

eBPF-Assisted Relays for Multimedia Streaming

Daniel Alexander Antonius Pfeifer

Technical University of Munich

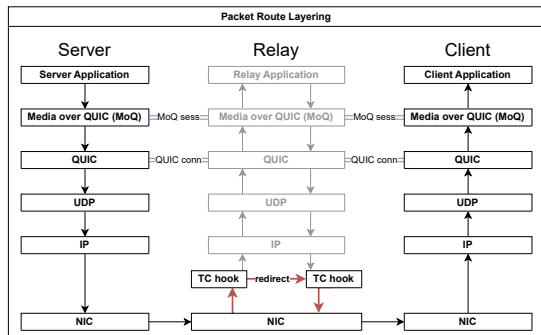
August 16, 2024

- 1 Introduction
- 2 QUIC and eBPF
- 3 Fast-Relays
- 4 Testing and Results
- 5 Conclusion and Future Work

- 1 Introduction
- 2 QUIC and eBPF
- 3 Fast-Relays
- 4 Testing and Results
- 5 Conclusion and Future Work

Motivation

- Shorten Critical Path
- Avoid Network Stack Traversal
- Reduce Forwarding Delay



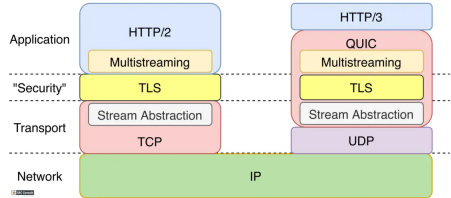
Research Question

- *Improve relay performance by using eBPF technology?*
 - *Remove userspace packet-processing from critical path?*
 - *Handle packet en- and decryption?*
 - *Communication between userspace and the eBPF program?*
 - *Generalize to support other protocols?*

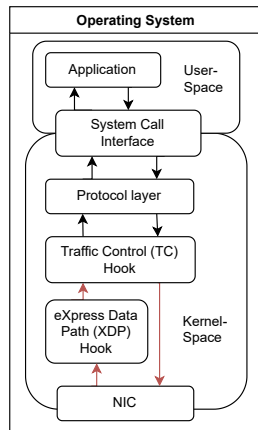
- 1 Introduction
- 2 QUIC and eBPF**
- 3 Fast-Relays
- 4 Testing and Results
- 5 Conclusion and Future Work

QUIC

- Started by Google as *Quick UDP Internet Connections*
- Standardized by IETF
- Fast Development Cycle since Userspace Implementation
- Gets rid of Issues like Head-of-Line Blocking



- Kernel-Internal Virtual Machine
- Used for Packet Filtering and Tracing
- Multiple Hook-Points in the Kernel (e.g. XDP and TC)
- Userspace Communication via Maps



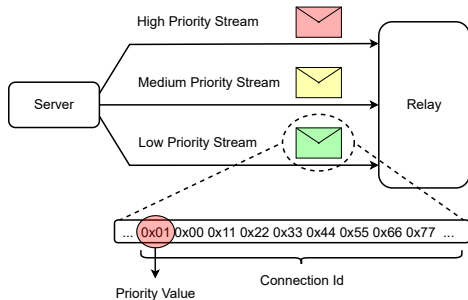
- 1 Introduction
- 2 QUIC and eBPF
- 3 Fast-Relays**
- 4 Testing and Results
- 5 Conclusion and Future Work

QUIC Adaptations

- Turn off en- and decryption
- Priorities for packets
- Public endpoint for packet registration
- Function pointer additions for eBPF state handling
 - Relay developer defines functions for eBPF map access
 - Called within quic-go if defined

Packet Priorities

- One priority per stream
- Saved in connection-id
- Additional connection-id retirement constraint



Public Endpoint for Packet Registration

```
1 go func(conn quic.Connection) {  
2     /* ... */  
3     for {  
4         /* ... */  
5         packet := common.RetrieveNextPacketFromMap()  
6         conn.RegisterBPFPacket(packet)  
7         /* ... */  
8     }  
9 }(conn)
```

Listing 1: Packet registration within relay code.

Function Pointer Additions

```
1 /* Function pointer call within actual quic-go code */
2 if packet_setting.ConnectionIdUpdateBPFHandler != nil /* &&
   potentially other conditions */ {
3     packet_setting.ConnectionIdUpdateBPFHandler(connId.Bytes(),
         uint8(connId.Len()), p.connection)
4 }
```

Listing 2: Function-pointer addition to the quic-go library.

```
1 /* Function pointer signature definition within additional
   config file */
2 ConnectionIdUpdateBPFHandler func(id []byte, l uint8, conn
   QuicConnection) = nil
```

Listing 3: The signature will be defined within the library itself.

Function Pointer Additions

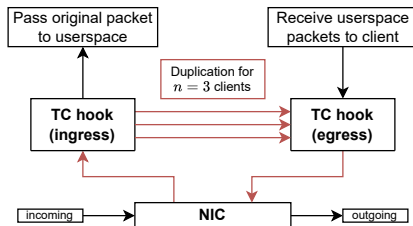
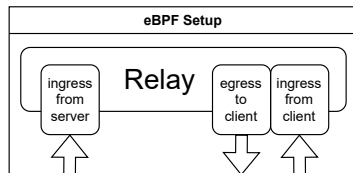
```
1 /* Definition of the function within the local relay code */
2 func localUpdateConnectionId(id []byte, l uint8, conn
   packet_setting.QuicConnection) {
3     /* handle the connection update by interacting with the eBPF
       program */
4 }

5
6 /* Providing the function to the quic-go library */
7 func main() {
8     /* ... */
9     packet_setting.ConnectionIdUpdateBPFHandler =
       localUpdateConnectionId
10    /* ... */
11 }
```

Listing 4: An example of how the addition looks on the relay side.

eBPF Setup

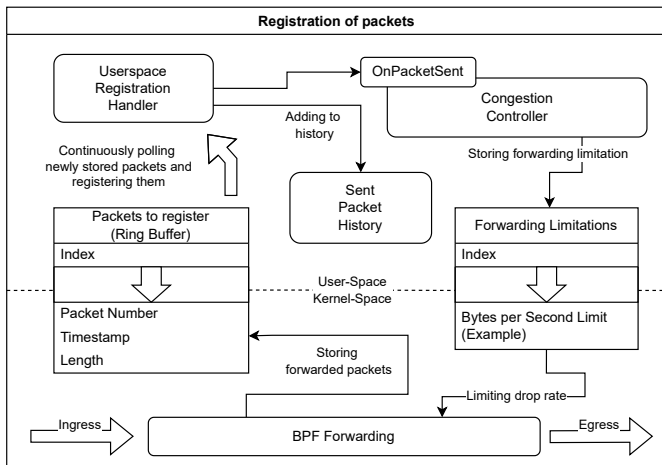
- Three eBPF Programs
 - Client ingress (client registration)
 - Server ingress (packet duplication and forwarding)
 - Client egress (state management)



Userspace Synchronization

- Number of clients
- Connection state (e.g. connection-id, id-translations, etc.)
- Incoming packet information (e.g. timestamp, etc.)
- Priority drop threshold for a connection
- Congestion control updates

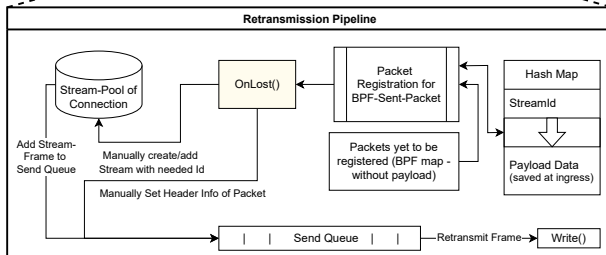
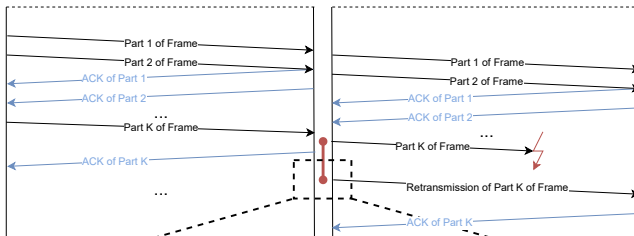
Userspace Synchronization cont.



Packet Retransmission

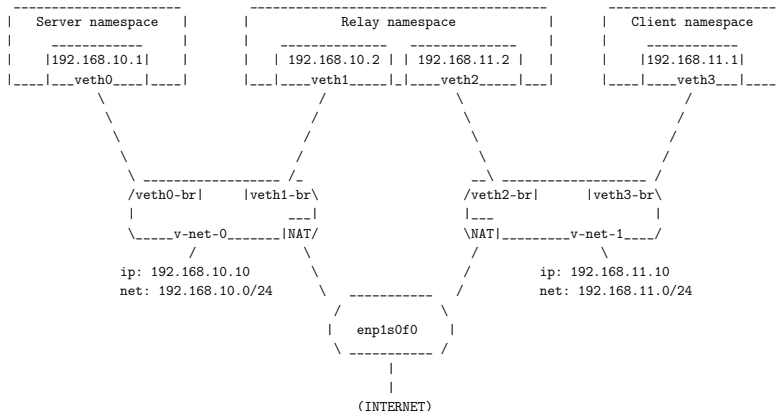
- Retransmission happen at stream level
- Relay might not have correct stream state
- Client needs **all** parts of a frame for correct media display

Packet Retransmission cont.



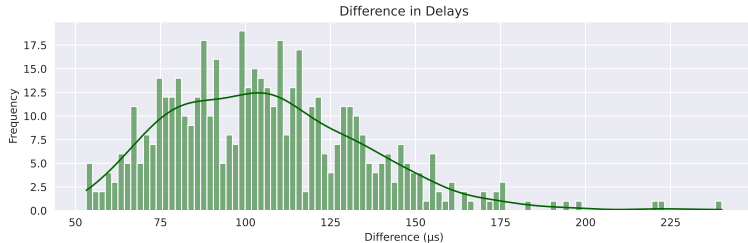
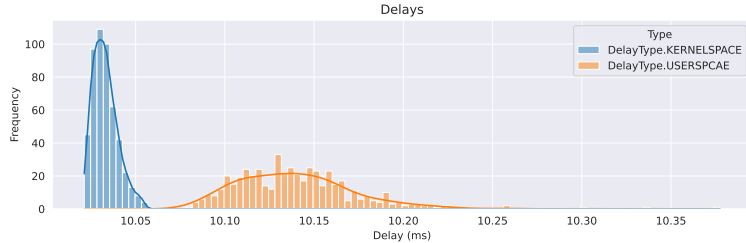
- 1 Introduction
- 2 QUIC and eBPF
- 3 Fast-Relays
- 4 Testing and Results**
- 5 Conclusion and Future Work

Test Setup



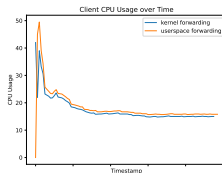
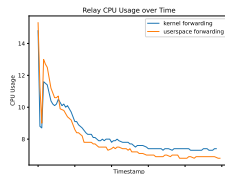
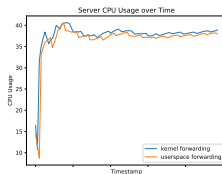
Test Results Delay Reduction

Delay analysis of messages with and without kernel-space forwarding



Test Results CPU Usage

- No Impact on CPU Usage
- Fewer System Calls
 - Mainly due to reduced Userspace Synchronization



System Calls

- Example Stream of 30 Seconds
- Overall System Calls
 - Userspace forwarding: 296132 calls
 - eBPF forwarding: 225674 calls
 - Reduction of 24%
- *futex*
 - Reduction of 34%
 - 21666 calls instead of 32940
- *nanosleep*
 - Reduction of 42%
 - 14293 calls instead of 24716
- *epoll_wait*
 - Reduction of 67%
 - 11289 calls instead of 34149

- 1 Introduction
- 2 QUIC and eBPF
- 3 Fast-Relays
- 4 Testing and Results
- 5 Conclusion and Future Work**

Conclusion

- Delay reduction via eBPF forwarding
- More application specific relay code needed
- No impact on CPU usage

Future Work

- Hardware offloading of en- and decryption
- Expand to other protocols
- Prototype completion
 - Congestion control
 - Physical setup for testing

That's it!

Any Questions?