UCLouvain

Pluginizing QUIC

Quentin De Coninck, **François Michel**, Maxime Piraux, Florentin Rochet,
Thomas Given-Wilson, Axel Legay, Olivier Pereira, Olivier Bonaventure

SIGCOMM 2019
UCLouvain, Belgium

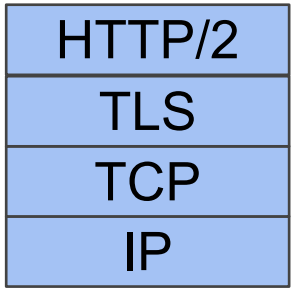
Wallonie

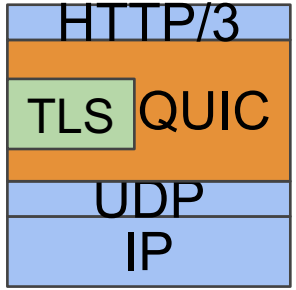
fnrs
FREEDOM TO RESEARCH

QUIC

- Currently under standardization at IETF
- Providing TCP/TLS1.3 service atop UDP
  - (Nearly) everything is encrypted
  - Reliable, in-order, multistream data transfer
- Frame-based protocol

- Large companies are deploying both client and server sides
  - Limited interoperability today





Application Requirements Evolve

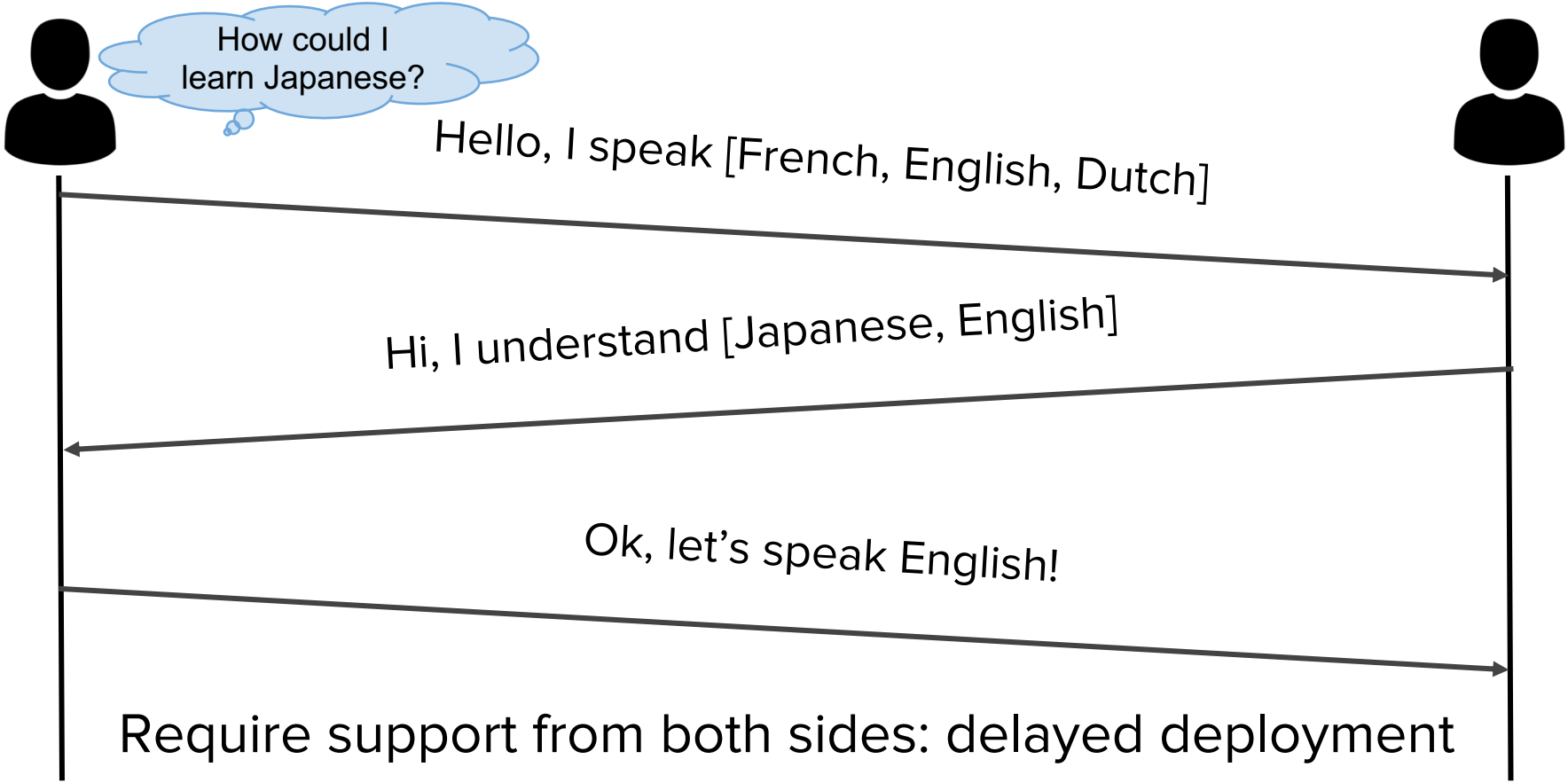
Simple reliable file transfer

Mobility, (live) video streaming

Latency critical, partially unreliable

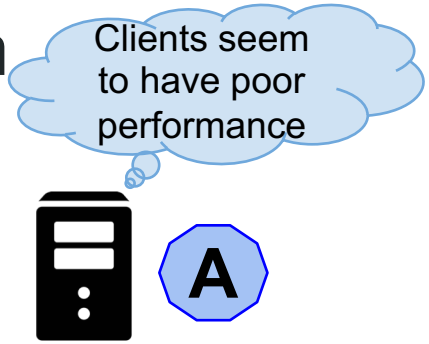
Transport protocols need to adapt!

Negotiating Extensions: The Human Protocol

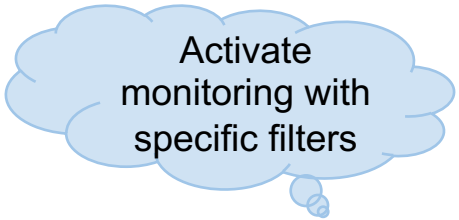


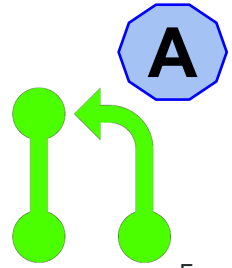
Require support from both sides: delayed deployment

Adding Monitoring to an Implementation



- High-performance, native server implementation
- Client side based on open-source 3rd-party library
  - **A** opens Pull Request to extend the library
    - Get monitoring information for some connections
  - But not merged by **B**





What If We Could Have...

A base, simple implementation enabling

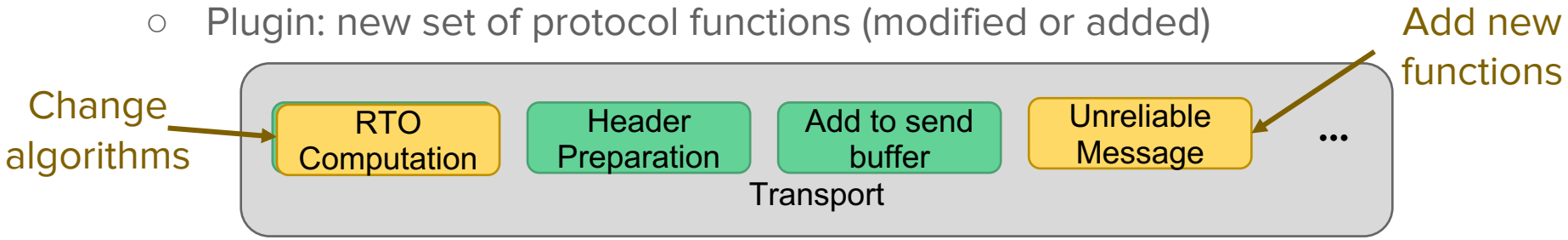
Personalized Transport Functionalities
Per-Connection Deployment

Agenda

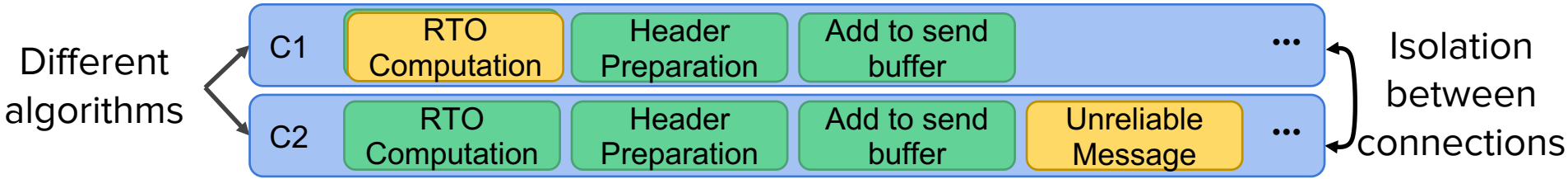
- Motivations
- **Pluginized QUIC Design**
- Evaluating PQUIC with Use Cases
- Conclusion and Future Works

Pluginized QUIC

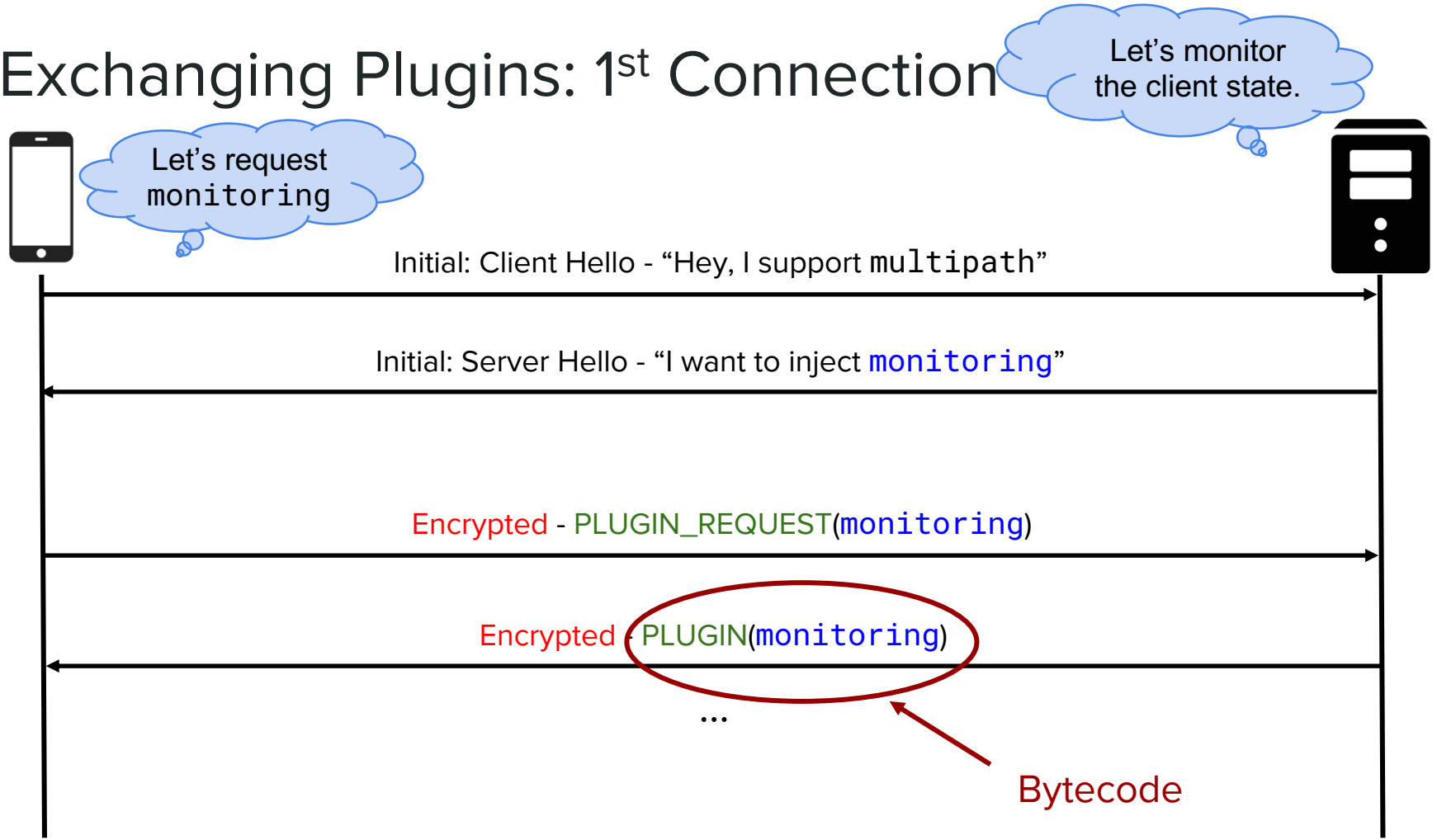
- Revisit the structure of protocol implementations
- Transport protocol = set of basic functions (protocol operations)
  - Plugin: new set of protocol functions (modified or added)



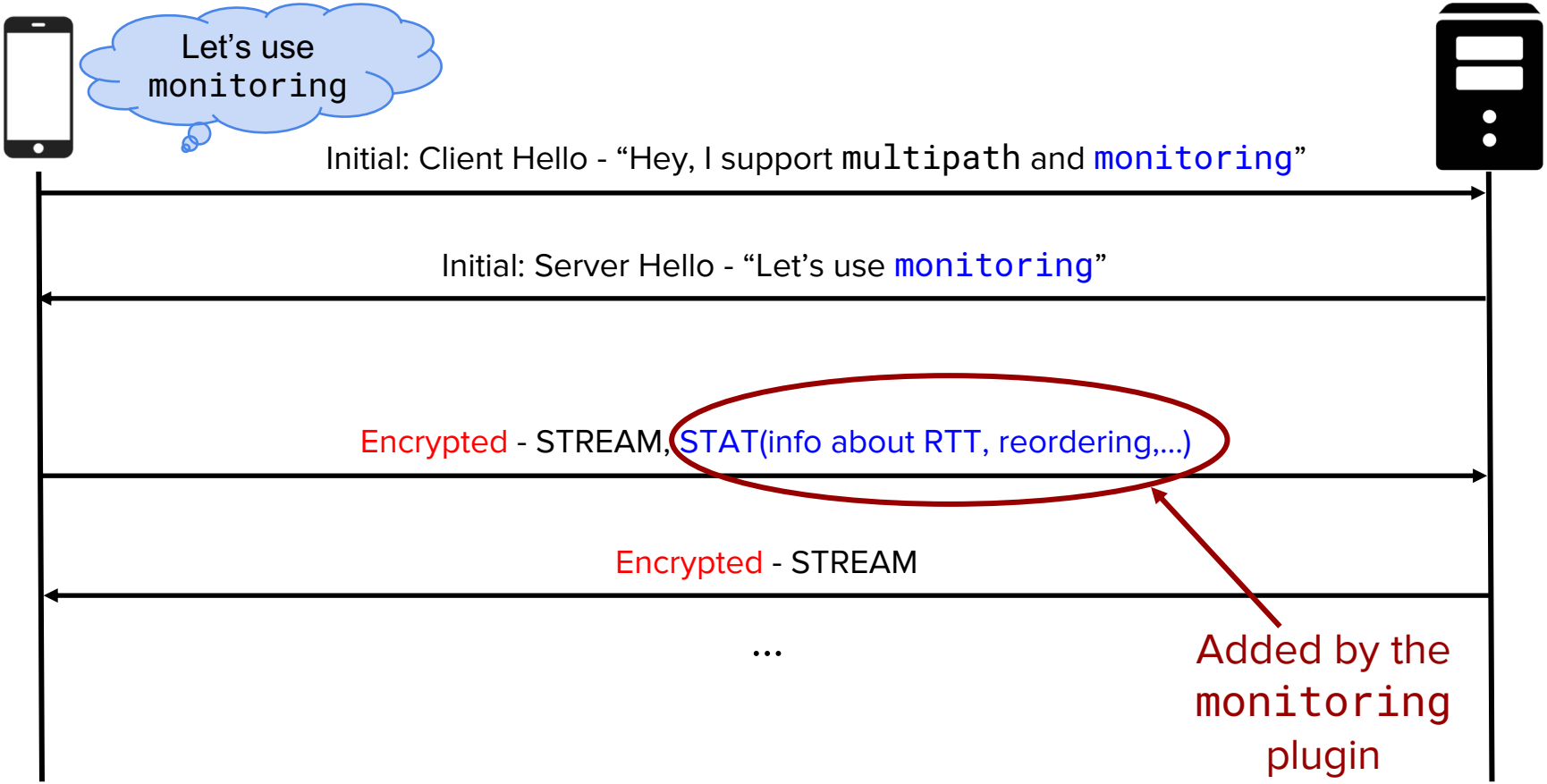
- Dynamic per-connection customization



Exchanging Plugins: 1st Connection



Exchanging Plugins: Next Connections



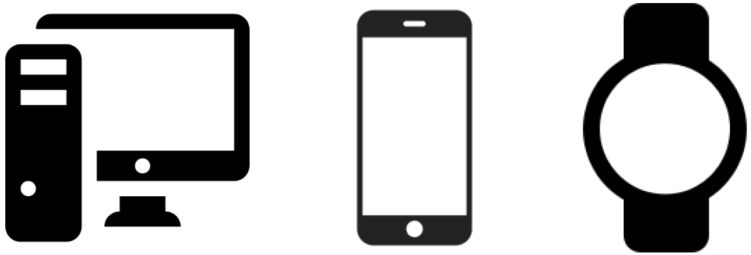
Revisiting Protocol Implementation Design

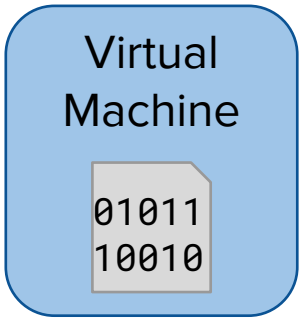
What do we need?

- Running plugins in a safe environment
- Identifying protocol operations
- Providing an API to the protocol plugins

Running Plugins

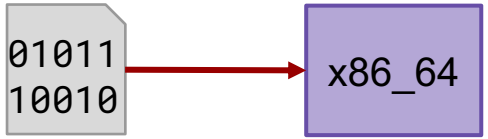
- Plugin = set of bytecodes
  - 1 bytecode ➜ 1 function
  - Hardware/OS independent
  - Kept under control
- Rely on a virtual machine
  - Isolate bytecode from host implementation



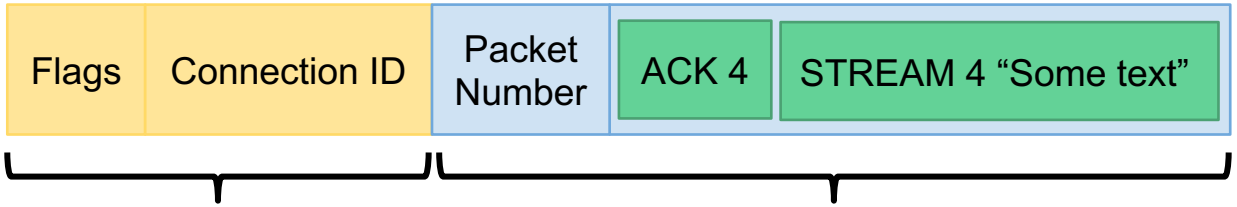


eBPF Virtual Machine

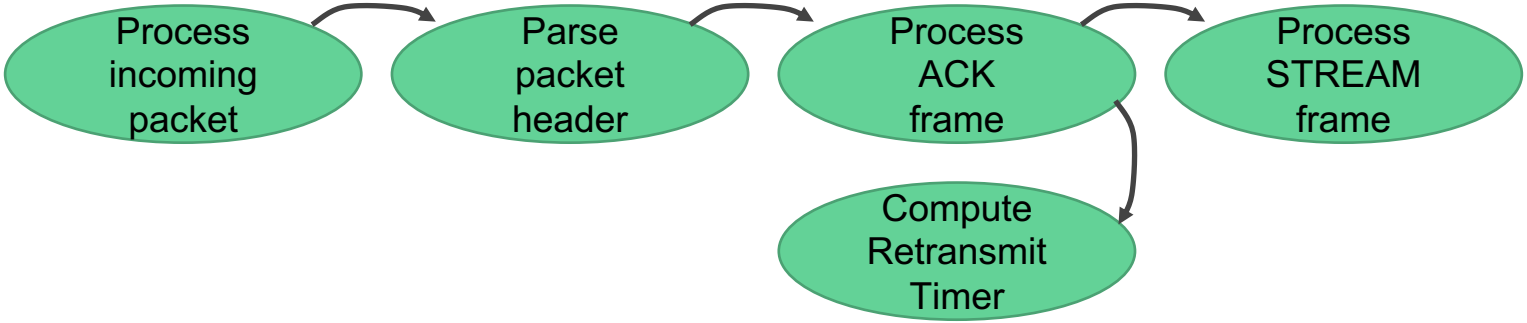
- Lightweight, integrated in Linux kernel since 2014
  - RISC instruction set (~100)
    - ALU, memory and branch purposes
- Bytecode recompiled to native architecture
- Verifier to ensure programs safety
- Dedicated, isolated stack memory
  - But no persistence
- Rely on a user-space implementation
  - With relaxed verifier
  - With persistent heap memory



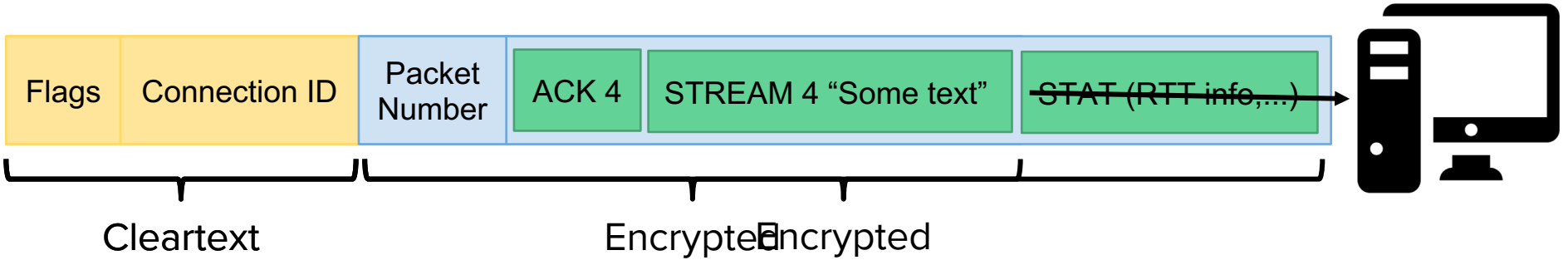
Identifying Protocol Operations



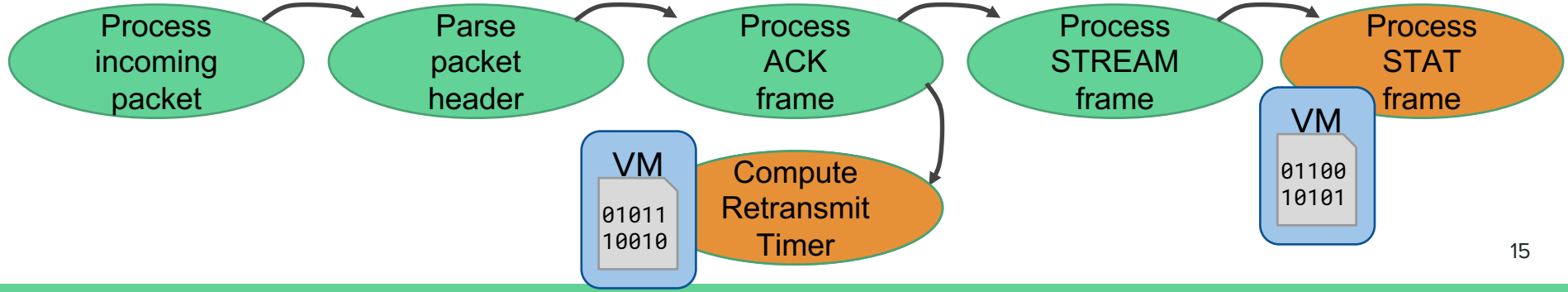
Each operation has its own built-in implementation



Modifying/Adding Protocol Operations



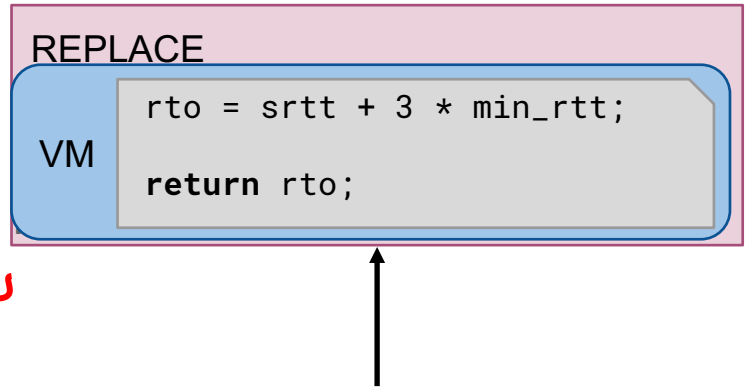
Each operation has its own built-in implementation



Changing the Retransmission Timer

```
int process_ack_frame(args) {

  // Some processing

  ret = compute_retransmission_timer(args);

  // Some processing

  return val;

}
```

```
int compute_retransmission_timer(args) {

}
```

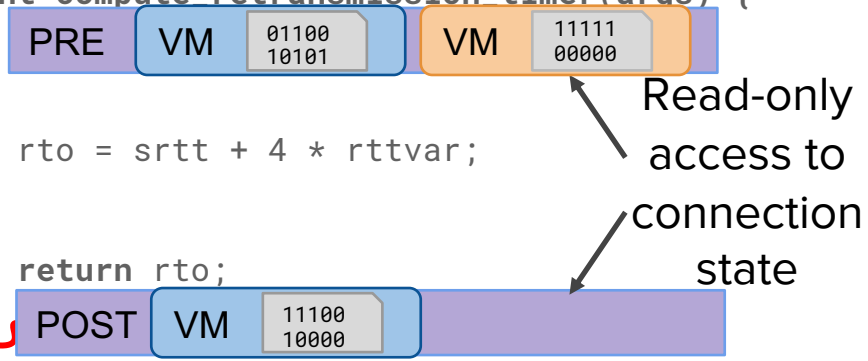


Exclusive full-access to connection state

Inserting Monitoring Plugin

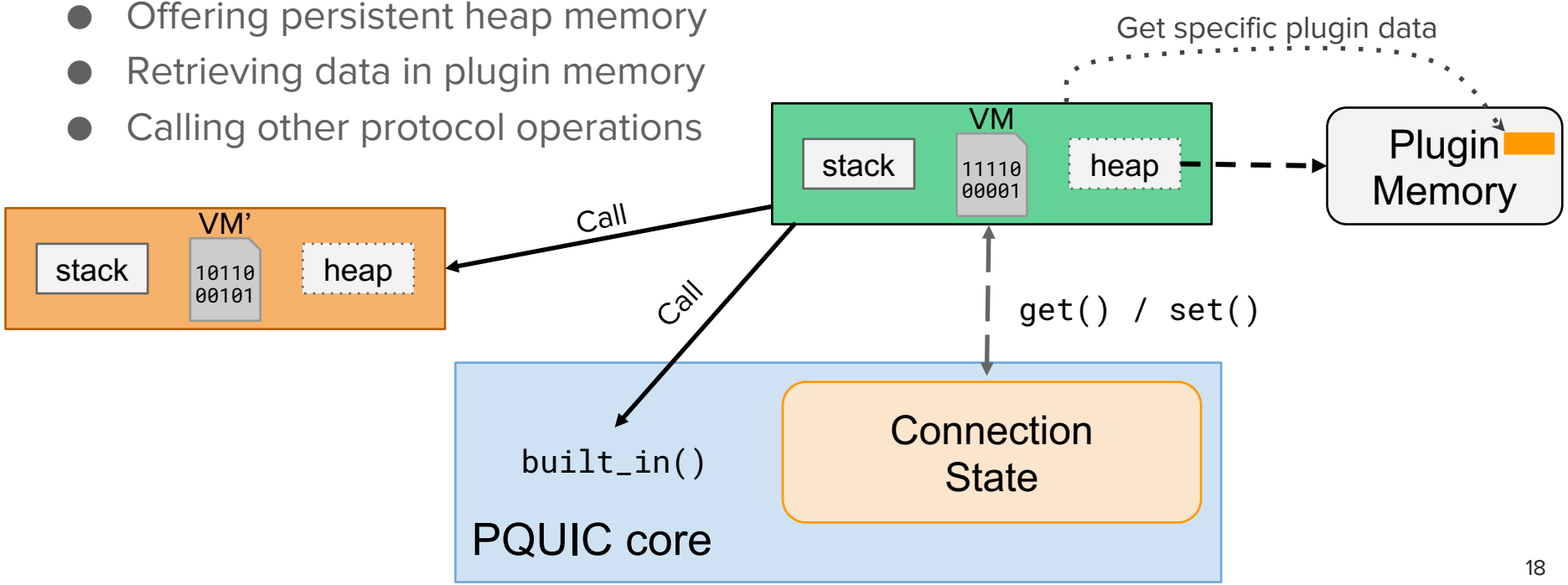
```
int process_ack_frame(args) {

  // Some processing

  ret = compute_retransmission_timer(args);



  // Some processing

  return val;

}
```

```
int compute_retransmission_timer(args) {

rto = srtt + 4 * rttvar;

return rto;
}
```

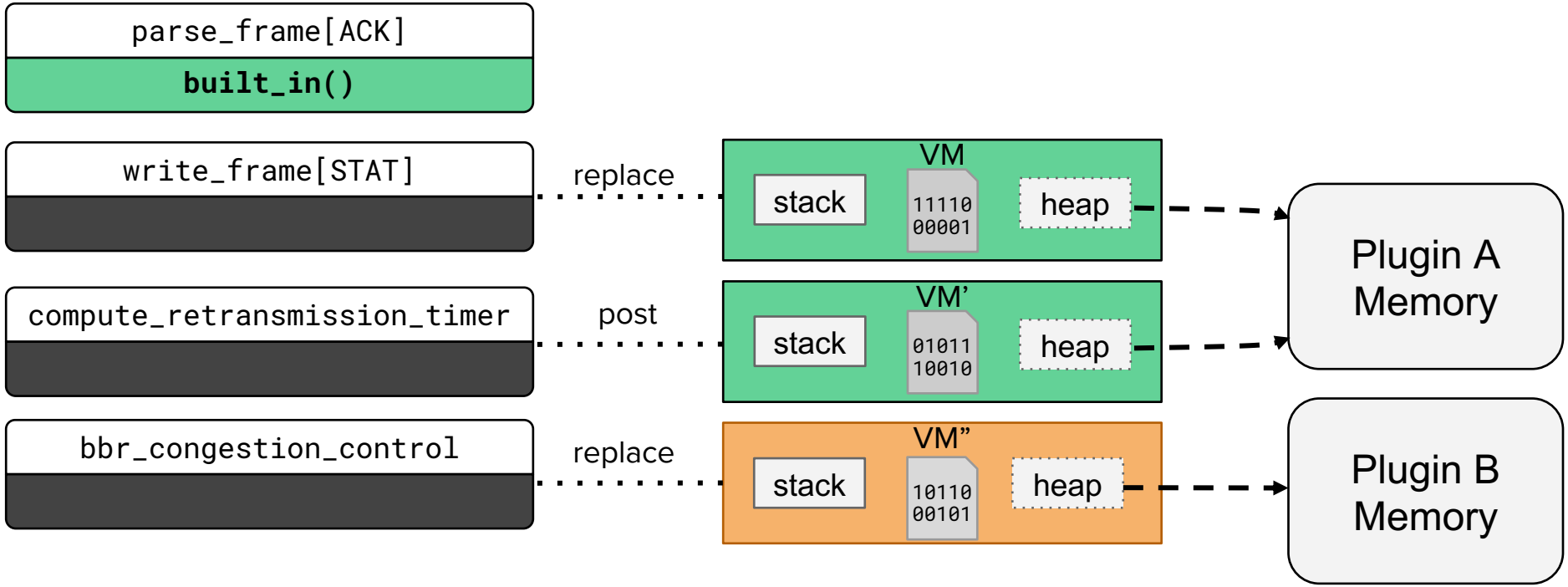


Linking Plugins to the Core Implementation

- Exposing connection fields
- Offering persistent heap memory
- Retrieving data in plugin memory
- Calling other protocol operations



Communication between Plugin Bytecodes



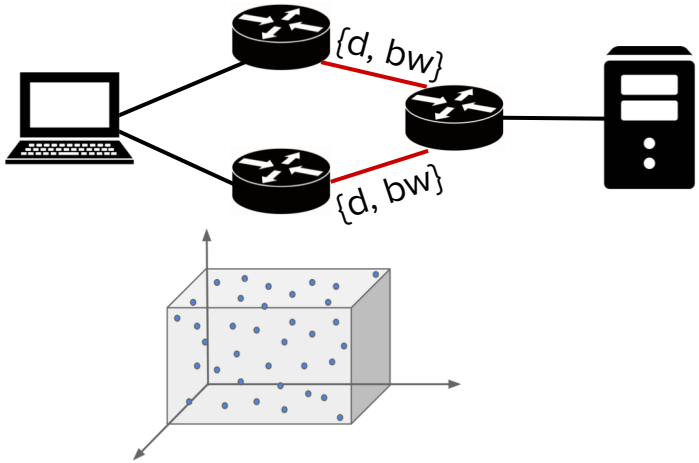
Agenda

- Motivations
- Pluginized QUIC Design
- **Evaluating PQUIC with Use Cases**
- Conclusion and Future Works

https://github.com/private-octopus/picoquic

Evaluation Overview

- Implementation based on `picoquic`
  - IETF QUIC in C language
- Evaluation in our lab
  - Use NetEm and HTB to configure links
- Experimental design
  - 139 points
  - Consider median over 9 runs



Metric	Minimum	Maximum
One-way delay (ms)	2.5	25
Bandwidth (Mbps)	5	50

Exploring Use Cases

- Monitoring
- A QUIC VPN
- Multipath
- Forward Erasure Correction

Plugin	Lines of C Code	Number of bytecodes
Monitoring	500	14
QUIC VPN	500	11
Multipath	2600	32
Forward Erasure Correction	2500	51

Exploring Use Cases

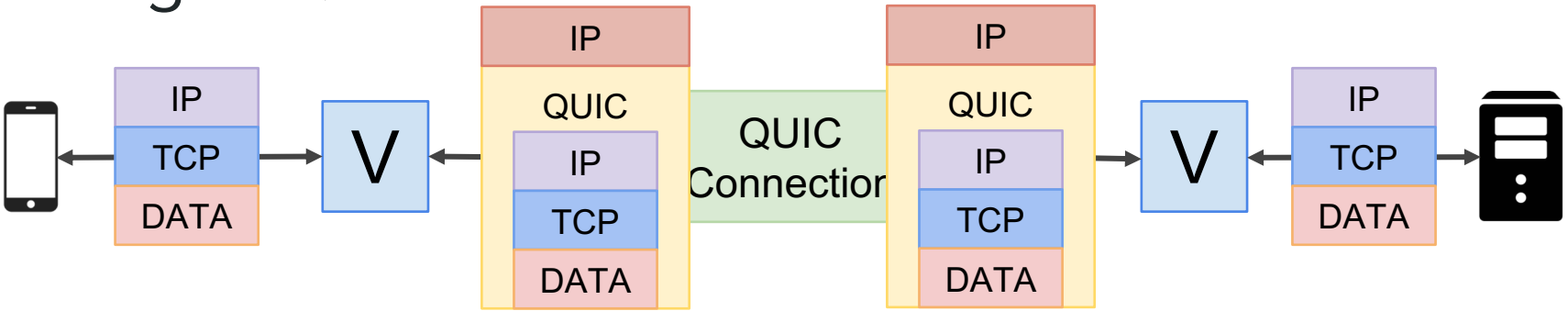
- Monitoring
- **A QUIC VPN**
- **Multipath**
- Forward Erasure Correction

Read paper for other use cases

Plugin	Lines of C Code	Number of bytecodes
Monitoring	500	14
QUIC VPN	500	11
Multipath	2600	32
Forward Erasure Correction	2500	51

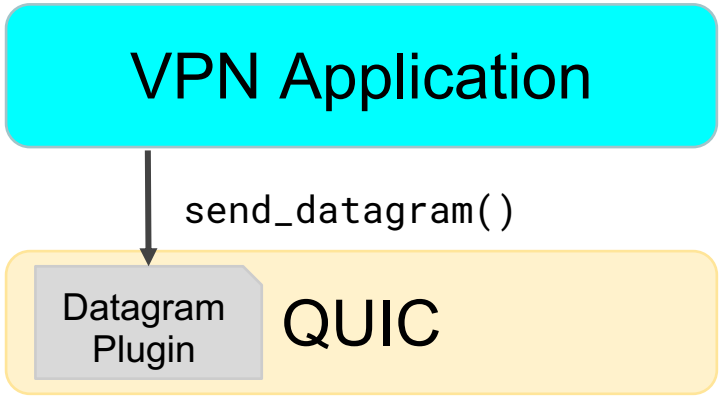
1st Plugin: QUIC VPN

Alternative to DTLS, IPsec



➔ But QUIC provides **reliable**, **in-order** data delivery

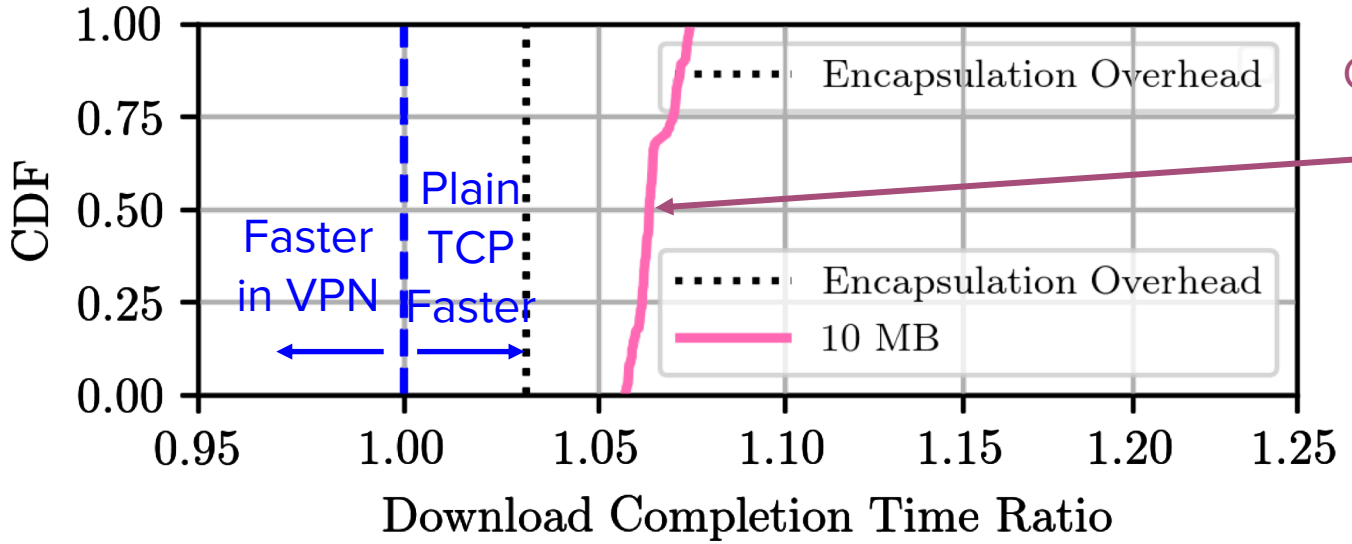
- Datagram plugin
  - **Unreliable** data delivery
  - Application API
    - Protocol operation



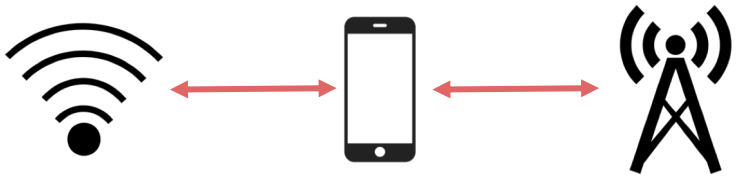
QUIC VPN in Action

Compare download time for a single file transfer using TCP_{Cubic} over 1 path

$$\text{Ratio} = \frac{\textit{Download Completion Time Inside the QUIC VPN}}{\textit{Download Completion Time with plain TCP}}$$

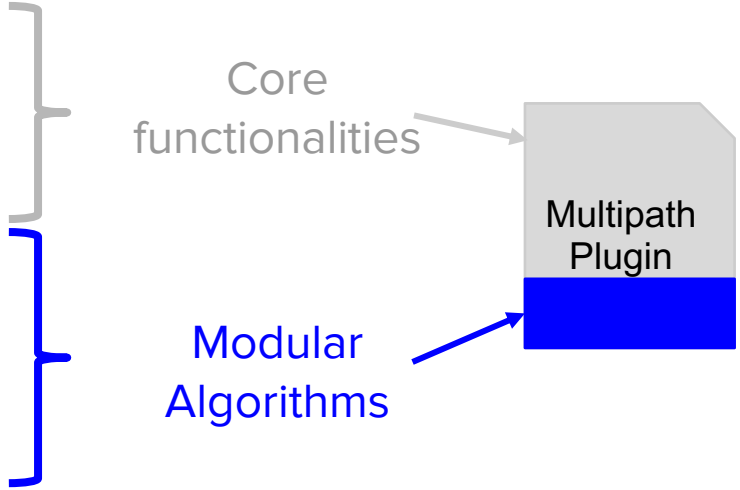


2nd Plugin: Multipath QUIC



Implement the IETF Multipath draft

- Link paths to specific networks
  - Connection not bound to 4-tuple
- Exchange IP addresses
- Create paths over pair of IP addresses
  - Full-mesh
- Schedule packets over paths
  - Round-robin fashion



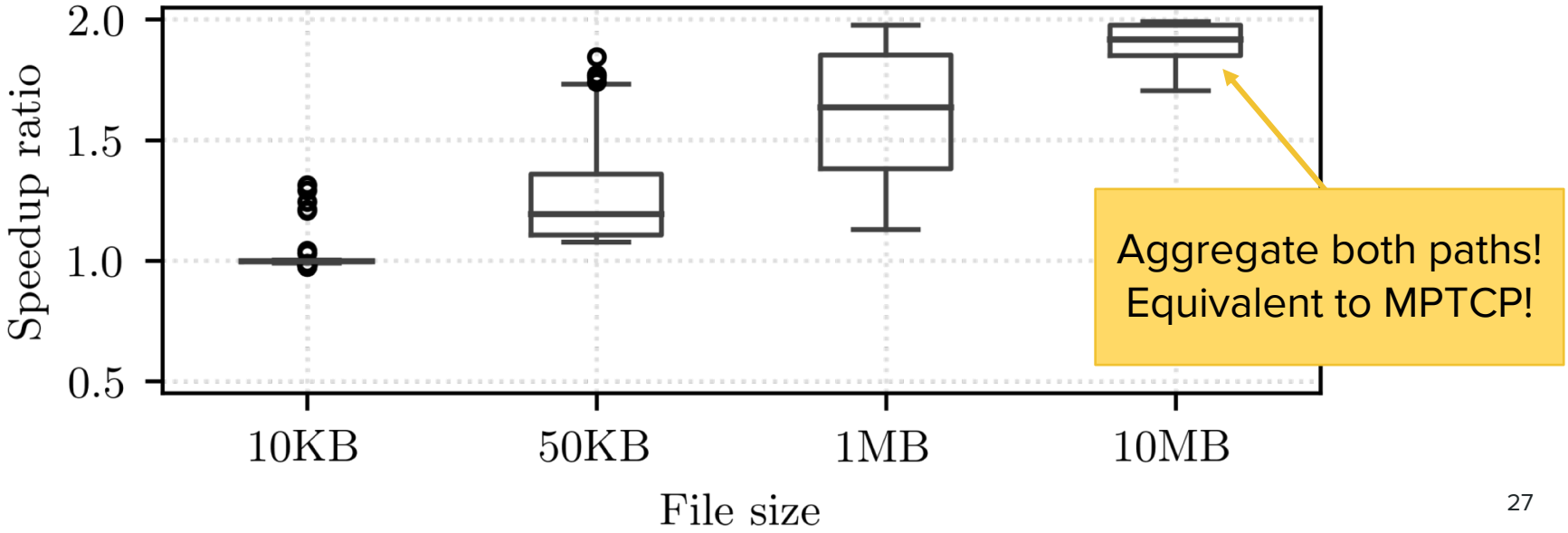
https://datatracker.ietf.org/doc/draft-deconinck-quic-multipath/

Multipath QUIC in Action

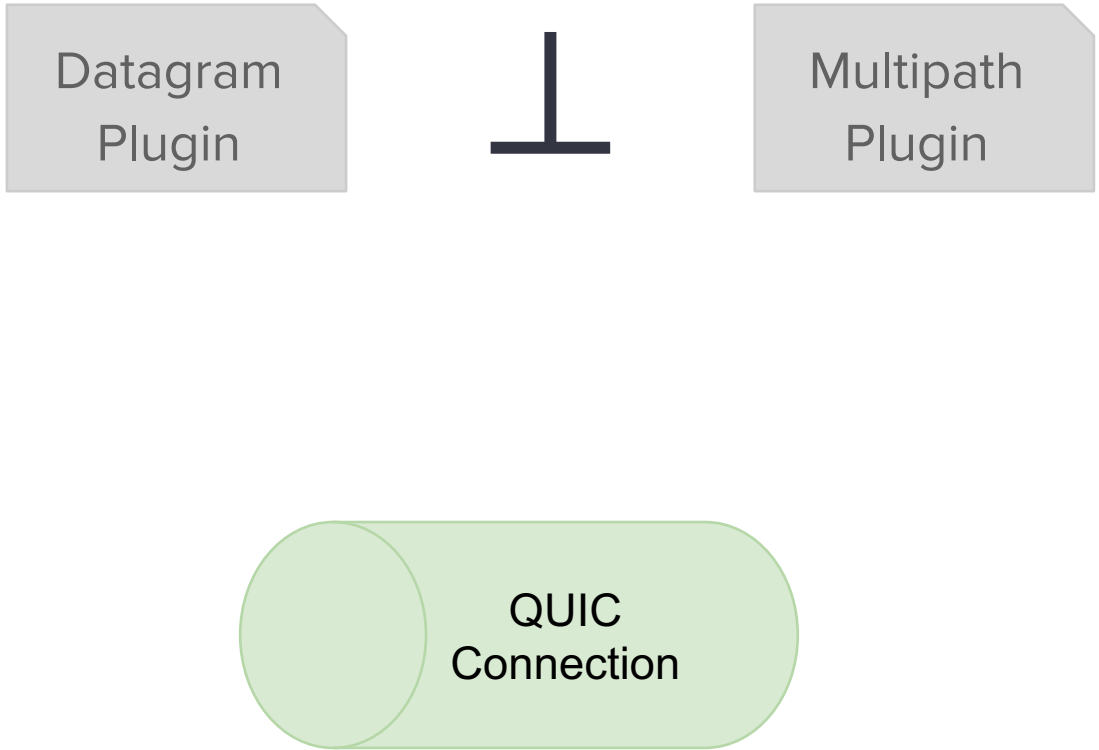
$$\text{Speedup Ratio} = \frac{\textit{Download Completion Time Single Path QUIC}}{\textit{Download Completion Time Multipath QUIC}}$$

GET request over a dedicated stream

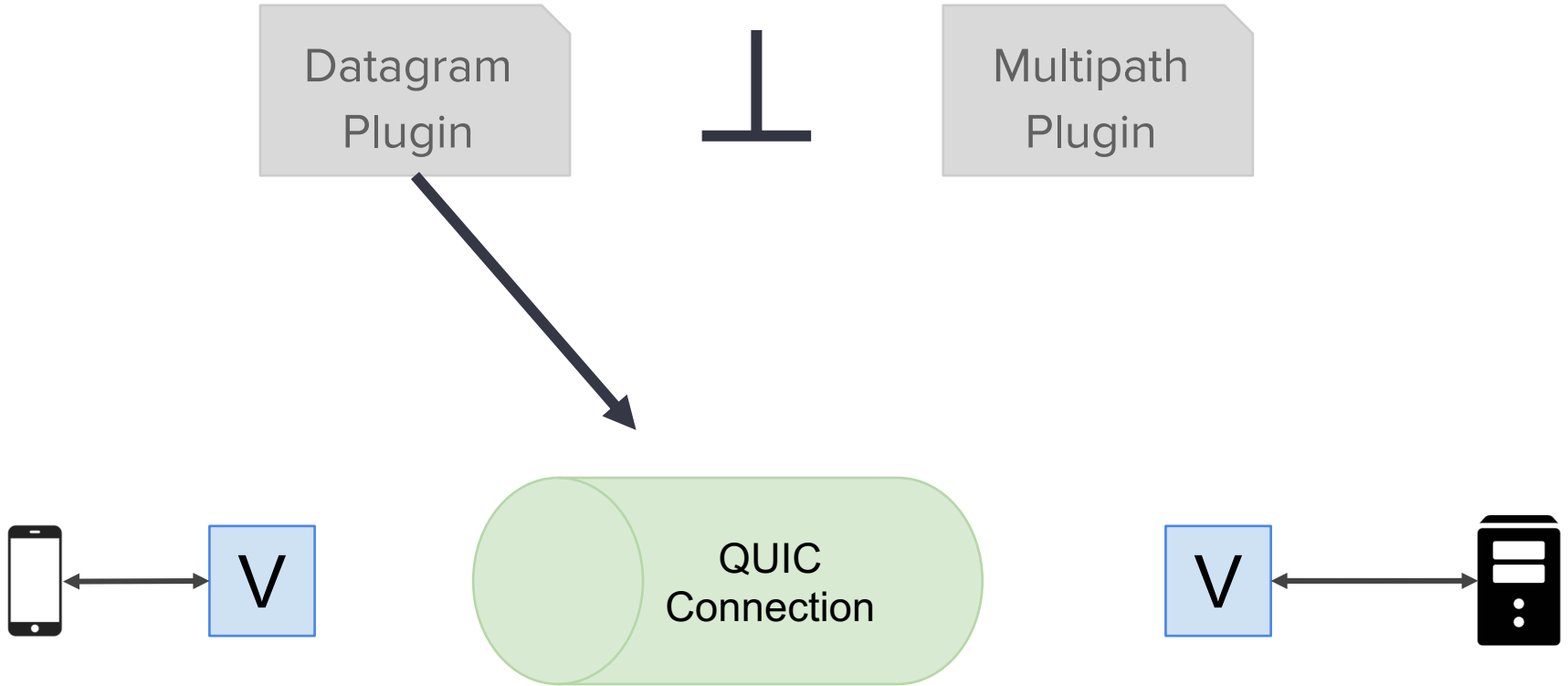
- Consider 2-path network with **symmetric characteristics**



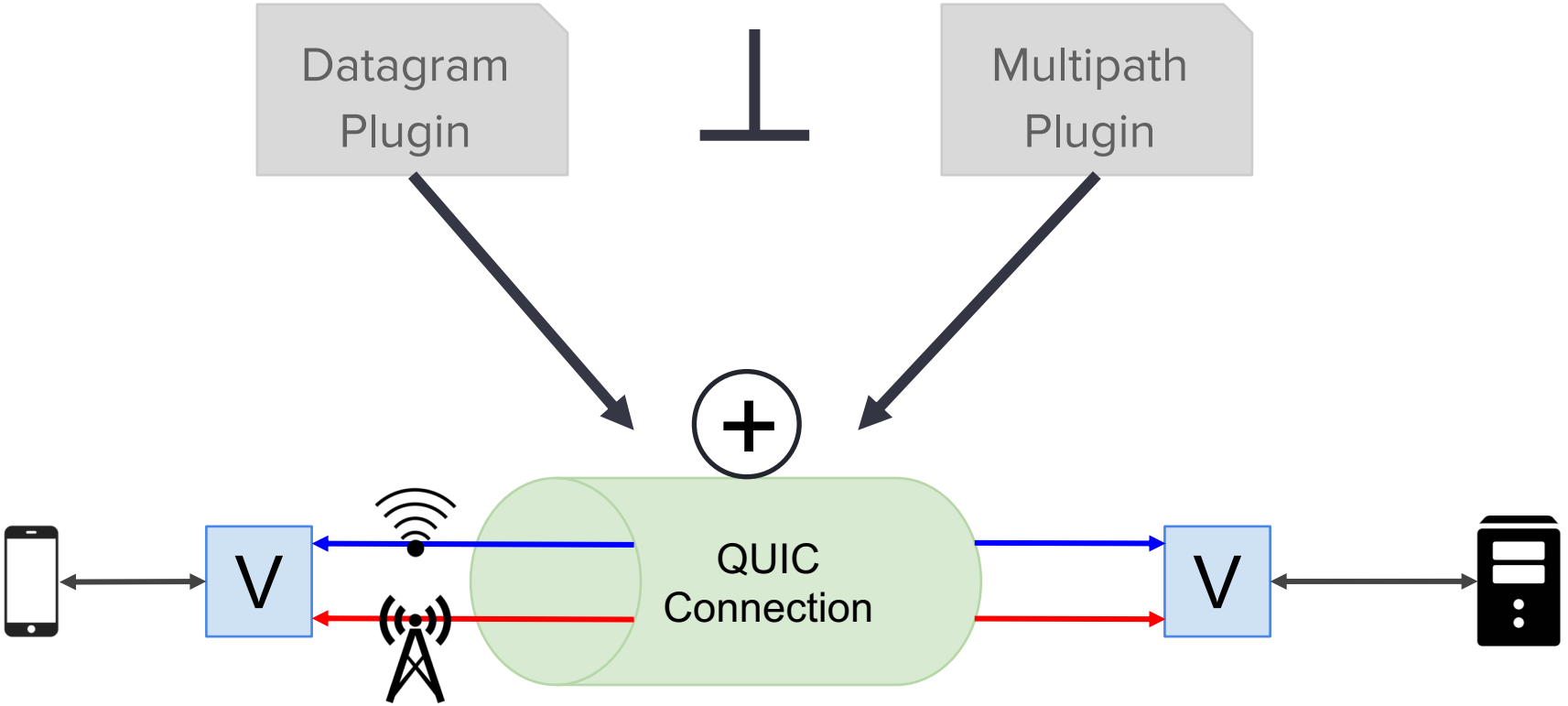
Combining Plugins into a Multipath QUIC VPN



Combining Plugins into a Multipath QUIC VPN



Combining Plugins into a Multipath QUIC VPN

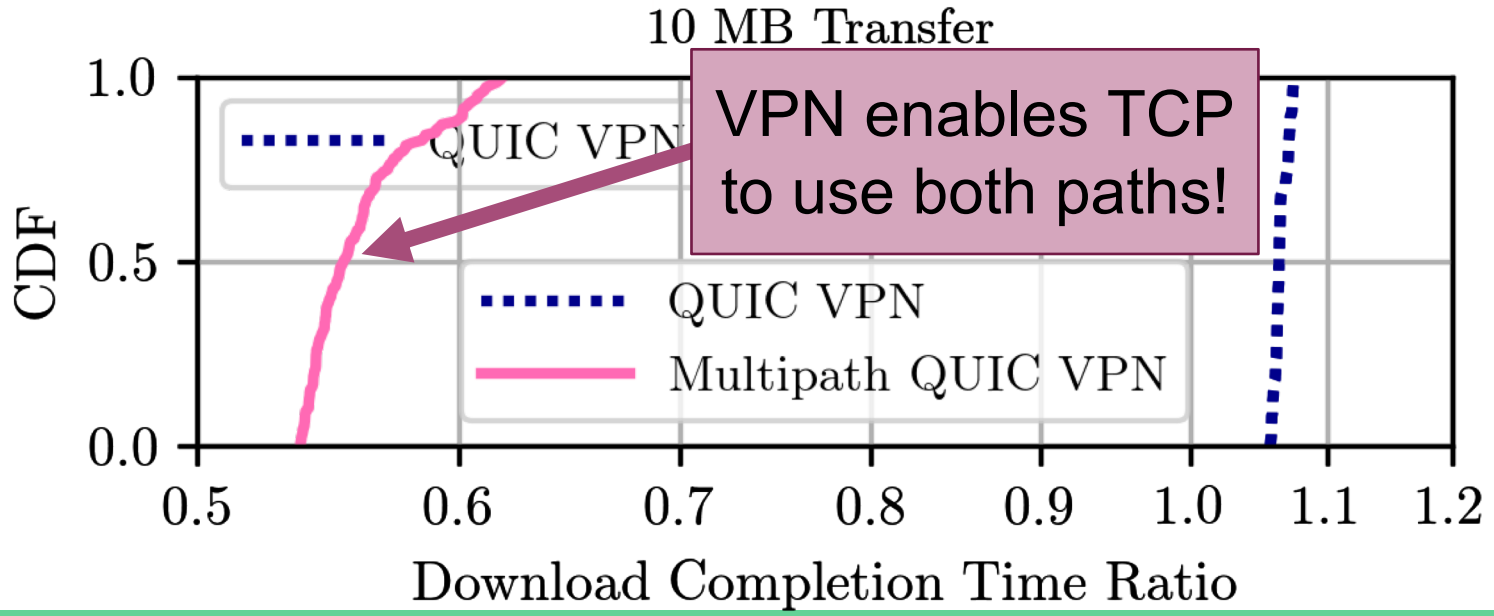


Multipath QUIC VPN in Action

Compare download time for a single file transfer using TCP_{Cubic}

- 2 **symmetric** network paths

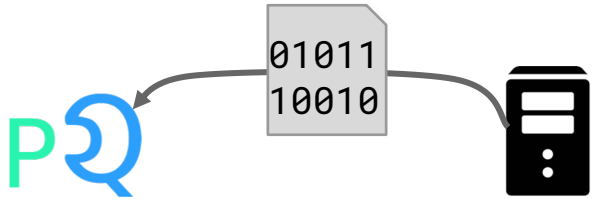
$$\text{Ratio} = \frac{\textit{Download Completion Time Inside the QUIC VPN}}{\textit{Download Completion Time with plain TCP}}$$



Agenda

- Motivations
- Pluginized QUIC Design
- Evaluating PQUIC with Use Cases
- **Conclusion and Future Works**

Conclusion and Future Works



- **PQUIC**: Dynamically extends protocol implementations
- Common **pluginizable client**, **specialized server** implementation
  - Possible through secure plugin exchange
- Four different use-cases **fully implemented with plugins**
  - Monitoring, VPN, Multipath, Forward Erasure Correction
- Future works
  - Approach applicable to **many other protocols**
  - How can we have **independent, interoperable PQUIC implementations**?
    - Specification of the VM, protocol operations, API,...

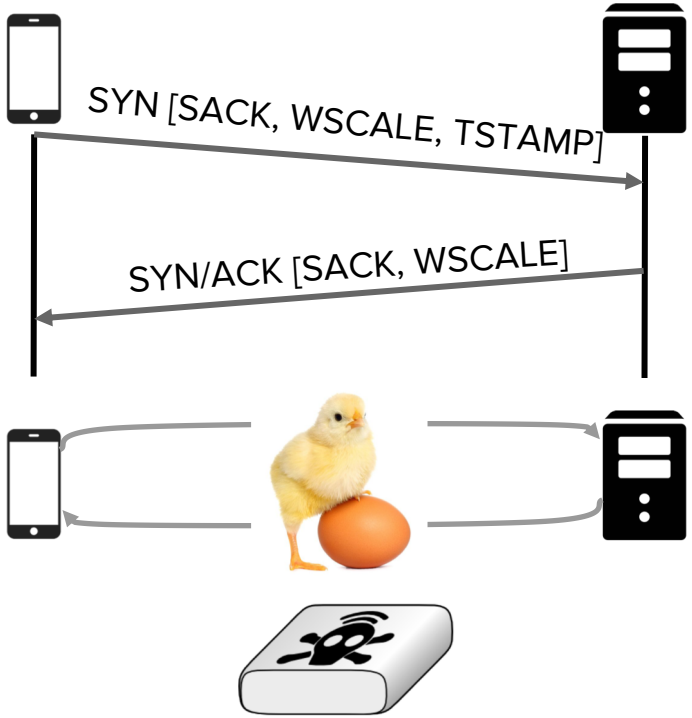
https://pquic.org

Thanks for your attention!

https://pquic.org

Extending/Customizing TCP Is Hard

- TCP Option Negotiation
- Extension deployment challenges
  - Specification process (IETF)
  - Implementation puzzle
    - Client waits for server support
    - Server waits for client support
  - Middlebox interferences
- Coarse protocol tuning
  - Global parameters (sysctl,...)
  - Very specific socket options



Processing New STAT frames

Type of the frame
=
Parameter to a **generic** protocol operation

STAT	Frame Fields (ex. used for RTT estimation)

```
int process_frame(args) {

  for each frame {

    ret = process_frame_param(frame.type, args);

    // Some processing

  }

  return val;

}
```

```
int process_stat_frame(args) {
```

PRE

REPLACE

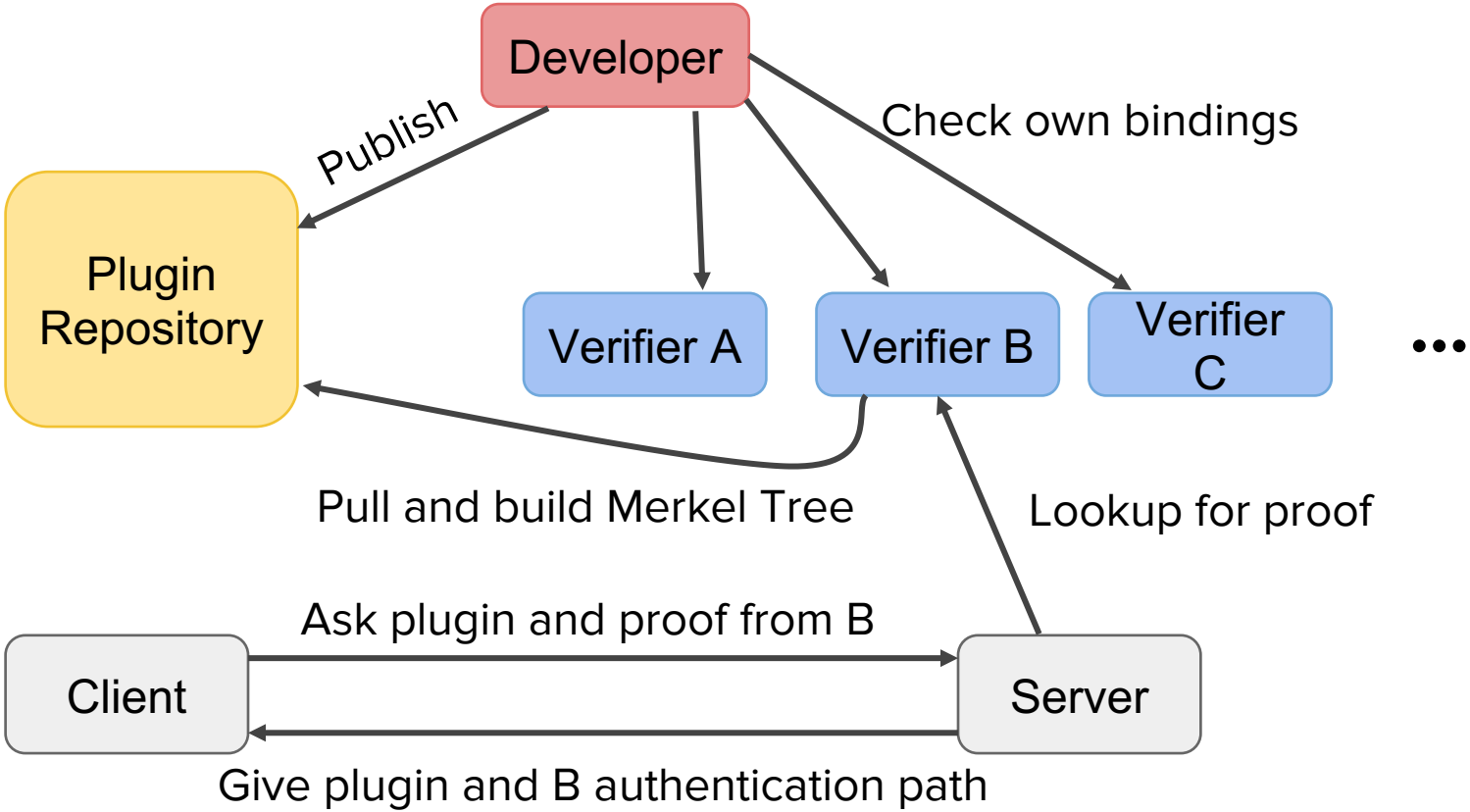
VM

```
// Code processing STAT

return ret;
```

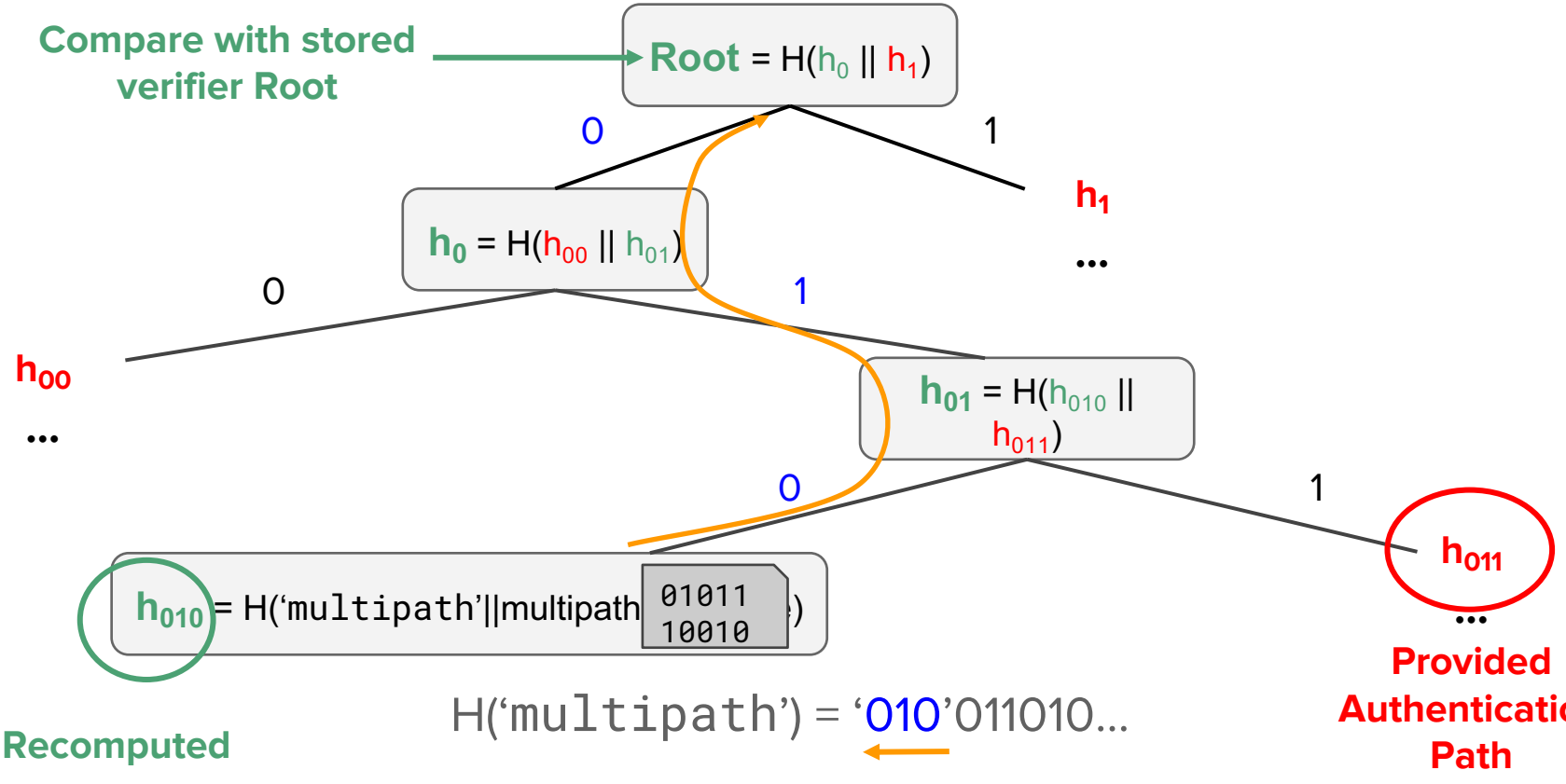
POST

```
}
```

Trusting Plugins

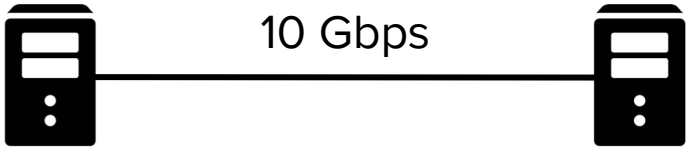


Proof of Consistency



Plugin Overhead

Intel Xeon E5-2640 v3



Plugin	Mean Goodput
PQUIC, no plugin	1104 Mbps
Monitoring	1037 Mbps
Multipath 1 path	757 Mbps
Monitoring + Multipath 1 path	714 Mbps

-7% goodput
+8% CPU instructions

JITed eBPF ~2x slower than native code
Get/set interface ~5x slower than direct access

Verifying the Termination of Plugins with T2

Plugin	Lines of C Code	# Bytecodes	Proven terminating
Monitoring	500	14	13
VPN	500	11	8
Multipath	2600	32	29
FEC	2500	51	37