systems, and summarize the companion signaling process as the base of P2P Streaming Protocol standardization.

2. Terminologies and concepts

Chunk: A chunk is a basic unit of partitioned streaming media, which is used by a peer for the purpose of storage, advertisement and exchange among peers [Sigcomm:P2P streaming].

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

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..

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

P2P streaming protocols: P2P streaming protocols refer to multiple protocols such as streaming control, resource discovery, streaming data transport, etc. which are needed to build a P2P streaming system.

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

PPSP protocols: PPSP protocols refer to the key signaling protocols among various P2P streaming system components, including the tracker and peers. PPSP protocols are a part of P2P streaming protocols.

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

Tracker/PPSP tracker: A tracker/PPSP tracker refers to a directory service which maintains the lists of peers/PPSP peers storing chunks for a specific channel or streaming file, and answers queries from peers/PPSP peers.

Video-on-demand (VoD): The scenario where different clients watch different parts of the media recorded and stored during past events.

3. Survey of P2P streaming system

3.1. Joost

Joost announced to give up P2P technology last year. However, as a once very popular P2P streaming application, it's worthwhile to understand how Joost works.

The key components of Joost include servers, super nodes and peers. There are 5 types of servers: Tracker server, Version server, Backend server, Content server and Graphics server. The architecture of Joost system is shown in Figure 1.

First, we introduce the functionalities of Joost's key components through three basic phases. Then we will discuss the Peer protocol and Tracker protocol of Joost.

Installation: Backend server is involved in the installation phase. Backend server provides peer with an initial channel list in a SQLite file. No other parameters, such as local cache, node ID, or listening port, are configured in this file.

Bootstrapping: In case of a newcomer, Tracker server provides several super node addresses and possibly some content server addresses. Then the peer connects Version server for the latest software version. Finally, the peer starts to connect some super nodes to obtain the list of other available peers and begin streaming video contents. Different from Skype, super nodes in Joost only deal with control traffic. They do not relay/forward any media data.

Channel switching: Super nodes are responsible for redirecting clients to content server or peers.

Peers communicate with servers over HTTP/HTTPs and with super nodes/ other peers over UDP.

Tracker Protocol: Because there are super nodes responsible for providing the peerlist/content servers, protocol spoken between tracker server and peers is rather simple. Peers get the addresses of super nodes and content servers from Tracker server over HTTP. After that, Tracker sever will not appear in any stage, e.g. channel switching, VoD interaction. In fact, the protocol spoken between peers and super nodes is more like what we normally called "Tracker Protocol". It enables super nodes to check peer status, maintain peer lists for several, if not all, channels. It provides peer list/ content servers to peers. So in the rest of this section, when we mention Tracker Protocol, we mean the one spoken between peers and super nodes.

Peers will communicate with super nodes in some scenarios using Tracker Protocol.

1 When a peer starts Joost software, after the installation and bootstrapping, the peer will communicate with one or several super nodes to get a list of available peers/content servers.

2 For on-demand video functions, super nodes periodically exchange small UDP packets for peer management.

3 When switching between channels, peers contact super nodes and the latter help the peers find available peers to fetch the requested media data.

Peer Protocol: Until now, we have not gotten enough materials to show what is negotiation process between peers. However, we try to reverse-engineer what is negotiated based on the data in [Joost-experiment]. We omitted the analysis process and directly show our conclusion. Media data in Joost is split into chunks and then encrypted. A chunk is about 5-10 seconds of video data. After receiving peer list from super nodes, a peer negotiates with some or all of the peers in the list to find out what chunks they have. Then the peer makes decision about from which peers to get the chunks. No peer capability information is exchanged in the Peer Protocol.

```
                   +---------------+       +--------------------+
                   | Version Server|       |   Tracker Server   |
                   +---------------+       +--------------------+
                             \                        |
                              \                       |
                               \                      | +----------------+
                                \                     | |Graphics Server|
                                 \                    | +----------------+
                                  \                   |     |
   +---------------+       +--------------+       +---------------+
   |Content Server|--------|     Peer1     |--------|Backend Server|
   +---------------+       +--------------+       +---------------+
                                     |
                                     |
                                     |
                                     |
                              +------------+       +----------+
                              | Super Node |-------|  Peer2   |
                              +------------+       +----------+
```

Figure 1, Architecture of Joost system

3.2. Octoshape

CNN has been working with a P2P Plug-in, from a Denmark-based company Octoshape, to broadcast its living streaming. Octoshape helps CNN serve a peak of more than a million simultaneous viewers. Figure 2 depicts the architecture of the Octoshape system.

Octoshape maintains a mesh overlay topology. Its overlay topology maintenance scheme is similar to that of P2P file-sharing applications, such as BitTorrent. There is no Tracker server in Octoshape, thus no Tracker Protocol. Peers obtain live streaming from content servers and peers over Octoshape Protocol. Several data streams are constructed from live stream. No data streams are identical and any number K of data streams can reconstruct the original live stream. The number K is based on the original media playback rate and the playback rate of each data stream. For example, a 400Kbit/s media is split into four 100Kbit/s data streams, and then k = 4. Data streams are constructed in peers, instead of Broadcast server, which release server from large burden. The number of data streams constructed in a particular peer equals the number of peers downloading data from the particular peer, which is constrained by the upload capacity of the particular peer. To get the best performance, the upload capacity of a peer should be larger than the playback rate of the live stream. If not, an artificial peer may be added to deliver extra bandwidth.

Each single peer has an address book of other peers who is watching the channel. A Standby list is set up based on the address book. The peer periodically probes/asks the peers in the standby list to be sure that they are ready to take over if one of the current senders stops or gets congested. [Octoshape]

Peer Protocol: The live stream is firstly sent to a few peers in the network and then be spread to the rest. When a peer joins a channel, it notifies all the other peers about its presence over Peer Protocol, which will drive the others to add it into their address books. Although [Octoshape] declares that each peer records all the peers joining the channel, we suspect that not all the peers are recorded, considering the notification traffic will be large and peers will be busy with recording when a popular program starts in a channel and lots of peers switch to this channel. Maybe some geographic or topological neighbors are notified and the peer gets its address book from these neighbors.

The peer sends requests to some selected peers for the live stream and the receivers answers OK or not according to their upload capacity. The peer continues sending requests to peers until it finds enough peers to provide the needed data streams to redisplay the original live stream. The details of Octoshape are (not?) disclosed yet, we hope someone else can provide much specific information.

```
         +------------+   +--------+
         |   Peer 1   |---| Peer 2 |
         +------------+   +--------+
              |    \    /      |
              |     \  /       |
              |      \         |
              |     / \        |
              |    /   \       |
              |  /      \      |
   +--------------+    +-------------+
   |     Peer 4   |----|    Peer3    |
   +--------------+    +-------------+

   *****************************************
                   |
                   |
              +---------------+
              | Content Server|
              +---------------+
```

Figure 2, Architecture of Octoshape system

3.3. PeerCast

PeerCast adopts a Tree structure. The architecture of PeerCast is shown in Figure 3.

Peers in one channel construct the Broadcast Tree and the Broadcast server is the root of the Tree. A Tracker can be implemented independently or merged in the Broadcast server. Tracker in Tree based P2P streaming application selects the parent nodes for those new peers who join in the Tree. A Transfer node in the Tree receives and transfers data simultaneously.

Peer Protocol: The peer joins a channel and gets the broadcast server address. First of all, the peer sends a request to the server, and the server answers OK or not according to its idle capability. If the broadcast server has enough idle capability, it will include the peer in its child-list. Otherwise, the broadcast server will choose at most eight nodes of its children and answer the peer. The peer records the nodes and contacts one of them, until it finds a node that can server it.

In stead of requesting the channel by the peer, a Transfer node pushes live stream to its children, which can be a transfer node or a receiver. A node in the tree will notify its status to its parent periodically, and the latter will update its child-list according to the received notifications.

```
               ------------------------------
               |            +---------+      |
               |            | Tracker |      |
               |            +---------+      |
               |                  |          |
               |                  |          |
               |   +---------------------+   |
               |   |   Broadcast server  |   |
               |   +---------------------+   |
               |------------------------------
                     /                     \
                    /                       \
                   /                         \
                  /                           \
            +---------+                  +---------+
            |Transfer1|                  |Transfer2|
            +---------+                  +---------+
             /      \                       /      \
            /        \                     /        \
           /          \                   /          \
      +---------+  +---------+     +---------+  +---------+
      |Receiver1|  |Receiver2|     |Receiver3|  |Receiver4|
      +---------+  +---------+     +---------+  +---------+
```

Figure 3, Architecture of PeerCast system

3.4. Conviva

Conviva[TM][conviva] is a real-time media control platform for Internet multimedia broadcasting. For its early prototype, End System Multicast (ESM) [ESM04] is the underlying networking technology on organizing and maintaining an overlay broadcasting topology. Next we present the overview of ESM. ESM adopts a Tree structure. The architecture of ESM is shown in Figure 4.

ESM has two versions of protocols: one for smaller scale conferencing apps with multiple sources, and the other for larger scale broadcasting apps with Single source. We focus on the latter version in this survey.

ESM maintains a single tree for its overlay topology. Its basic functional components include two parts: a bootstrap protocol, a parent selection algorithm, and a light-weight probing protocol for tree topology construction and maintenance; a separate control structure decoupled from tree, where a gossip-like algorithm is used for each member to know a small random subset of group members; members also maintain pathes from source.

Upon joining, a node gets a subset of group membership from the source (the root node); it then finds parent using a parent selection algorithm. The node uses light-weight probing heuristics to a subset of members it knows, and evaluates remote nodes and chooses a candidate parent. It also uses the parent selection algorithm to deal with performance degradation due to node and network churns.

ESM Supports for NATs. It allows NATs to be parents of public hosts, and public hosts can be parents of all hosts including NATs as children.

```
            ------------------------------
            |            +---------+      |
            |            | Tracker |      |
            |            +---------+      |
            |                  |          |
            |                  |          |
            |   +---------------------+   |
            |   |    Broadcast server |   |
            |   +---------------------+   |
            |------------------------------
                  /                     \
                 /                       \
                /                         \
               /                           \
         +---------+                   +---------+
         |  Peer1   |                  |  Peer2  |
         +---------+                   +---------+
          /      \                       /      \
         /        \                     /        \
        /          \                   /          \
   +---------+  +---------+     +---------+  +---------+
   |  Peer3  |  |  Peer4  |     |  Peer5  |  |  Peer6  |
   +---------+  +---------+     +---------+  +---------+
```

Figure 4, Architecture of ESM system

3.5. PPLive

PPLive is one of the most popular P2P streaming software in China. It has two major communication protocols. One is Registration and peer discovery protocol, i.e. Tracker Protocol, and the other is P2P chunk distribution protocol, i.e. Peer Protocol. Figure 5 shows the architecture of PPLive.

Tracker Protocol: First, a peer gets the channel list from the Channel server, in a way similar to that of Joost. Then the peer chooses a channel and asks the Tracker server for the peerlist of this channel.

Peer Protocol: The peer contacts the peers in its peerlist to get additional peerlists, which are aggregated with its existing list.

For the video-on-demand (VoD) operation, because different peers watch different parts of the channel, a peer buffers up to a few minutes worth of chunks within a sliding window to share with each others. Some of these chunks may be chunks that have been recently played; the remaining chunks are chunks scheduled to be played in the next few minutes. Peers upload chunks to each other. To this end, peers send to each other "buffer-map" messages; a buffer-map message indicates which chunks a peer currently has buffered and can share. The buffer-map message includes the offset (the ID of the first chunk), the length of the buffer map, and a string of zeroes and ones indicating which chunks are available (starting with the chunk designated by the offset). PPlive transfer Data over UDP.

Video Download Policy of PPLive

1 Top ten peers contribute to a major part of the download traffic. Meanwhile, the top peer session is quite short compared with the video session duration. This would suggest that PPLive gets video from only a few peers at any given time, and switches periodically from one peer to another;

2 PPLive can send multiple chunk requests for different chunks to one peer at one time;

PPLive maintains a constant peer list with relatively small number of peers. [P2PIPTV-measuring]

```
         +------------+    +--------+
         |   Peer 2   |----| Peer 3 |
         +------------+    +--------+
                  |          |
                  |          |
                 +--------------+
                 |    Peer 1    |
                 +--------------+
                        |
                        |
                        |
                 +---------------+
                 | Tracker Server|
                 +---------------+
```

Figure 5, Architecture of PPlive system

3.6. PPStream

The system architecture and working flows of PPStream is similar to PPLive. PPStream transfers data using mostly TCP, only occasionally UDP.

Video Download Policy of PPStream

1 Top ten peers do not contribute to a large part of the download traffic. This would suggest that PPStream gets the video from many peers simultaneously, and its peers have long session duration;

2 PPStream does not send multiple chunk requests for different chunks to one peer at one time;

PPStream maintains a constant peer list with relatively large number of peers. [P2PIPTV-measuring]

3.7. SopCast

The system architecture and working flows of SopCast is similar to PPLive. SOPCast transfer data mainly using UDP, occasionally TCP;

Top ten peers contribute to about half of the total download traffic. SOPCast's download policy is similar to PPLive's policy in that it switches periodically between provider peers. However, SOPCast seems to always need more than one peer to get the video, while in PPLive a single peer could be the only video provider;

SOPCast's peer list can be as large as PPStream's peer list. But SOPCast's peer list varies over time. [P2PIPTV-measuring]

3.8. TVants

The system architecture and working flows of TVants is similar to PPLive. TVAnts is more balanced between TCP and UDP in data transmission;

The system architecture and working flows of TVants is similar to PPLive. TVAnts is more balanced between TCP and UDP in data transmission;

TVAnts' peer list is also large and varies over time. [P2PIPTV-measuring]

We extract the common Main components and steps of PPLive, PPStream, SopCast and TVants, which is shown in Figure 6.

```
                        +------------+
                        |   Tracker  |
                       /+------------+
                      /
                     /    +------+
                1,2/     /|Peer 1|
                  /     / +------+
                 /     /3,4,6
           +---------+/              +------+
           |New Peer |---------------|Peer 2|
           +---------+\     4,6      +------+
           |5  |       \
           |---|        \ +------+
                   3,4,6 \|Peer 3|
                          +------+
```

Figure 6, Main components and steps of PPLive, PPStream, SopCast and Tvants

The main steps are:

(1) A new peer registers with tracker / distributed hash table (DHT) to join the peer group which shares a same channel / media content;

(2) Tracker / DHT returns an initial peer list to the new peer;

(3) The new peer harvests peer lists by gossiping (i.e. exchange peer list) with the peers in the initial peer list to aggregate more peers sharing the channel / media content;

(4) The new peer randomly (or with some guide) selects some peers from its peer list to connect and exchange peer information (e.g. buffer map, peer status, etc) with connected peers to know where to get what data;

(5) The new peer decides what data should be requested in which order / priority using some scheduling algorithm and the peer information obtained in Step (4);

(6) The new peer requests the data from some connected peers.

4. A common P2P Streaming Process Model

As shown in Figure 7, a common P2P streaming process can be summarized from Section 3:

1) When a peer wants to receive streaming content:

   1.1) Peer acquires a list of peers/parent nodes from the tracker.

   1.2) Peer exchanges its content availability with the peers on the obtained peer list, or requests to be adopted by the parent nodes.

   1.3) Peer identifies the peers with desired content, or the available parent node.

   1.4) Peer requests for the content from the identified peers, or receives the content from its parent node.

2) When a peer wants to share streaming content with others:

   2.1) Peer sends information to the tracker about the swarms it belongs to, plus streaming status and/or content availability.

```
+----------------------------------------------------------+
|   +--------------------------------+                     |
|   |              Tracker           |                     |
|   +--------------------------------+                     |
|        ^  |                    ^                         |
|        |  |                    |                         |
|  query |  | peer list/         |streaming Status/        |
|        |  | Parent nodes       |Content availability/   |
|        |  |                    |node capability          |
|        |  |                    |                         |
|        |  V                    |                         |
|   +-------------+         +------------+                 |
|   |    Peer1    |<------->|  Peer 2    |                 |
|   +-------------+ content/+------------+                 |
|                   join requests                          |
+----------------------------------------------------------+
```

Figure 7, A common P2P streaming process model

The functionality of Tracker and data transfer in Mesh-based application and Tree-based is a little different. In the Mesh-based applications, such as Joost and PPLive, Tracker maintains the lists of peers storing chunks for a specific channel or streaming file. It provides peer list for peers to download from, as well as upload to, each other. In the Tree-based applications, such as PeerCast and Canviva, Tracker directs new peers to find parent nodes and the data flows from parent to child only.

5. Security Considerations

This document does not consider security issues. It follows the security consideration in [draft-zhang-ppsp-problem-statement].

6. Acknowledgments

We would like to acknowledge Jiang xingfeng for providing good ideas for this document.

7. Informative References

[PPLive] "www.pplive.com".

[PPStream] "www.ppstream.com".

[CNN] "www.cnn.com".

[OctoshapeWeb] "www.octoshape.com".

[Joost-Experiment] Lei, Jun, et al., "An Experimental Analysis of Joost Peer-to-Peer VoD Service".

[Sigcomm_P2P_Streaming] Huang, Yan, et al., "Challenges, Design and Analysis of a Large-scale P2P-VoD System", 2008.

[Octoshape] Alstrup, Stephen, et al., "Introducing Octoshape-a new technology for large-scale streaming over the Internet".

[Conviva] "http://www.rinera.com/".

[ESM04] Zhang, Hui., "End System Multicast, http://www.cs.cmu.edu/~hzhang/Talks/ESMPrinceton.pdf", May .

[Survey] Liu, Yong, et al., "A survey on peer-to-peer video streaming systems", 2008.

[draft-zhang-alto-traceroute-00] "www.ietf.org/internet-draft/draft-zhang-alto-traceroute-00.txt".

[P2PStreamingSurvey]
Zong, Ning, et al., "Survey of P2P Streaming", Nov. 2008.

[P2PIPTV_measuring]
Silverston, Thomas, et al., "Measuring P2P IPTV Systems".

[Challenge]
Li, Bo, et al., "Peer-to-Peer Live Video Streaming on the Internet: Issues, Existing Approaches, and Challenges", June 2007.

Authors' Addresses

Gu Yingjie
Huawei
Baixia Road No. 91
Nanjing, Jiangsu Province 210001
P.R.China

Phone: +86-25-84565868
Fax: +86-25-84565888
Email: guyingjie@huawei.com

Zong Ning
Huawei
Baixia Road No. 91
Nanjing, Jiangsu Province 210001
P.R.China

Phone: +86-25-84565866
Fax: +86-25-84565888
Email: zongning@huawei.com

Hui Zhang
NEC Labs America.

Email: huizhang@nec-labs.com

Zhang Yunfei
China Mobile

Email: zhangyunfei@chinamobile.com

Gonzalo Camarillo
Ericsson

Email: Gonzalo.Camarillo@ericsson.com

Liu Yong
Polytechnic University

Email: yongliu@poly.edu