PPETP: a Peer-to-Peer Epi-Transport Protocol implemented in Ada

Riccardo Bernardini
DSP group – DIEGM – University of Udine
riccardo.bernardini@uniud.it
Tel: +39-0432-55-8271

February 4-5, 2012 – FOSDEM – Brussels

Outline

- Introduction
  - The application
  - The problem
  - Issues with P2P streaming
  - Overview of PPETP
- Implementation details
- Conclusions

Introduction

The application

- Multimedia streaming over the Internet
  -  Personalized content
  -  Personalized commercials
  -  Interactivity
  -  Easier to upgrade to new formats (e.g. HD, 3D, smells. . .)
  -  Adaptable to many devices (e.g., home theater, TV, mobile phone)

The problem

- Good quality content = large bandwidths

DVD quality	$\rightarrow$	1 Mbit/sec
HD quality	$\rightarrow$	10 Mbit/sec

- Successful program = many users

F1 races	$\approx$	60 millions viewers/race	(2008)
Beijing Olympics	$\approx$	2 billions viewers	(opening ceremony)

- Good quality $\times$ Success = Very large bandwidth required
  - Bandwidth required = Content bandwidth $\times$ Number of users

Solutions
Currently available

- Content Delivery Network (CDN)
  - More than one server used
  -  Load shared
  -  The overall bandwidth remains the same
- Multicast
  -  Low bandwidth requirements
  -  Multicast devices needed
  -  A mess across different Autonomous Systems
  - Attractive for Internet Service Providers

Solutions (2)

Currently researched: Peer-to-Peer Streaming

Use the *fire brigade principle*

→ Each user passes the content to another user

→ Just feed only few users

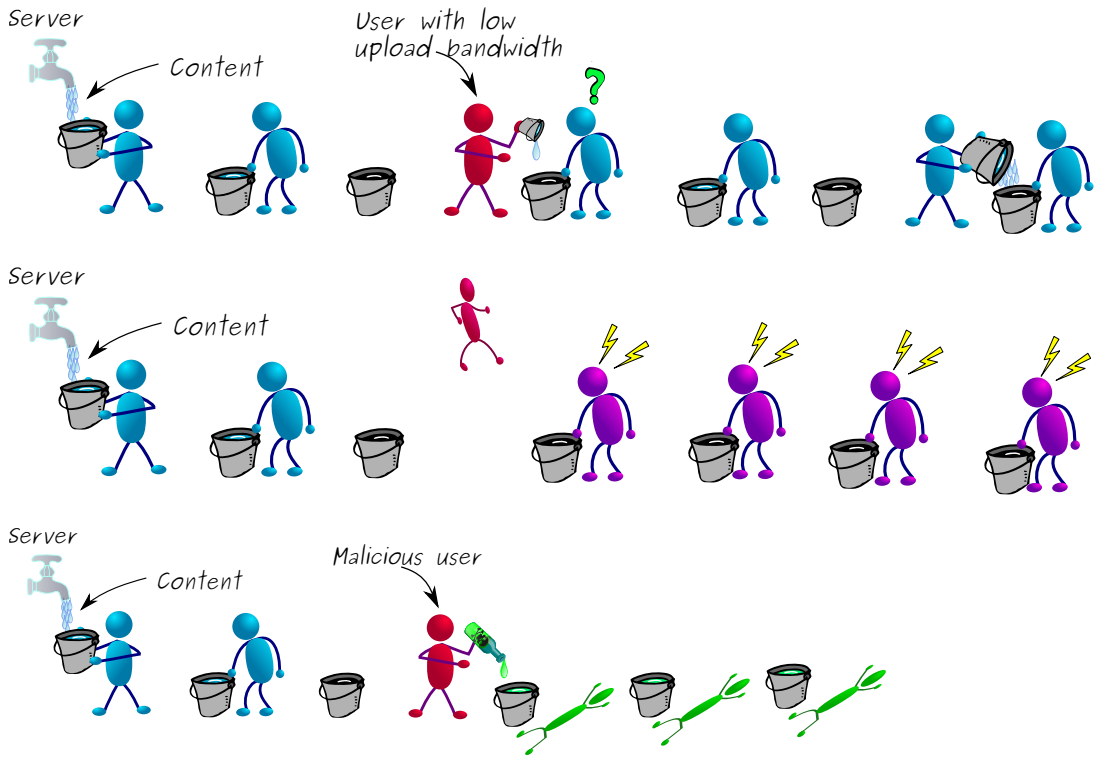
→ The network will take care of itself

Peer-to-Peer Streaming Challenges

Limited upload bandwidth

Sudden user departure

Stream poisoning



Introduction

Reduction functions

Reducing packets: the basic idea

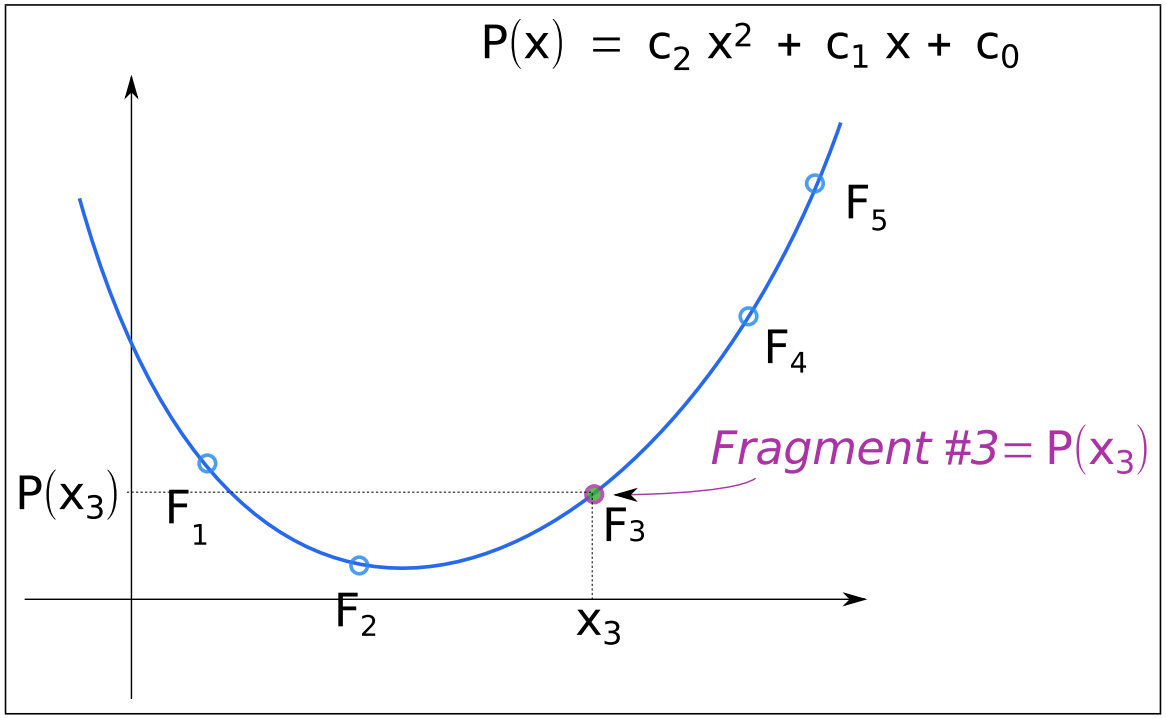
- Aimed to solving the asymmetric bandwidth problem
- Objective: to reduce the upload bandwidth by a factor R
- Solution: split each *content packet*
  - In $N \geq R$ *fragments*
  - With the property that

> *Content recoverable from* any set *of* R *fragments*

- How do we do that?
- It is easy...

. . . when you know the trick. . .

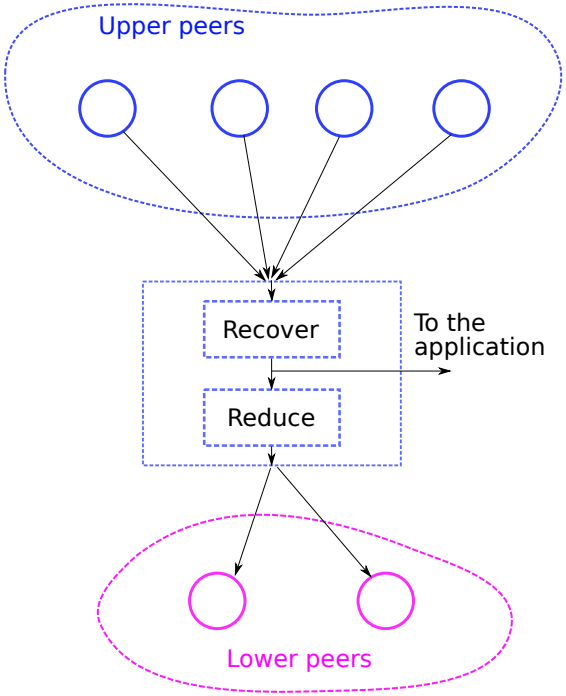
Suppose a packet has $R = 3$ bytes c_0, c_1 and c_2



Fragment bandwidth = 1/3 Content bandwidth

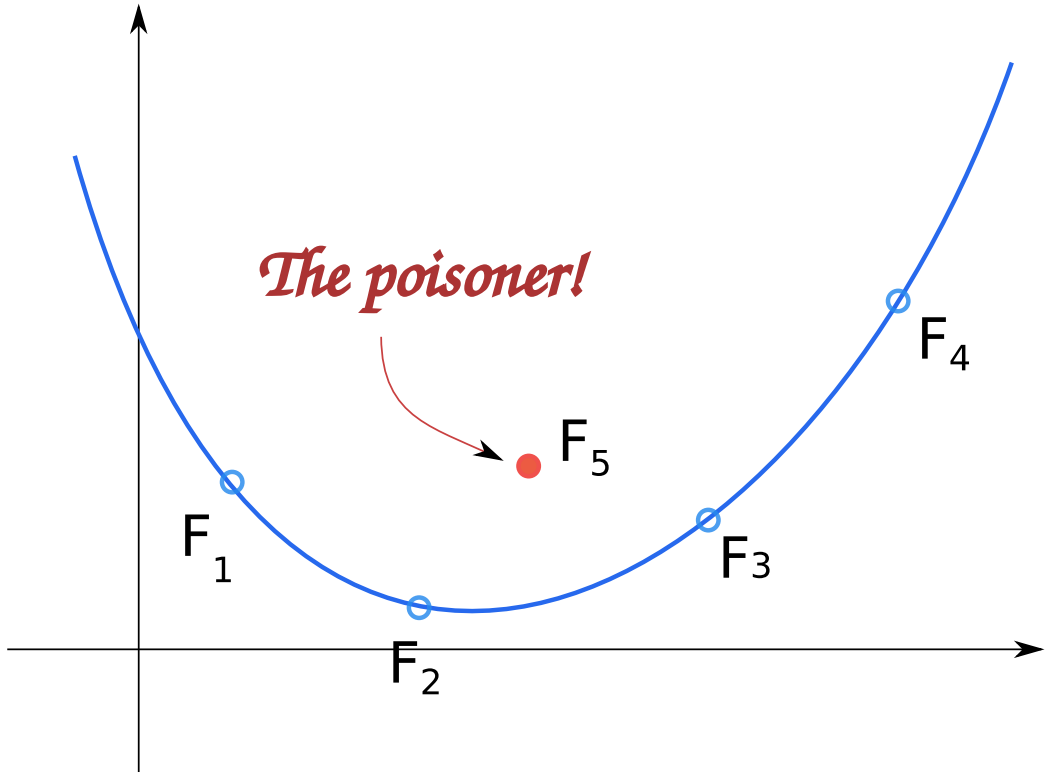
Reduction in a ~~nutshell~~ slide

- At startup
  1. Choose a value x_i
  2. Send x_i to your lower peers
  3. Contact N upper peers
- For every packet
  1. Receive reduced packets from the upper peers
  2. Recover the content packet
  3. Move the content packet to the application
  4. Reduce the recovered content packet
  5. Send the fragment(s) to your lower peers
- Advantages
  - Smaller upload bandwidth
  - Resilience to packet loss
  - Counteract poisoning



Counteracting poisoning

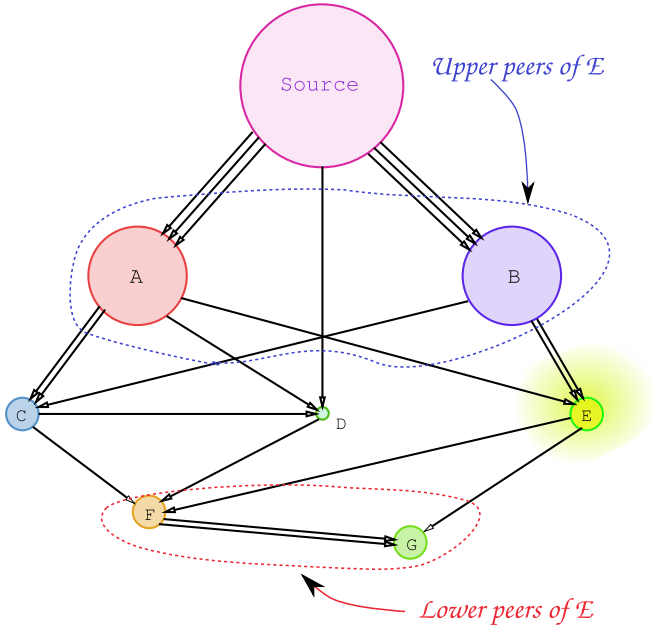
$$P(x) = c_2 x^2 + c_1 x + c_0$$



PPETP

PPETP = Peer-to-Peer Epi-Transport Protocol

- Push network
- Any topology
- Any data type
- Security feautures (e.g., signature, encryption) built-in
- Easy integration with existing stuff



PPETP API cheat sheet

Action	PPETP	Multicast
Open	`Ses := PPETP.Open`	`s=socket()`
Join	`PPETP.Connect (Ses, Host, Port)`	`setsockopt()`
Read	`PPETP.Receive (Ses, Buffer, Last)`	`recv()`
Write	`PPETP.Send (Ses, Data)`	`send()`
Close	`PPETP.Close (Ses)`	`close()`

- PPETP Sessions identified by a pseudo-address (host, port)
  - port is a 16-bit session ID
  - host is a configuration server


Why is this nice?

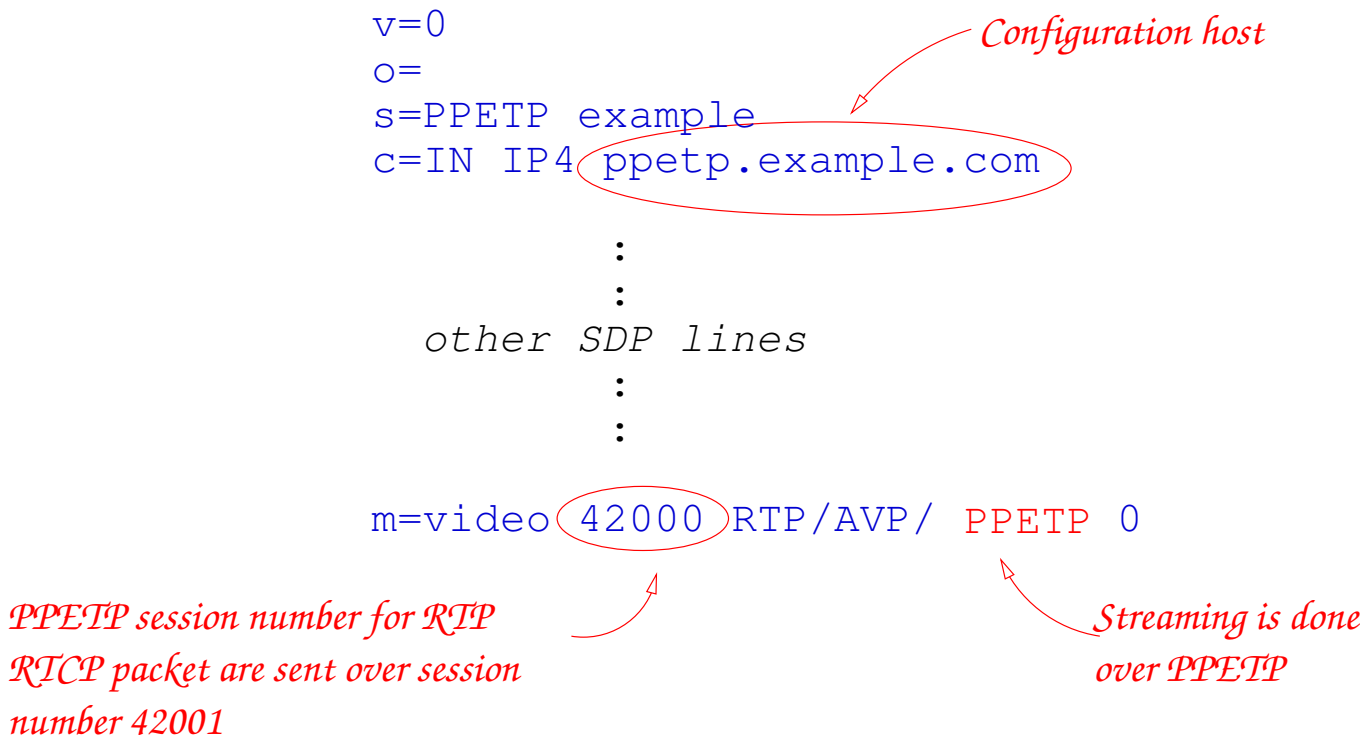

PPETP syntactically similar to multicast


Basic usage very similar to multicast


More flexibility using low-level functions

Pseudo-address and existing protocols

Pseudo-address makes it easy to integrate PPETP with existing protocols



If you can do it with multicast, you can do it with PPETP

Conversion to BSD sockets

- A PPETP session is represented by an object of type `PPETP_Session` which is not a BSD socket, but. . .

  . . . most of current software uses BSD sockets

  . . . PPETP sockets are not usable with `select()`

"Convert" PPETP sessions into BSD sockets

```
function PPETP.Read_Socket (Session) return Integer;
function PPETP.Write_Socket (Session) return Integer;
```

Those are true BSD sockets

Implementation Details

Outline

- Introduction
- Implementation details
  - Current status
  - API
  - Internal structure
  - Plugin implementation
  - Network Access
  - Conversion to Sockets
- Conclusions

Current status

- Described in an Internet-Draft
- A library implemented in INTERCAL
  - Almost 1.5 Mbyte of source code!
  - Developed on Linux, works also on Windos
- "Satellite" software (e.g., converter UDP $\leftrightarrow$ PPETP)
- Current version works, but no official release yet
- Hosted on *SourceForge* http://corallo.sourceforge.net

Current status

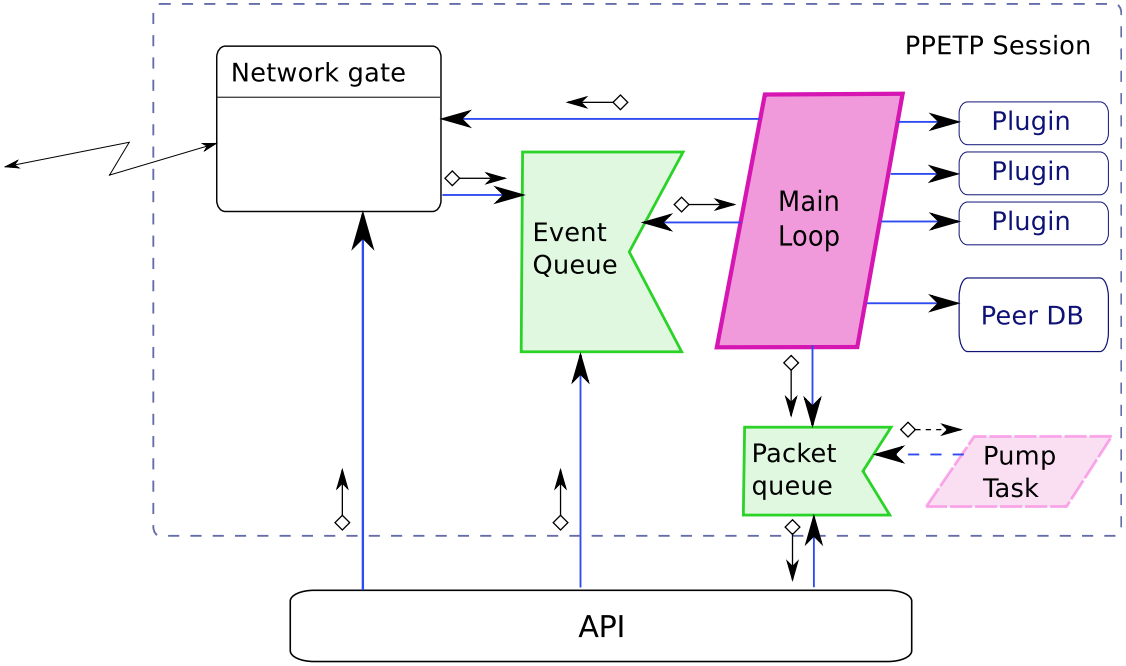
- Described in an Internet-Draft (> 100 pages )
- A library implemented in ~~INTERCAL~~ Ada :-)
  - Almost 1.5 Mbyte of source code!
  - Developed on Linux, works also on Windos
- "Satellite" software (e.g., converter UDP $\leftrightarrow$ PPETP)
- Current version works, but no official release yet
- Hosted on *SourceForge* http://corallo.sourceforge.net

Implementation Details

Internal Structure

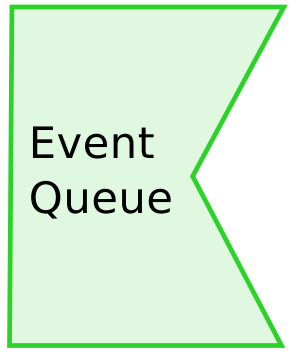
Internal Structure Overview

- One socket = one session
- Most of the processing done by the Main Loop
- The plugins interface themselves with the main loop
- Requests read from the Event Queue
- Packets are sent and received via the Network Gate
- Recovered packets are sent to the application via the Packet Queue

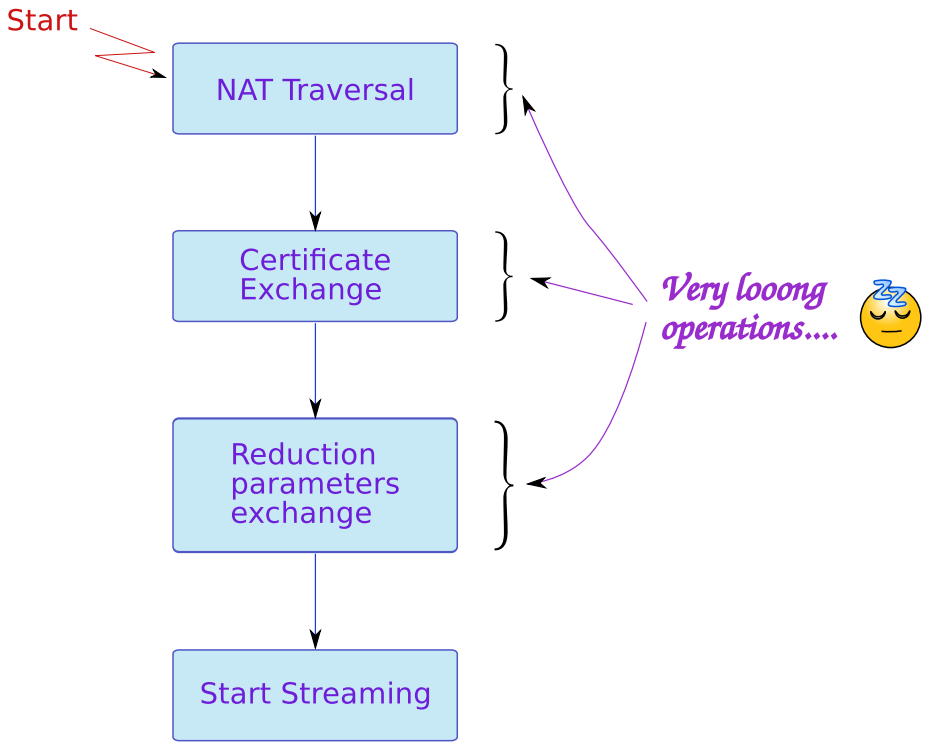


The Event Queue

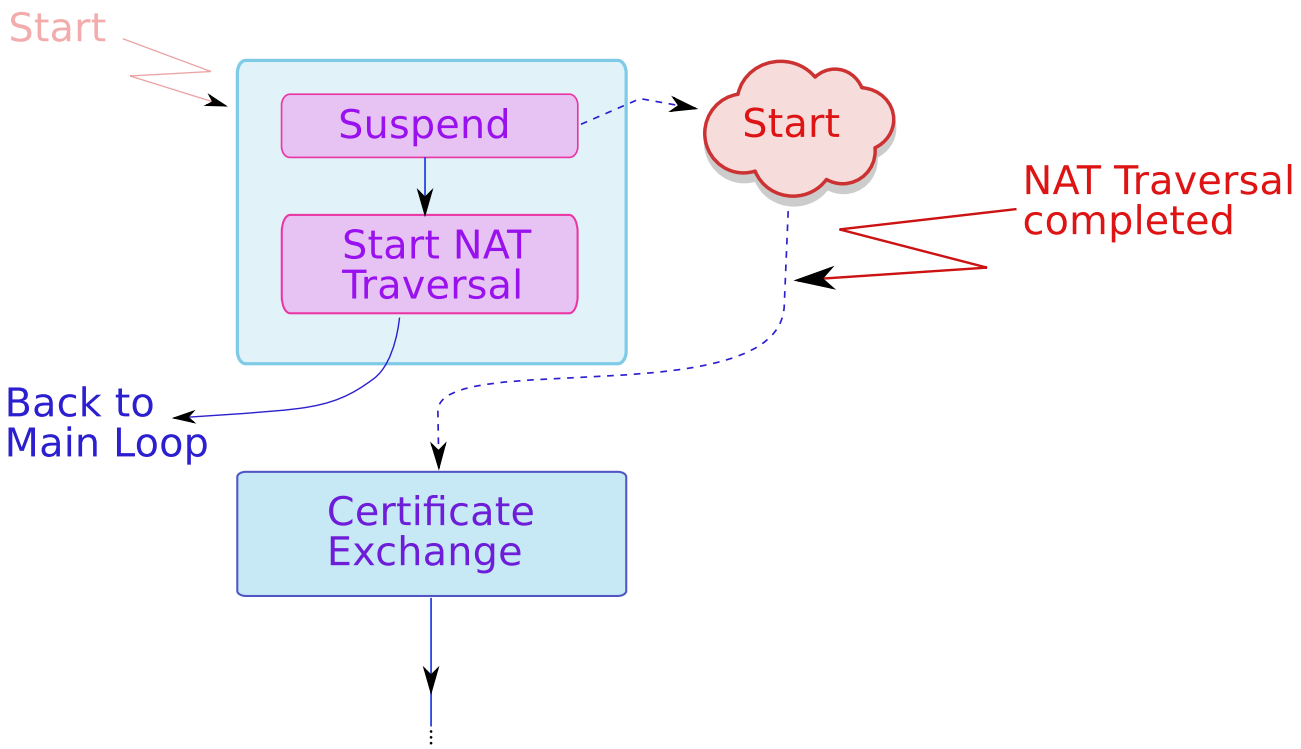
- The main loop gets the requests from the Event Queue
- Events can be generated by
  - The network (received packets)
  - The API
  - The main loop itself
- Events can be
  - Immediate
  - Scheduled (timeouts)
  - Suspended



Suspended Events



Suspended Events



Implementation Details

Plugin Implementation

PPETP Plugins

- In PPETP many parts are defined as plugins
- Plugins are used for
  - Reduction
  - Signatures
  - Congestion control
  - ...

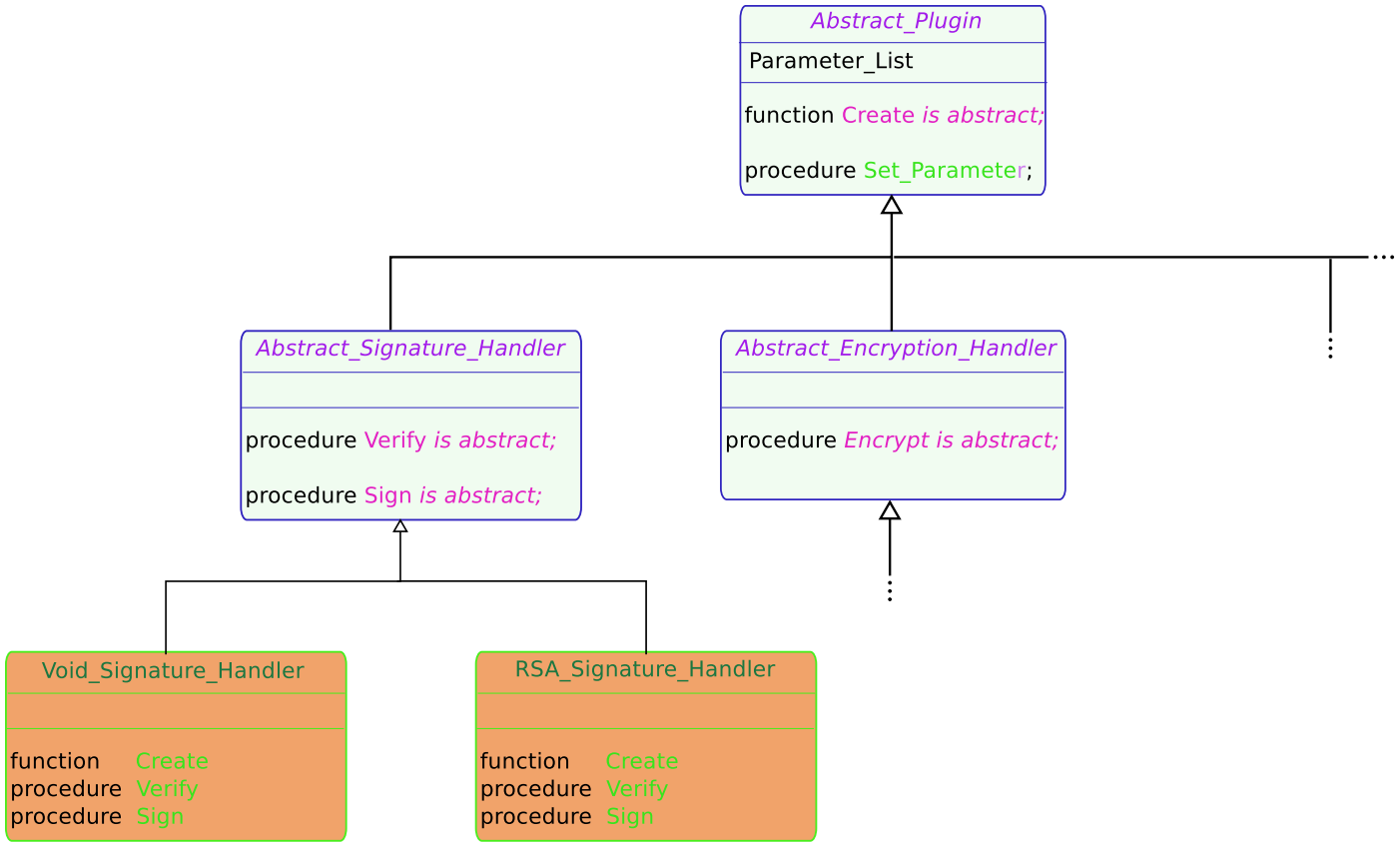
Plugin structure

- Every plugin has
  - A class (reduction, signature, . . .)
  - A name and an index
  - A set of parameters; Every parameter has
    - A name
    - A positional index
- Each session → one plugin per class
- Plugins chosen at configuration time

Constraints

- Session plugins
  - Specified at configuration time by their name
  - Configured via a parameter list
    $\rightarrow$ list of pairs of strings `(Name, Value)`
  - Details of parameter formats *private* to the plugin
- We want to add plugins without recompiling the library
- Plugins should be dynamically searchable and loadable

The plugin hierarchy



Example

```
package PPETP.Signature_Handlers is
   pragma Elaborate_Body;              -- <== Important!

   type Abstract_Signer is abstract new Plugins.Abstract_Plugin with private;

   procedure Sign
       (Handler :in out Abstract_Signer;
        Data    :in out Data_Buffer)
   is abstract;
   -- Compute the signature of Data and append it to Data

   type Signer_Access is access all Abstract_Signer'Class;

   function New_Signer (Name :  String) return Signer_Access;

   procedure Register_Plugin (Name  :  String;
                              Index :  Plugin_Index;
                              Tag   :  Ada.Tags.Tag);
end PPETP.Signature_Handlers;
```

Example

```
package PPETP.Signature_Handlers.Void is

   type Signer is new Abstract_Signer with private;

   overriding procedure Sign (Handler :  in out Signer;
                              Data    :  in out Data_Buffer);

end PPETP.Signature_Handlers.Void;
```

. . . meanwhile, in the body. . .

```
package body PPETP.Signature_Handlers.Void is
   ...
begin
   Register_Plugin (Name  => "void";
                    Index => 0;
                    Tag   => Signer'Tag);
end PPETP.Signature_Handlers.Void;
```

Plugin maps

```
generic
    type Root_Type (<>) is abstract new
        Abstract_Plugin
    with
        private;

    with function Create (Param :  not null access Plugin_Identifier)
        return Root_Type is abstract <>;
package PPETP.Plugins.Generic_Plugin_Maps is

    procedure Register_Tag (Name  :  String;
                            Index :  Plugin_Index;
                            Tag   :  Ada.Tags.Tag);

    function New_Handler (Index :  Plugin_Index) return Root_Type'Class;

    function New_Handler (Name :  String) return Root_Type'Class;

end PPETP.Plugins.Generic_Plugin_Maps;
```

Plugin searching

```
package PPETP.Plugins.Abstract_Plugin_Finders is

    type Abstract_Finder is abstract tagged private;

    type Finder_Access is access all Abstract_Finder'Class;

    procedure Search_Plugin
        (Finder :  in out Abstract_Finder;
         Class  :  in     Plugin_Class;     -- Signature, reduction, ...
         Name   :  in     String;
         Found  :     out Boolean)
    is abstract;
    -- Find and install the plugin

end PPETP.Plugins.Abstract_Plugin_Finders;
```

Finders are registered with

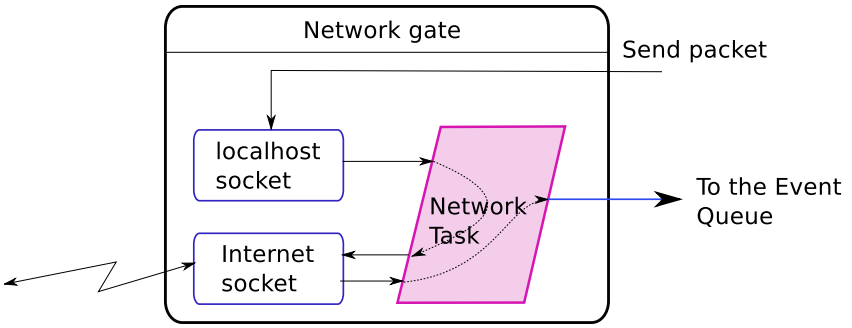
```
procedure Use_Finder (Finder :  Finder_Access);
```

Implementation Details

Other details

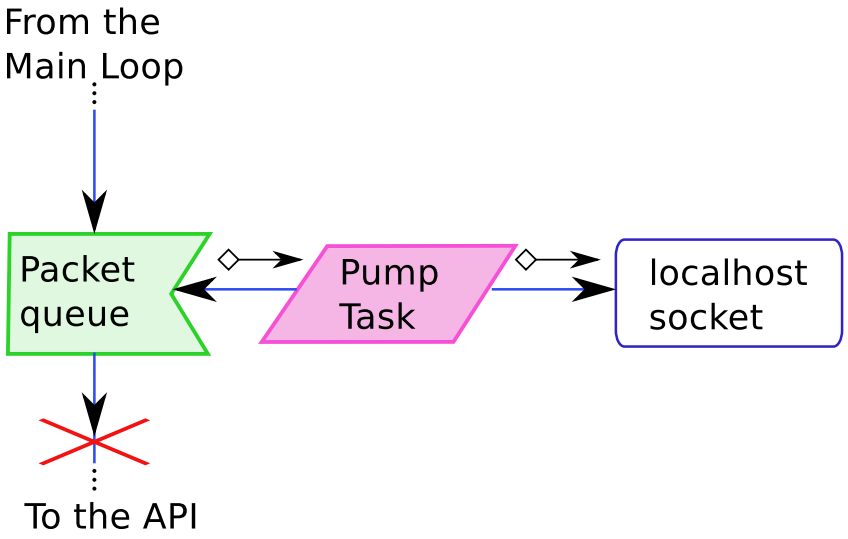
Network Gate

- Packets must be sent and read via the same UDP port
  - The socket is "owned" by the Network Gate
  - The Network Task waits on
    - An Internet socket
    - A localhost socket
  - Packets from Internet are sent to the Event Queue
  - Packets from localhost are sent to the Internet
  - Interface exposed: just a Send procedure



Conversion to BSD Socket

- Use a Pump Task started on request
- Content packets to localhost socket
- API disallowed to access the queue



Conclusions

Outline

- Introduction
- Problem Statement
- Overview of PPETP
- Internal details
- Applicative examples
- Conclusions
  - PPETP in a ~~nutshell~~ slide
  - Future developments

PPETP in a ~~nutshell~~ slide

- Aimed to
  - **Live streaming** to many users
  - Efficient use of available upload bandwidths (even small ones)
  - **Resilience** to packet losses, sudden departures and poisoning
- Characteristics
  - Runs over **UDP** (IPv4/IPv6)
  - Easy **integration** with existing protocols (RTP/RTSP/SDP/...)
    - No *preferred data type* (use it for everything)
    - Transport *separated* from network building (no preferred topology)
    - Looks like *multicast* from the application standpoint
  - **Plugin structure** for future extensions

The Future

- First public release (hopefully) soon
- Creation of an Open Source community
- Sheperd the I-D to RFC
- Side-software production, for example,
  - ▷ Integration of PPETP in VLC (gstreamer, . . .)
  - ▷ Wireshark dissector
  - ▷ Basic network mananger
  - ▷ . . .

That's all folks!

Applications

Outline

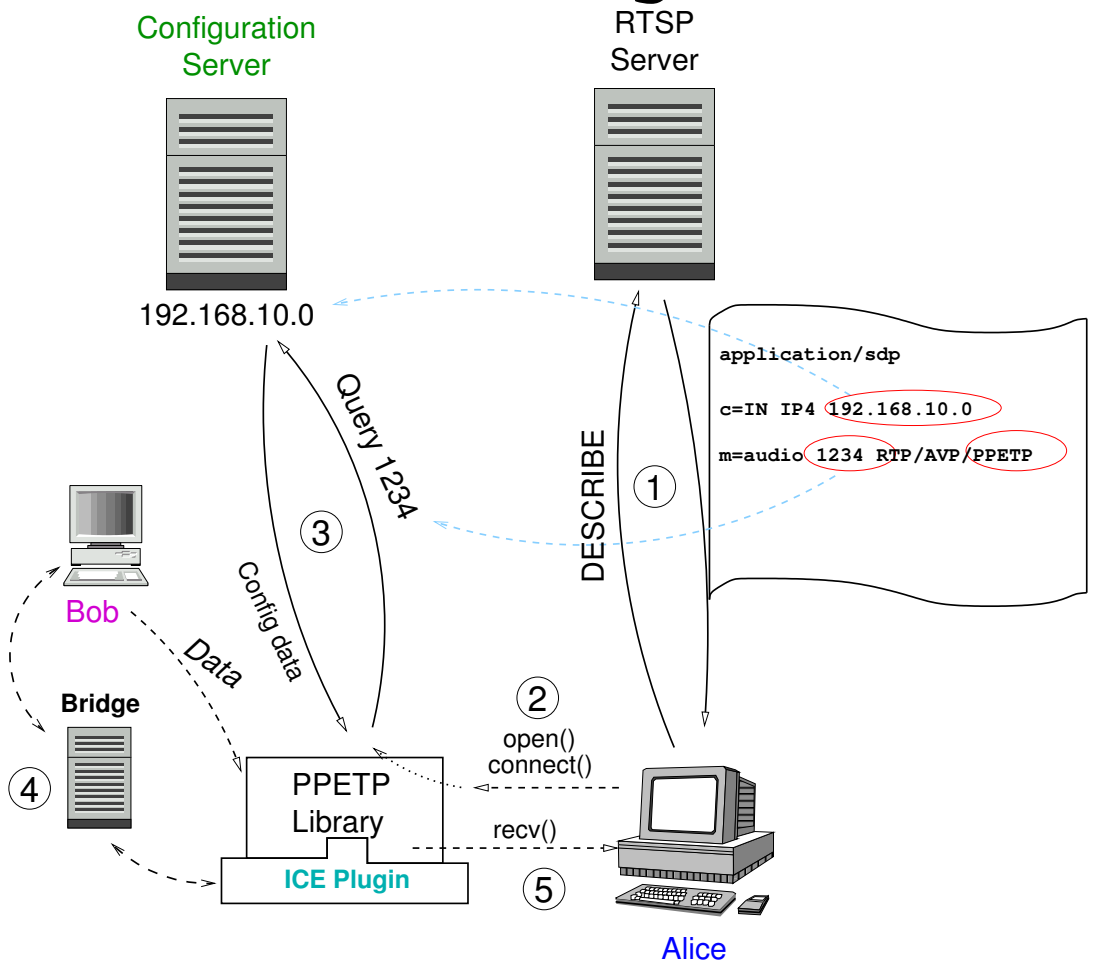
- Introduction
- Problem Statement
- Overview of PPETP
- Internal details
- Applicative examples
  - Live streaming
  - Conference
- Conclusions

Applications

Live streaming

Example of live streaming

1. *Alice* queries a RTSP server. The SDP reply specifies **PPETP** as transport. The address in the `c=` line and the port in the `m=` line makes the session **pseudo-address**
2. Alice opens a PPETP socket and does a `Join`
3. The PPETP library queries the **configuration server** that replies with configuration data, including the **generalized address** of the other peers (of ICE class)
4. The **ICE-plugin** does the ICE procedure with Bob.
5. Alice reads data from her local socket

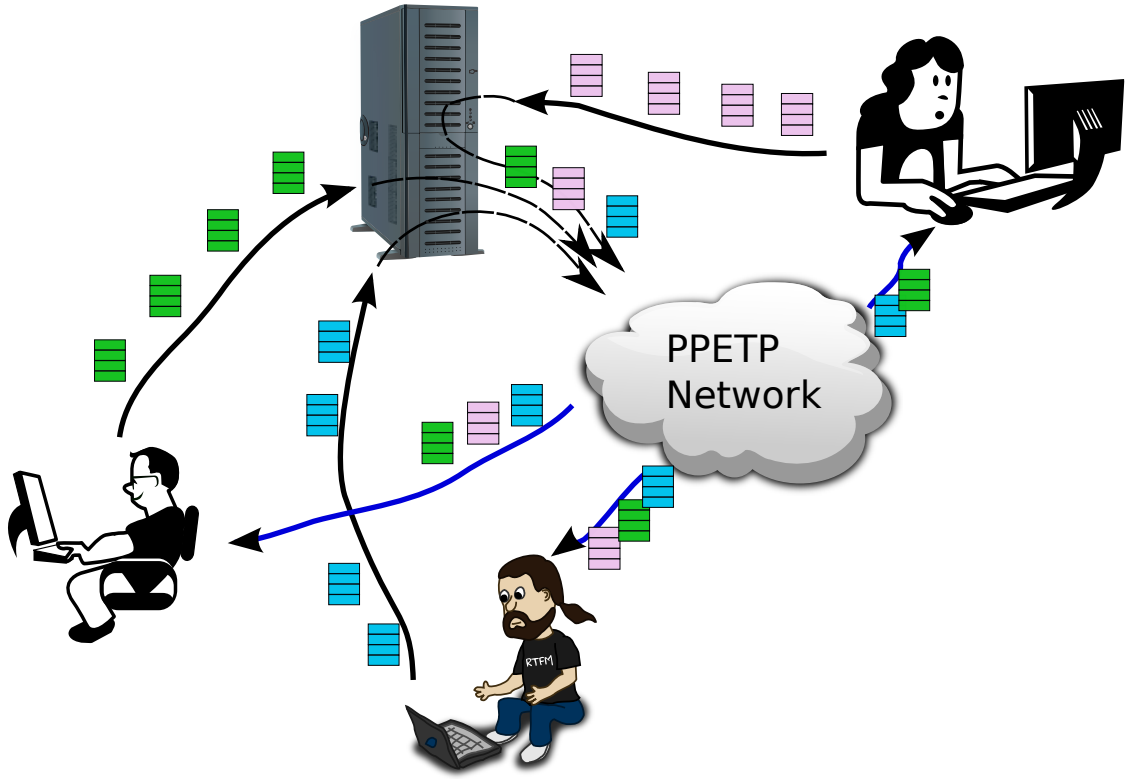


Applications

Conference

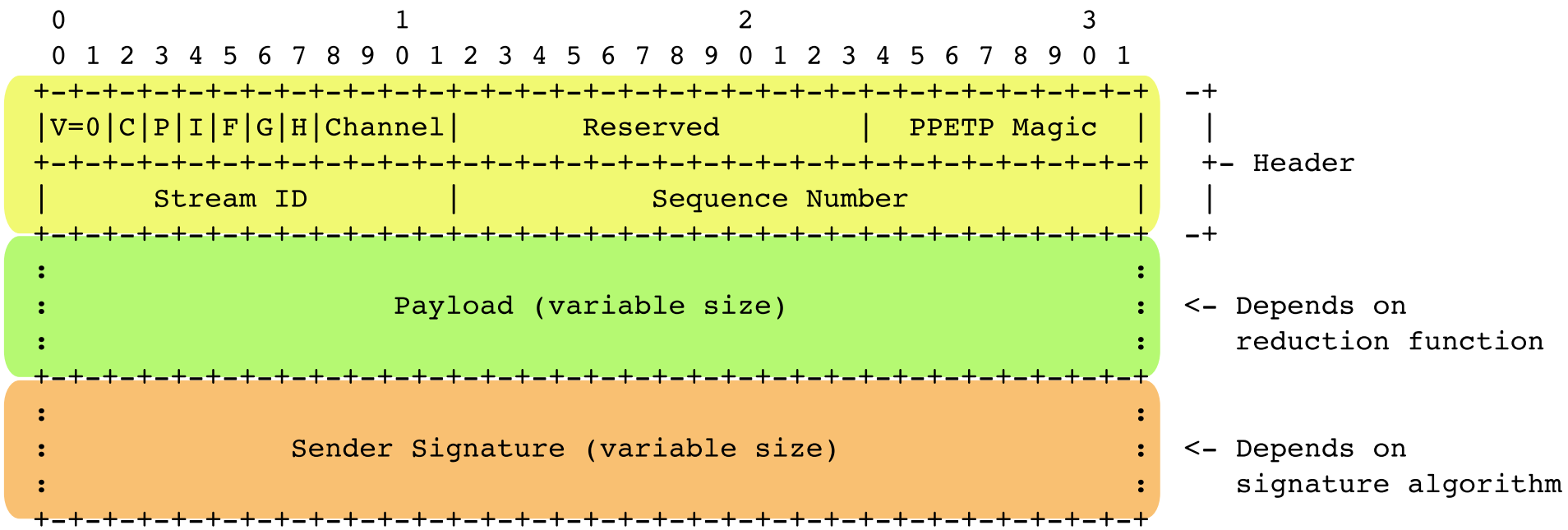
PPETP for conference

- Each participant sends its data in RTP packets to a **mirror node**
- The mirror replicates the data over **PPETP**
- Each participant joins the PPETP session
- Each participant uses the SSRC to discard its own packets



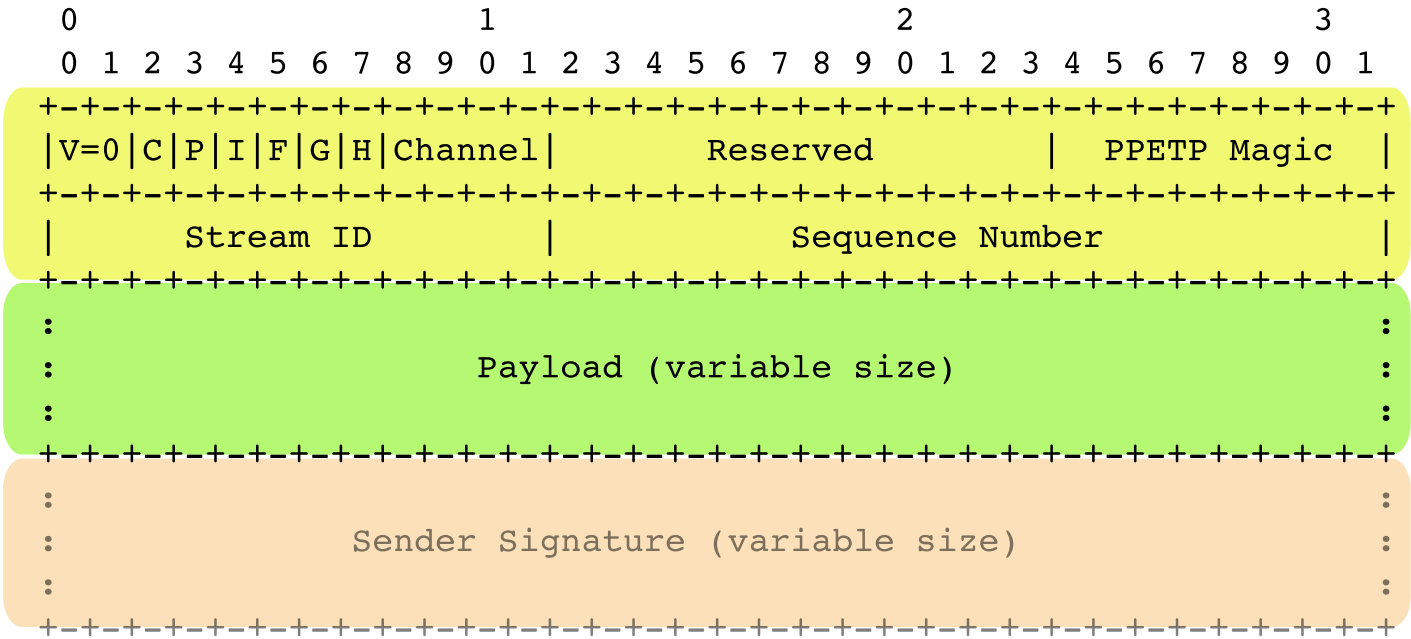
Data Formats

Data Packet Format



Data Packet Format (2)

After signature verification



Data Packet Format (3)

After header processing