PPETP: Peer-to-Peer Epi-Transport Protocol
draft-bernardini-ppetp-00

Riccardo Bernardini, Roberto Cesco Fabbro, Roberto Rinaldo
University of Udine

IETF-78, July 2010
Outline

1. Motivation (why I am here)

2. PPETP overview
   - Data reduction
   - Network structure
   - PPETP and other protocols (SDP/RTSP/...)
   - Plugin structure

3. Conclusions
Motivation

What is?

- The initial motivation was
  - P2P streaming multimedia data (e.g., video)
  - to nodes with limited upload bandwidth (e.g., residential users)
- It evolved in a overlay multicast P2P protocol
  - limited upload bandwidth nodes can contribute
  - Built-in resilience to peer departure/packet loss/poisoning
  - Integrable with existing protocols
  - Potentially useful tool
Motivation (2)
Current status

**Do you have running code?** Yes, we have

- Implemented in an Ada library (+ side software)
- Source code available on SourceForge
- **Stable** core, but fine details still “fluid”
- Current version documented in an I-D

**Right time to get some feedback**
PPETP Overview
PPETP Characteristics

- Reduction procedure
- Network structure
- Pseudo-address (and integration with other protocols)
- Plugin structure
PPETP Overview
Reduction functions
Initial Design Goals
(Historical)

- We wanted a protocol that could cope with

<table>
<thead>
<tr>
<th>Asymmetric bandwidth</th>
<th>Data losses</th>
</tr>
</thead>
</table>
| • Residential users have not enough bandwidth to upload live multimedia content | • Data losses due to  
  - Congestion  
  - Sudden peer departure |
The cornerstone: the reduction procedure

• The idea of reduction:
  – The data stream is a sequence of packets
  – We reduce the size of each packet (by a factor of $R$)
  – The node will upload the reduced packets
  – Each node will...  
    * Receive $N \geq R$ reduced packets from other peers
    * Reconstruct the content packet
    * Reduce the content packet and send the result to other peers
Example of reduction function
Inspired to secret sharing techniques (DCC’08)

1. A **content packet** is interpreted as a matrix

\[
C = \begin{bmatrix}
  c_1 & c_{R+1} & \cdots \\
  c_2 & c_{R+2} & \cdots \\
  \vdots & \vdots & \ddots \\
  c_R & c_{2R} & \cdots 
\end{bmatrix}
\]

with \( R \) rows and elements in \( \text{GF}(2^d) \)

2. **At startup** each node chooses the **reduction parameter** \( b \in \text{GF}(2^d) \) and creates the row vector

\[
r_b = [1, b, \cdots, b^{R-1}]
\]

3. The **reduced packet** is the packet corresponding to the row vector

\[
u_b = r_b C
\]

**Note:** \( u_b \) is \( R \) times **smaller** than \( C \)
Example of reduction function
Packet reconstruction

1. A node receives $u_1, \ldots, u_R$ reduced versions corresponding to reduction parameters $b_1, \ldots, b_R$

2. The node reconstructs $C$ by solving

\[
\begin{bmatrix}
u_1 \\
u_2 \\
\vdots \\
u_R
\end{bmatrix}
= \begin{bmatrix}
r_1 \\
r_2 \\
\vdots \\
r_R
\end{bmatrix}
\]

\[
C = \begin{bmatrix}
1 & b_1 & b_2^1 & \cdots & b_{R-1}^1 \\
1 & b_2 & b_2^2 & \cdots & b_{R-1}^2 \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
1 & b_R & b_R^2 & \cdots & b_R^{R-1}
\end{bmatrix}
\]

- Matrix $R$ invertible iff $b_i \neq b_j$
- Values $b_i$ communicated during handshaking $\Rightarrow$ matrix $R$ inverted only once
Reduction functions in PPETP

• **Generalization** of the just described approach

• PPETP does *not fix* a specific reduction procedure, but
  
  – Reduction procedures are expected to be defined in external documents (**plugin structure**, easier to extend)
  
  – Current I-D includes two *prêt-à-porter* reduction procedures
    
    * **Vandermonde** (described above)
    * **Basic** (no reduction at all)

• PPETP reduction functions are **expected to**
  
  – **Reduce** packet size
  
  – Be **parametrized**
  
  – Enjoy the **R-reconstruction property**
Reduction functions in PPETP
Why they help

• **Smaller bandwidth** required for reduced packets
  
  (puncturing options also available)

• Resilience to *packet loss*
  
  – **Receive** data from $N > R$ peers
  – **Reconstructs** as soon as $R$ out of $N$ reduced packets are received
  – Streaming can continue even if up to $N - R$ peers suddenly leave

• Counteract *poisoning*
  
  – **Receive** data from $N > R$ peers
  – **Reconstructs** using $R$ reduced packets
  – **Check coherence** of the remaining $N - R$ packets
Data format

- Content packets are just "a bunch of byte"
- Any type of packet (RTP, RTCP, . . .)
- Any data type (video, audio, 3D?, smells?)
- Encoded as you want (classical, scalable, multiple description, distributed, . . .)
PPETP Overview
Network structure
Network structure in PPETP

- **Stable** network (*push*)
  - Peers *open/close connections* using *control packets*
  - Control packets
    - can be sent under the *control* of the *application*
    - can be sent *on the behalf* of another node (if security policies allow)
    - can trigger *connection establishment procedure*
  - *Connection parameters* (e.g., reduction parameters) are communicated during the handshaking

- No specific network topology (*separation* between topology and transport)

- Runs over **UDP** (IPv4/IPv6) (DCCP?)
Network structure in PPETP (2)
Only constraint: at least $R$ upper peers

- Multiple trees
- Heterogeneous network (size = upload bandwidth)
- Onion skin
- Non-acyclic network
Network creation

Centralized or distributed? You choose...

- **Strongly centralized**
  - A central server sends Start packets on the behalf of the new peer
  - Server load
  - Good control on the network

- **Strongly distributed**
  - The new node receives “entry point” of a DHT and finds its own peers
  - Small server load
  - Less control on the network structure

- **Something in between...**
PPETP Overview
Pseudo-address
Pseudo-address of a PPETP session

- It would be convenient to refer to a PPETP session with an \((\text{host}, \text{port})\) pair

- A PPETP session is a \textit{distributed} object

❓ What kind of a meaning can we associate to a \((\text{host}, \text{port})\) pair?
Pseudo-address of a PPETP session (2)

- A PPETP session is a complex object that needs to be configured (e.g., by fixing the reduction procedure)

- In a PPETP pseudo-address (host, port)
  - **host** is the address of an host queried for configuration parameters
  - **port** is a 16-bit integer that identifies the PPETP session
  - Every P2P structure needs a **starting point**, for PPETP the starting point is the configuration server

- Configuration query protocol
  - Light-weight and stateless
  - Designed to be as DoS-resilient as possible
  - It allows for user authentication
  - It can redirect the user to more complex protocols (handled by plugins)
Pseudo-address and existing protocols
The availability of a pseudo-address makes it easy to integrate PPETP with existing protocols

v=0
o=
s=PPETP example
c=IN IP4 ppetp.example.com

other SDP lines

m=video 42000 RTP/AVP/ PPETP 0

PPETP session number for RTP
RTCP packet are sent over session number 42001

Streaming is done over PPETP
The standpoint of the application writer

To the application programmer PPETP looks like a **multicast** transport protocol.
PPETP Overview

Plugin structure
Plugin structure in PPETP

- PPETP makes use of several tools (e.g., reduction procedures, signature algorithms, connection establishment procedures)
- PPETP does not fix them, but demands their definition to external documents
- For the sake of usability the I-D includes definition of default versions of the tools
- Data format designed for easy extendability
  - Data processed by plugins are defined as opaque sequence of octets
  - It is possible to load new plugins at run-time
Plugin structure in PPETP (2)
Example: data packet format

```
+-----------------+-----------------+-----------------+-----------------+
| V=0 | C | P | I | F | G | H | Channel |
+-----------------+-----------------+-----------------+-----------------+
| Stream ID       | Sequence Number |
+-----------------+-----------------+-----------------+-----------------+
```

Payload (variable size) <- Depends on reduction function

Sender Signature (variable size) <- Depends on signature algorithm
## Plugin structure in PPETP (3)

### After signature verification

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>V=0</td>
<td>Channel</td>
</tr>
<tr>
<td>Stream ID</td>
<td>Sequence Number</td>
</tr>
<tr>
<td>Payload (variable size)</td>
<td></td>
</tr>
<tr>
<td>Sender Signature (variable size)</td>
<td></td>
</tr>
</tbody>
</table>
### Plugin structure in PPETP (4)

**After header processing**

<table>
<thead>
<tr>
<th>0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1</th>
<th>22</th>
</tr>
</thead>
<tbody>
<tr>
<td>V=0</td>
<td>C</td>
</tr>
<tr>
<td>Stream ID</td>
<td>Sequence Number</td>
</tr>
<tr>
<td>+-----------------------------------------------------------------+-------------------------------------</td>
<td></td>
</tr>
<tr>
<td>Payload (variable size)</td>
<td>&lt;- reconstruction</td>
</tr>
<tr>
<td>Sender Signature (variable size)</td>
<td></td>
</tr>
<tr>
<td>+-----------------------------------------------------------------+-------------------------------------</td>
<td></td>
</tr>
</tbody>
</table>

Handed to plugin
Conclusions
Conclusions

- PPETP
  - Peer-to-peer protocol oriented to *streaming* applications
  - Usable even with **low upload bandwidth** nodes
  - **Integrable** with current protocols (*multicast*-like)
  - **Stable** core, but *not frozen* yet