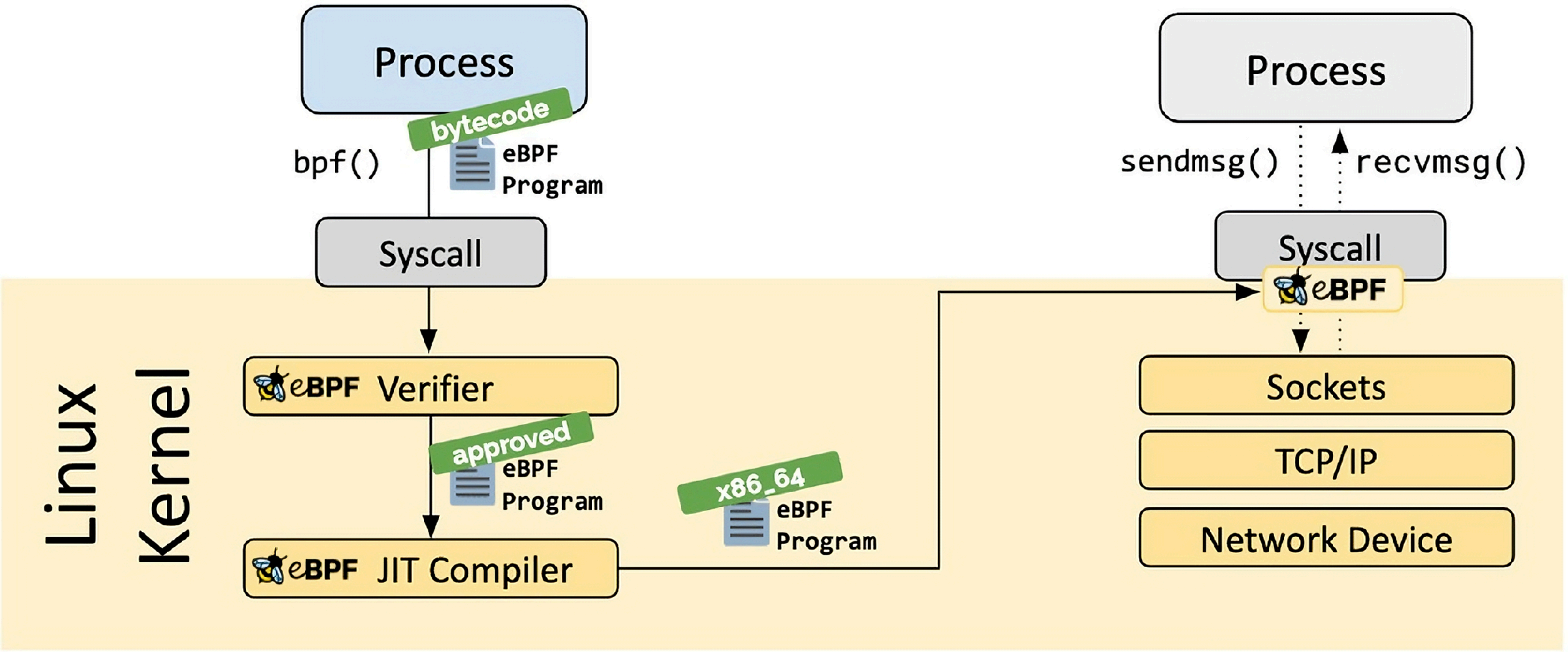
SwiftCall: Selective Forwarding using eBPF

Aditya Thaker
Prof. Erik Keller

Introduction

- eBPF enables safe, efficient kernel-space execution of custom programs.
- Widely adopted for security, observability, and networking in production.
- Programs are written in C, compiled to bytecode via LLVM, and JIT-compiled in the kernel.
- Kernel verifier ensures safety before execution.

eBPF

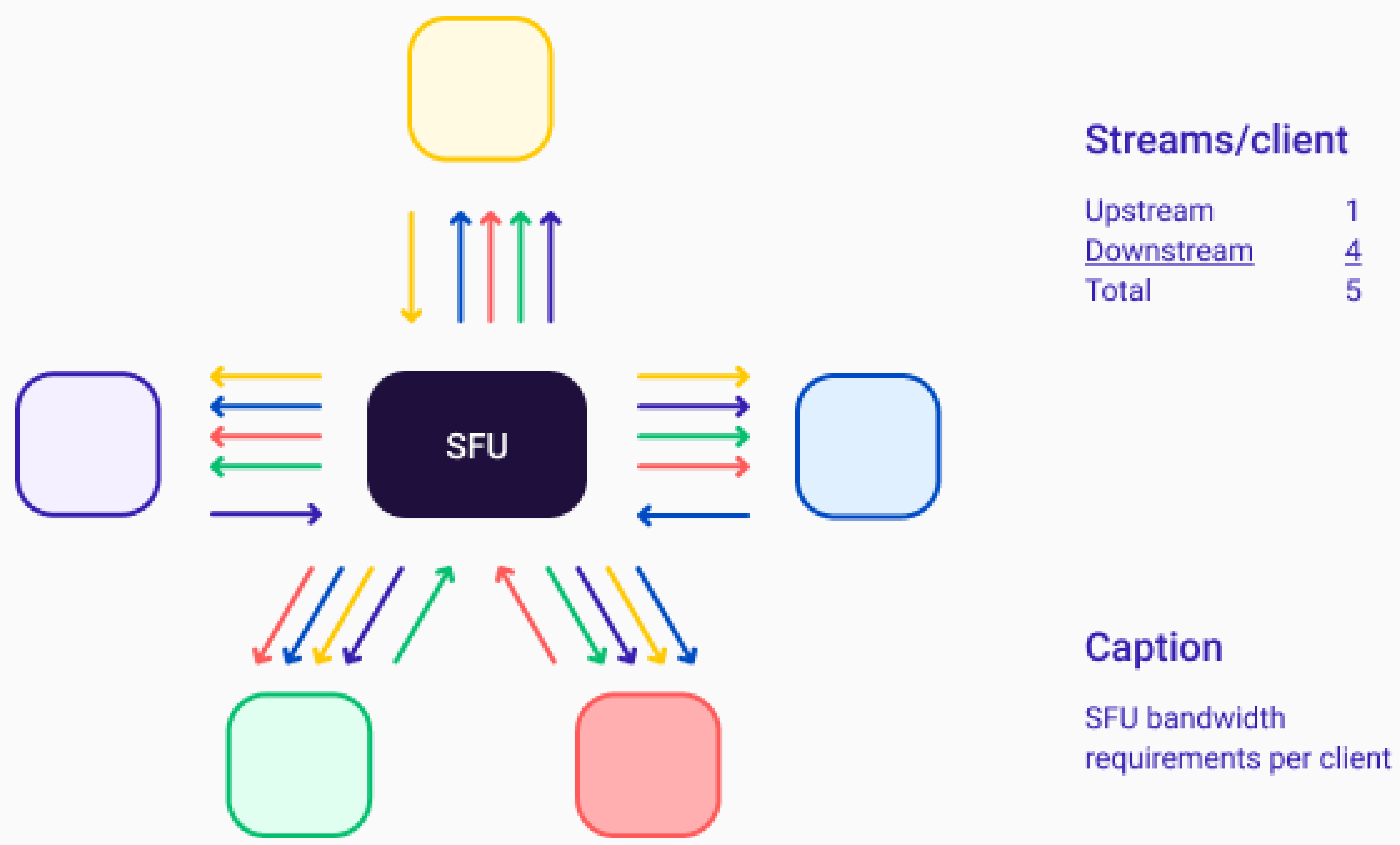


Process
bytecode
bpf()
eBPF
Program
Syscall
Linux
Kernel
eBPF Verifier
approved
eBPF
Program
eBPF JIT Compiler
x86_64
eBPF
Program
Process
sendmsg()
recvmsg()
Syscall
eBPF
Sockets
TCP/IP
Network Device

Selective Forwarding Unit

- SFU is a network architecture for video conferencing where the server receives all streams, processes them, and then forwards them to the participants.
- Data Packets have to go through entire linux stack before being forwarded.
- A list of participants is required to forward packets, scope for improving computation.
- Allow sending streams in multiple quality, formats, for flexibility.

Selective Forwarding Unit



Caption

SFU bandwidth requirements per client

Need for Optimization

- SFU handles multiple users on various ports and forwards RTP streams.
- eBPF can offload work from the CPU, by handling packets pre-OS.
- Works exclusively on RTP packets, malformed packets dropped before they are sent up the stack.
- Separation of Concerns - Control and Data plane.

eBPF

Fall 2024

University of Colorado Boulder

```
 0                   1                   2                   3
 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1 2 3 4 5 6 7 8 9 0 1
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|V=2|P|X|  CC   |M|     PT      |       Sequence Number         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                           Timestamp                           |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|           Synchronization Source (SSRC) identifier            |
+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+=+
|            Contributing Source (CSRC) identifiers             |
|                             ....                              |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|                            Payload                            |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

RTP Packet

Fall 2024

Focused on eBPF development, WebRTC protocols and how to off load the data plan

eBPF Programs

Experimented with eBPF network hooks like XDP, TC and compared feasibility of each

WebRTC Protocol Deep-dive

Researched the ins and outs of WebRTC protocols and how connections are made, discovered and updated to understand packet capture at the NIC level.

Issues

While, this was a great learning experience this project lacked heavily in these following key areas

Compilation and Attaching

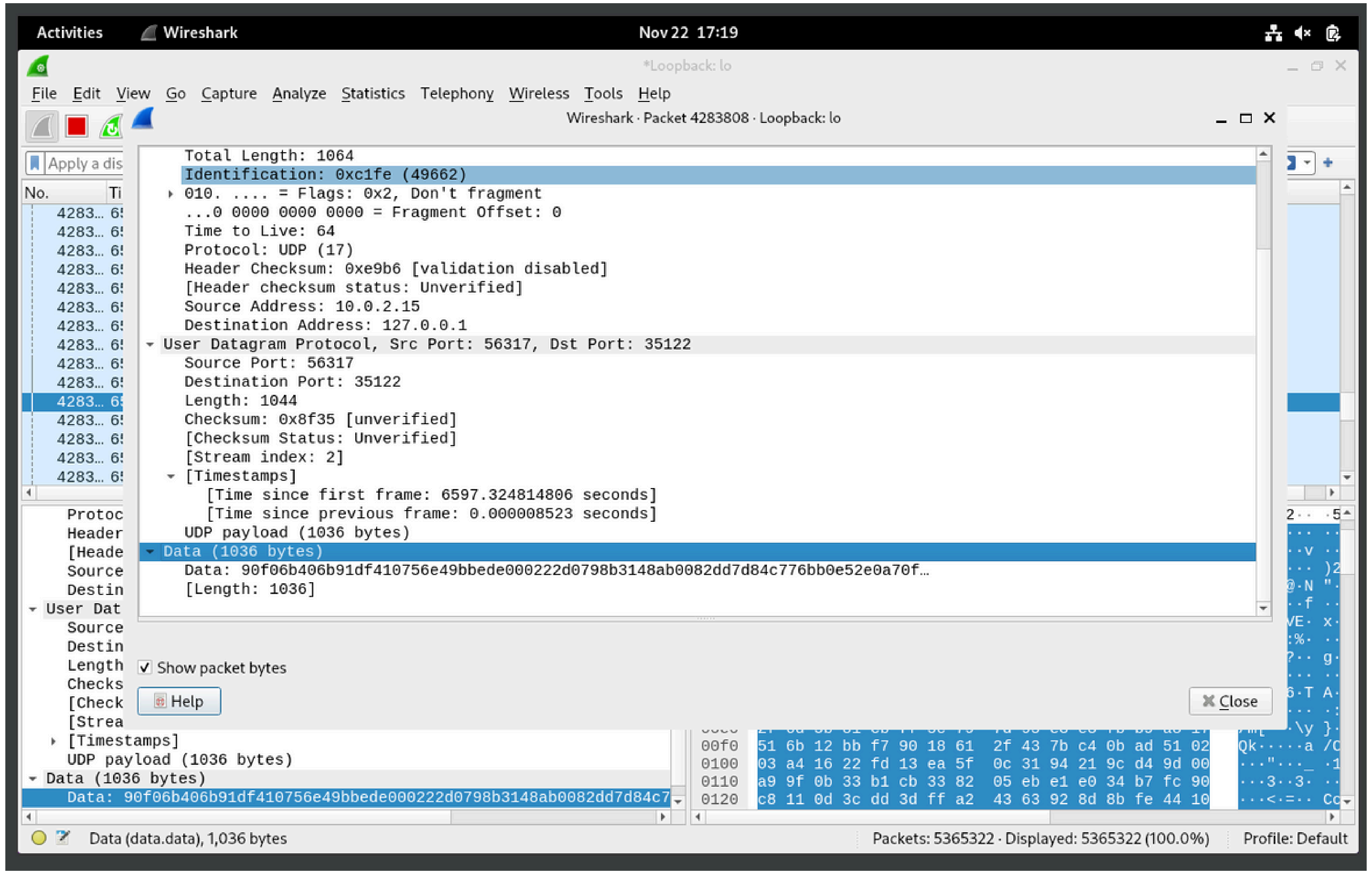
Manual Work to compile using clang and then attaching the entire thing to a specific interface and qdisc

Constant Limitations

Manually adding users to the eBPF map and only allowing 3 users to connect.

Lack of Signalling

The current work only focused on forwarding the packets, but no server to actually get clients.



UDP Packets being Captured

```
v=0
o=- 3546004397921447048 1596742744 IN IP4 0.0.0.0
s=-
t=0 0
a=fingerprint:sha-256 0F:74:31:25:CB:A2:13:EC:28:6F:6D:2C:61:FF:5D:C2:BC:B9:DB:3D:
a=group:BUNDLE 0 1
m=audio 9 UDP/TLS/RTP/SAVPF 111
c=IN IP4 0.0.0.0
a=setup:active
a=mid:0
a=ice-ufrag:CsxzEWmoKpJyscFj
a=ice-pwd:mktpbhgREmjEwUFSIJyPINPUhgDqJlSd
a=rtcp-mux
a=rtcp-rsize
a=rtpmap:111 opus/48000/2
a=fmtp:111 minptime=10;useinbandfec=1
a=ssrc:350842737 cname:yvKPspsHcYcwGFTw
a=ssrc:350842737 msid:yvKPspsHcYcwGFTw DfQnKjQQuwceLFdV
a=ssrc:350842737 mslabel:yvKPspsHcYcwGFTw
a=ssrc:350842737 label:DfQnKjQQuwceLFdV
a=msid:yvKPspsHcYcwGFTw DfQnKjQQuwceLFdV
a=sendrecv
a=candidate:foundation 1 udp 2130706431 192.168.1.1 53165 typ host generation 0
a=candidate:foundation 2 udp 2130706431 192.168.1.1 53165 typ host generation 0
```

SDP Protocol

Issues

Here were some issues I found with eBPF itself, that hindered progress back then

Stack Memory Limits

XDP only has 512 bytes of stack memory, that doesn't allow you to clone packets.

Limits on eBPF hooks

XDP only has 512 bytes of stack memory, that does not allow you to clone packets. TC allows you to clone and send packets but is slower than XDP.

Spring 2025

University of Colorado Boulder

SwiftCall

- Worked on creating an SFU server in Go using Pion, to handle control plane.
- Extended eBPF program to accomodate variable number of users.
- Used bpf2go to load eBPF programs using Go eBPF code.
- Tested server using tshark packet capture.
- Developed scripts to build□attach eBPF, build server-client and testing infrastructure.
- Parsed and plotted results, and a Makefile to run these things.

Design

SFU Server

- Opens HTTP Port on 8080, and collects client offers.
- Client offers are used to set Local and Remote Descriptions, i.e. SDP.
- On new connections, these SDPs are sent out again, like a change in contract, i.e. renegotiations.
- Each track in Go is handled by a callback, and then tracks are sent on different coroutines.

eBPF Program

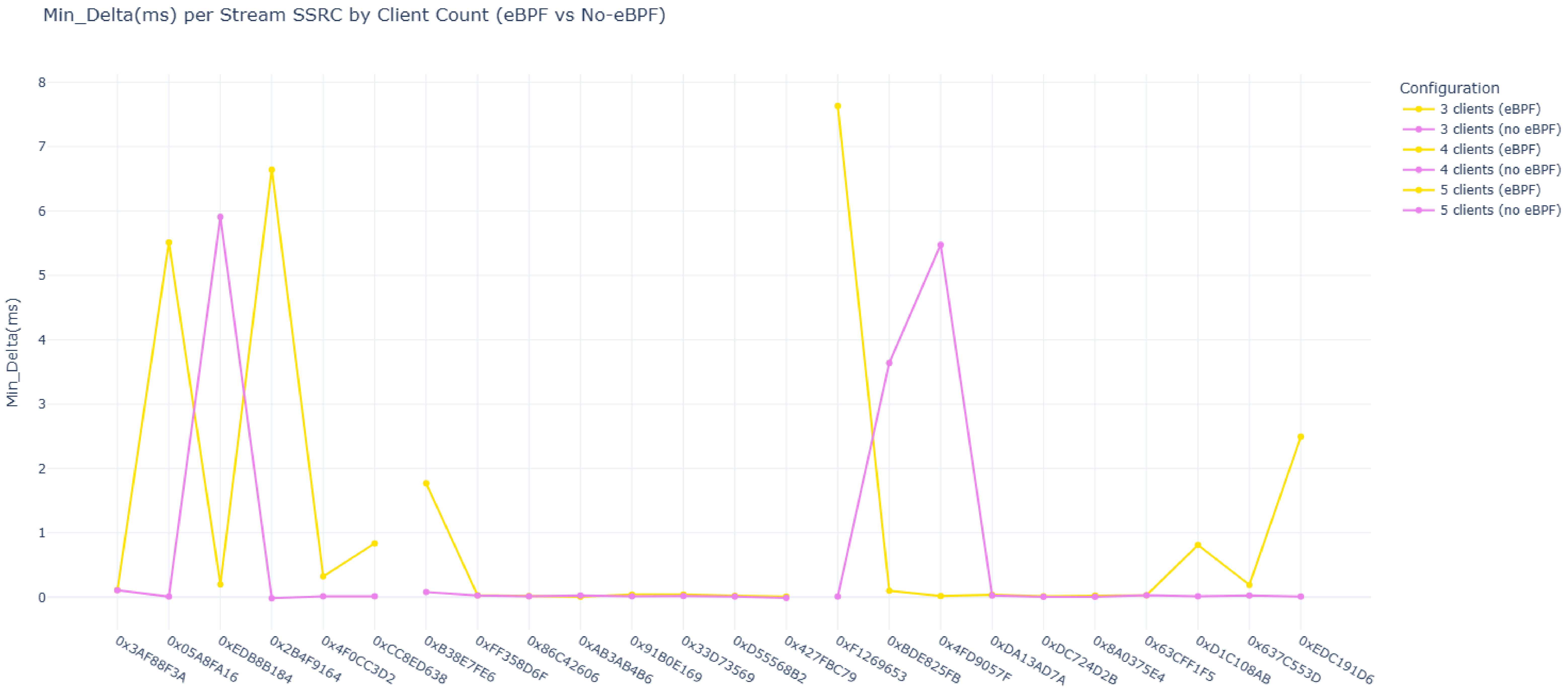
- Stores user info like IP and Port of each client, after signalling into an eBPF map.
- When attached to specific interface it forwards packets to all clients in the map, when an RTP packet is received.
- This is extensible to store user preferences like streams, and bandwidth.
- Deletes map on teardown.

Scripts

- build - This builds the eBPF program
- results.sh - This runs one trial of packet capture for given no. of clients, capture time.
- plots.py - This creates plots from the txt files in results directory.
- benchmarks.py - In sfu directory, this is used to create dummy clients for given time.
- clean - Deletes all logs, results, and captures.
- Makefile - A compilation of all these commands.

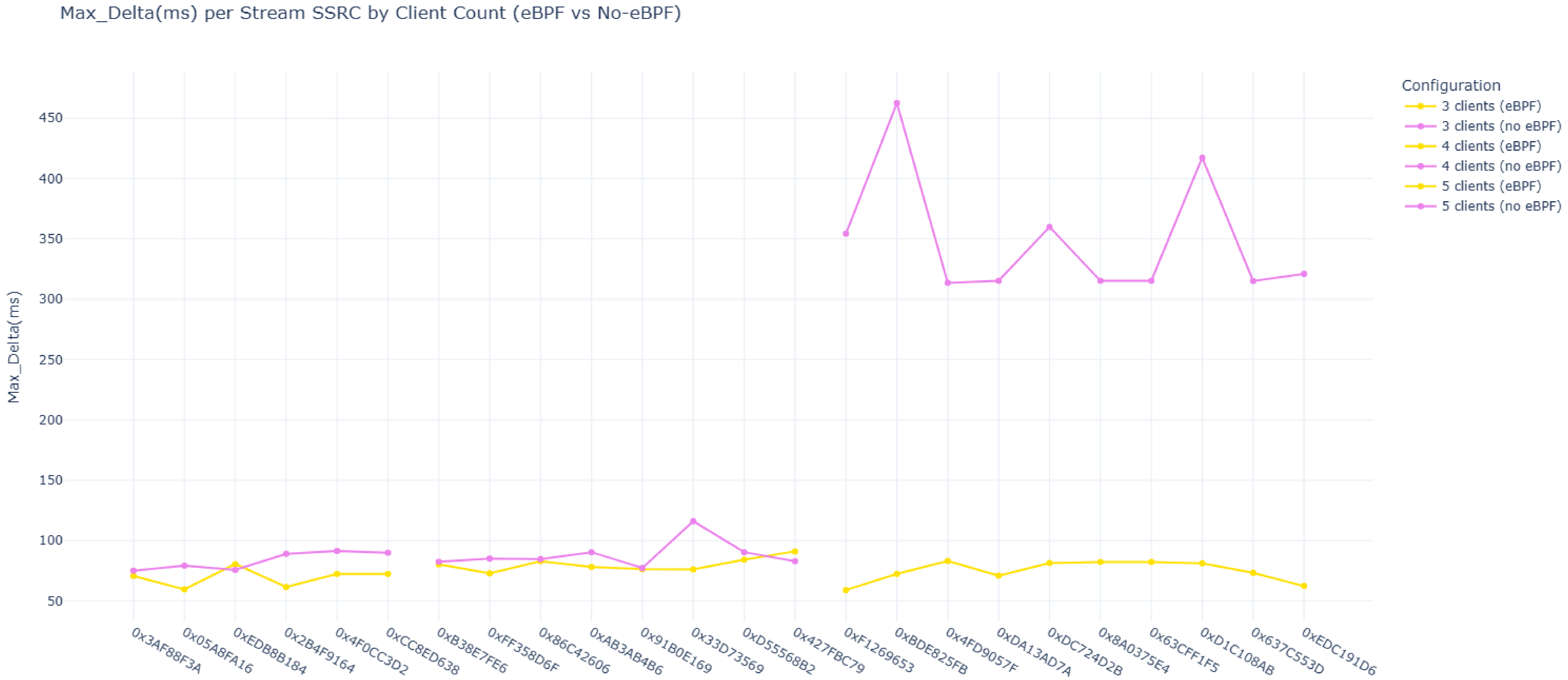
Evaluation

Results - Min Delta



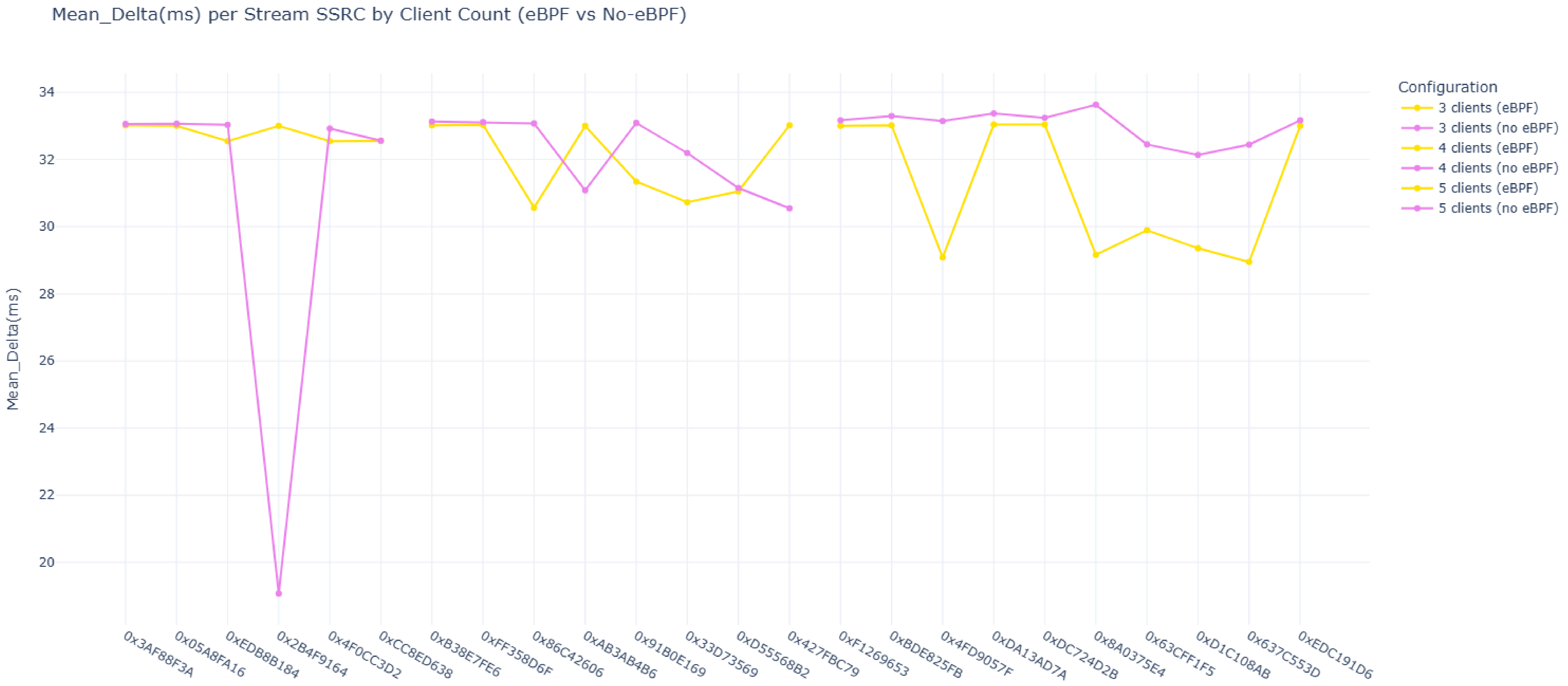
For each calculation tshark recorded for 60s after 30s of startup, clients lived for 100s

Results - Max Delta



For each calculation tshark recorded for 60s after 30s of startup, clients lived for 100s

Results - Avg Delta

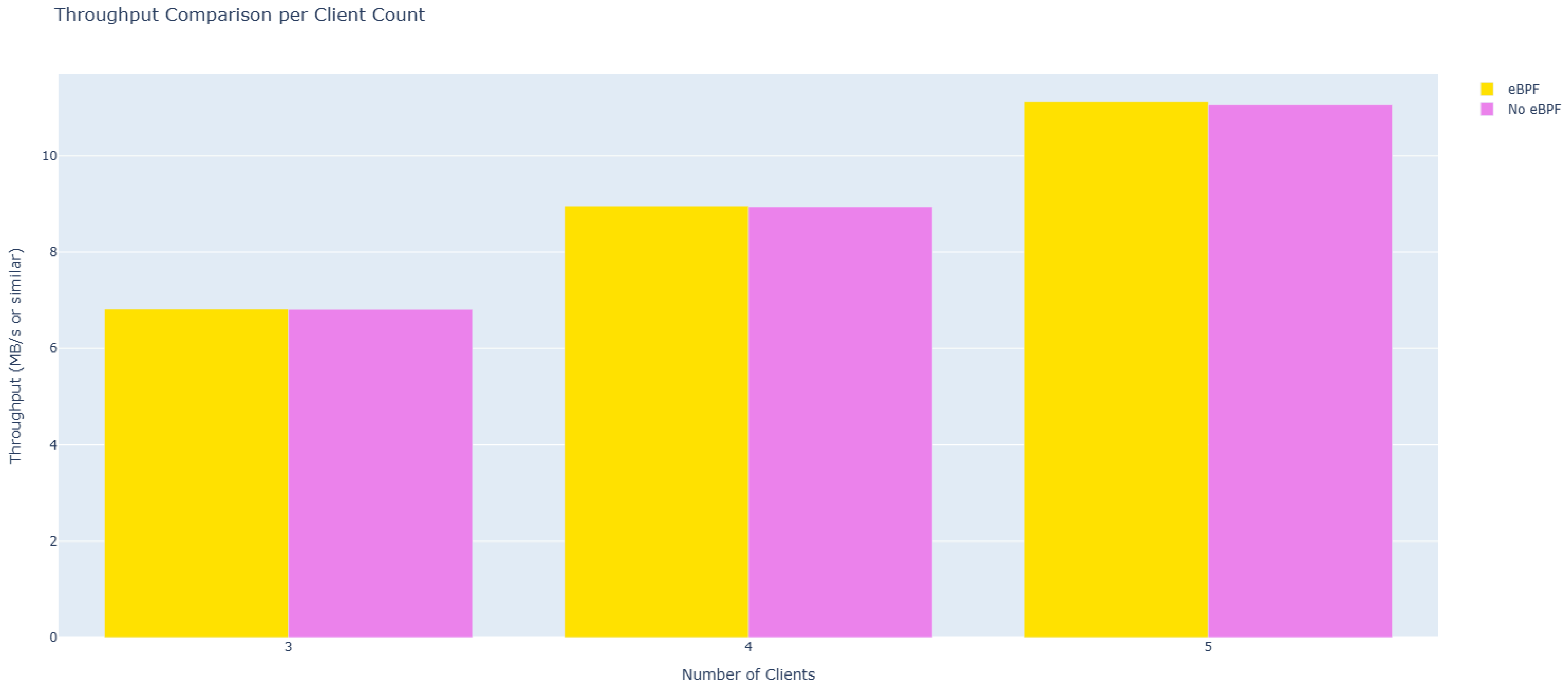


For each calculation tshark recorded for 60s after 30s of startup, clients lived for 100s

eBPF

University of Colorado **Boulder**

Throughput



For each calculation tshark recorded for 60s after 30s of startup, clients lived for 100s

eBPF

University of Colorado **Boulder**

Aggregate Results

2.70% Max Delta Reduction

Max Delta Reduction increases with increase in number of clients

0.0006% Throughput Decrease

There was a throughput decrease, which is not significant but does increase with number of clients

0.018% Avg Delta Reduction

Avg Delta Reduction increases with increase in number of clients

0.0021% Min Delta Increase

Min Delta Increase decreases with the increase in the number of clients.

For each calculation tshark recorded for 60s after 30s of startup, clients lived for 100s

Insights

- Delta - This is the difference in milliseconds between the arrival of 2 packets in a stream, i.e. Latency
- End Ranges - eBPF mode has higher min delta, and a way lower max delta
- Mean - This gives eBPF mode lower mean delta as the number of clients increase
- Throughput - The throughput is constant given my testing, however this demands a deeper investigation into throughput with more clients and robust infrastructure.

Bonus Work

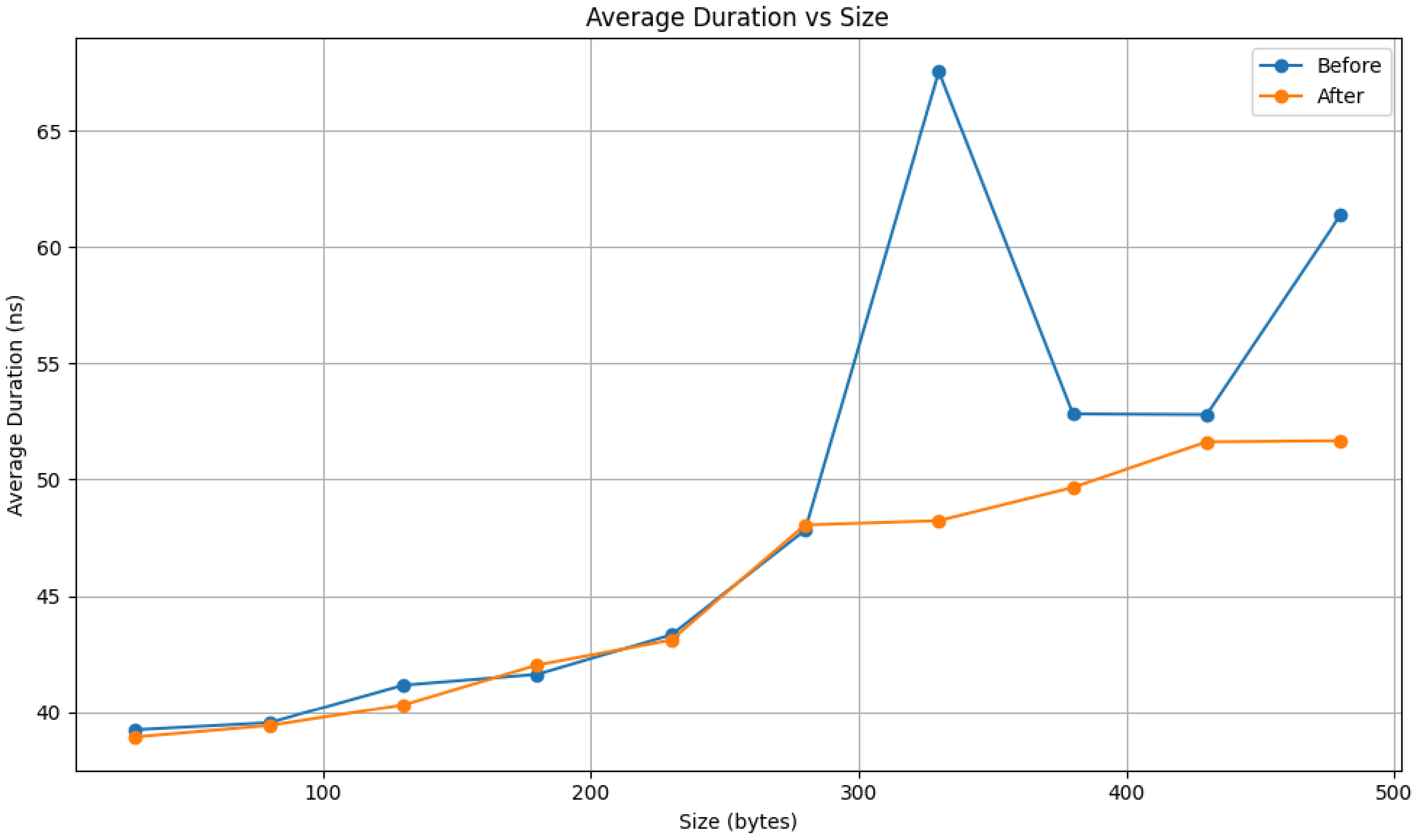
Bonus: Results

7.02% Time Saved

During initialization of large structs around 512 bytes. with a high of around 50%

0.08% Instruction Reduction

Calculated from running our optimizations on tracee and tetragon, then counting the difference in instruction count



*For each size, 10,000 operations were performed, and the mean was computed.

eBPF

Future Work

University of Colorado **Boulder**

Features

- Ability to subscribe to different streams - Audio, Video, Participant.
- Having a max of 20 clients - Easiest one
- Testing on isolated machines, more resources to test Go threading.
- Testing CPU Load when server is running in both modes - eBPF vs. Vanilla
- Understanding how Pion uses the system resources and update the naive eBPF implementation.

Conclusion

Thoughts

- Build an SFU server that can handle multiple clients in eBPF, non-eBPF modes.
- Sends constant fake video, but easily extensible to read from file or device.
- Testing and plotting information about packet capture at the network level, to iterate over the system.
- Extensible system on SFU server, eBPF and testing fronts.

References

- Tamás Lévai, et al. 2023. Supercharge WebRTC: Accelerate TURN Services with eBPF/XDP. In Proceedings of the 1st Workshop on eBPF and Kernel Extensions (eBPF '23). Association for Computing Machinery, New York, NY, USA, 70–76. https://doi.org/10.1145/3609021.3609296
- https://github.com/xdp-project/xdp-tutorial
- https://fedepaol.github.io/blog/2023/09/11/xdp-ate-my-packets-and-how-i-debugged-it/
- https://webrtcforthecurious.com
- https://voximplant.com/blog/an-introduction-to-selective-forwarding-units
- https://ebpf.io/what-is-ebpf/
- https://pkg.go.dev/github.com/cilium/ebpf/cmd/bpf2go
- https://github.com/pion/webrtc

Thank You

eBPF

University of Colorado Boulder