SURF Report
Presentation

QUIC++: Protocol for video streaming over QUIC

GUIDE: Dr. Mukulika Maity

Team Members:

- **VIDUR GOEL (BTech CSAM) 2021364**
- **MANAS NARANG (BTech CSAM) 2021473**

Abstract

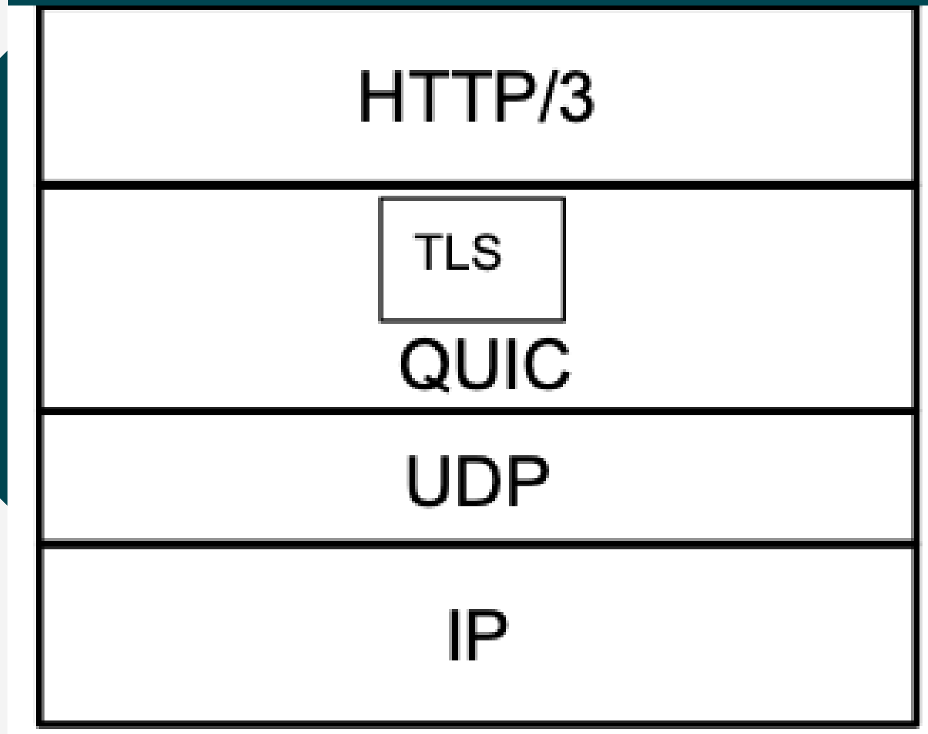
- QUIC++, a media-optimized QUIC extension, enhances streaming.
- Summer project refines protocol switching in Chromium for better streaming.
- Browsers cope with middlebox disruptions via QUIC to TCP fallback mechanism.
- Research tackles ambiguity between congestion and middlebox issues, elevating QoE.

QUIC

- "Quick UDP Internet Connections"
- Built on top of UDP.
- Reduce latency
- Enhance connection security.
- Swift 0-RTT hand-shake
- Resolution of the HOL problem and efficient stream multiplexing.
- Implemented in application space
- HTTP/3 uses QUIC.

HTTP/3

- Modern internet protocol facilitates data transfer between a user's web browser and a web server.
- Designed to enhance web page loading times and overall performance.
- Utilize the QUIC transport protocol.
- This protocol upgrade aims to provide users with faster and more reliable web browsing experiences.



What is Connection Racing ?

- Web browser technique to speed up network connections.
- Simultaneously initiates connections using different protocols, like HTTP/3 (QUIC) and HTTP/2.
- Uses the one that establishes a connection faster.
- If there are issues, it gracefully falls back to an alternative.
- Faster web page loading in variable network conditions.

Fallback Mechanism

- Mechanism implemented by browsers
- Switching from the QUIC protocol to the traditional TCP protocol
- Encountering issues like network congestion or middlebox interference, ensuring continued connectivity and data transfer.
- Stable connection in challenging network conditions.

Impacts of Fallback on Congestion State:

- Preserving the congestion state is a challenge
- Isolated implementations in user space and kernel space.
- Congestion state Info is not transferred from QUIC to TCP.
- TCP faces two potential scenarios after this transition:

1. New Connection:
    a. 3-WAY handshake before data transmission.
    b. Time to restore previous congestion state.
2. Existing Connection:
    a. Time to restore previous congestion state.

Does the fallback mechanism of QUIC significantly affect the performance of QUIC-enabled streams?

Does the fallback mechanism of QUIC significantly affect the performance of QUIC-enabled streams?

YES!

It has been mentioned in the following paper for reference:
YouTube over Google’s QUIC vs Internet Middleboxes: A Tug of War between Protocol Sustainability and Application QoE https://arxiv.org/pdf/2203.11977.pdf

Our Approach:

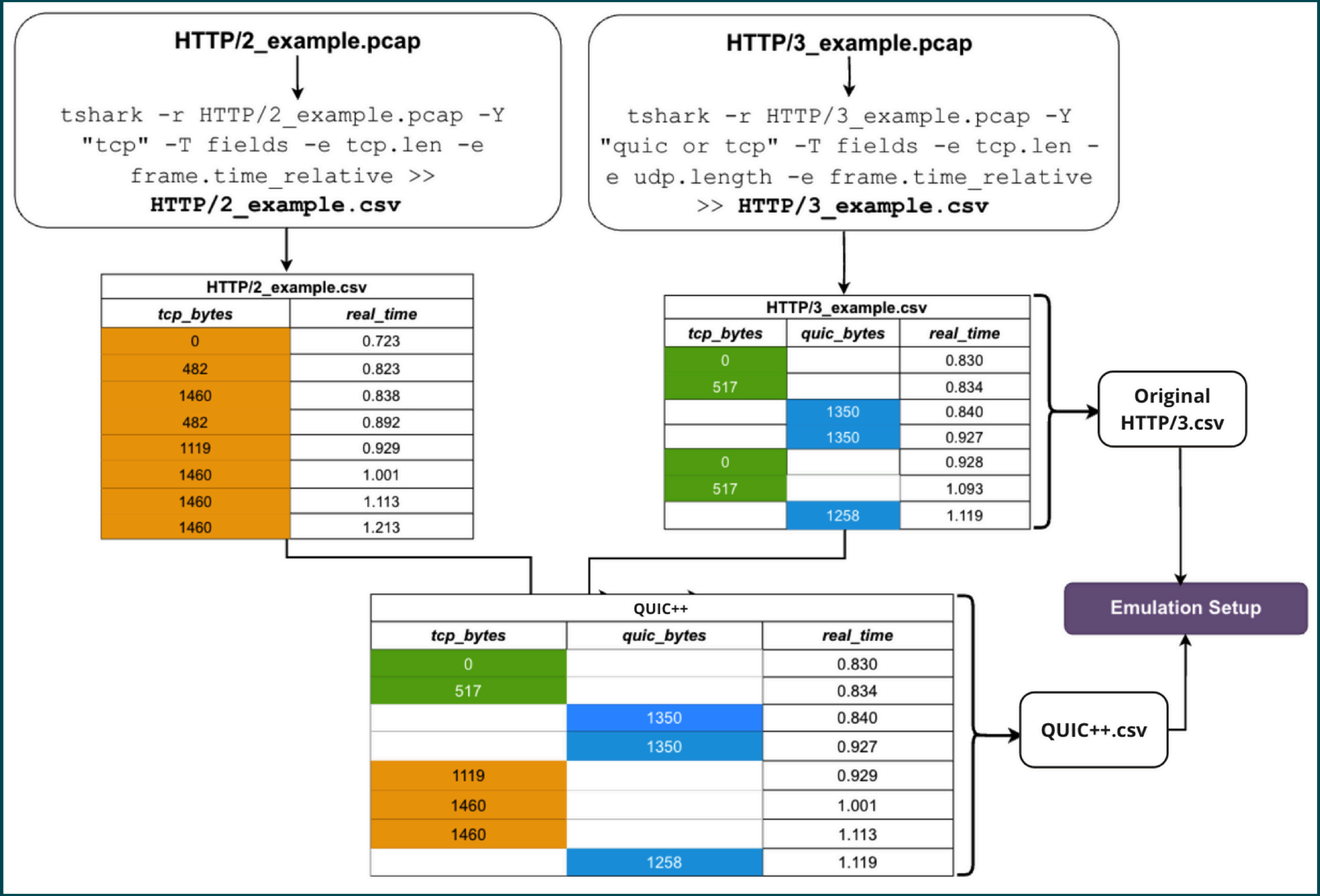
QUIC++ Design

- **Idea:** Transfer of congestion states from QUIC to TCP
- To ensure uninterrupted transitions between these protocols.
- Practical implementation is unattainable due to the unavailability of access to the YouTube server's underlying infrastructure.
- Evaluation of our idea by simulating its design.

Stitched QUIC

- QUIC and TCP both employ the CUBIC congestion control algorithm, forming the basis for our approach.
- Conducted two separate video streaming sessions: one utilizing QUIC and the other relying solely on TCP (pure TCP trace) to investigate our approach.
- Evaluated these 2 traces to create a new trace- stitched QUIC.
- We check the QUIC trace for fallback to TCP and replace the congestion state with that of the pure TCP for this duration.
- When protocol switches back to QUIC in the QUIC trace, we copy the same values as that of the original trace.

Stitched File Creation



```
tshark -r HTTP/2_example.pcap -Y
"tcp" -T fields -e tcp.len -e
frame.time_relative >>
HTTP/2_example.csv
```

```
tshark -r HTTP/3_example.pcap -Y
"quic or tcp" -T fields -e tcp.len -
e udp.length -e frame.time_relative
>> HTTP/3_example.csv
```

HTTP/2_example.csv

tcp_bytes	*real_time*
0	0.723
482	0.823
1460	0.838
482	0.892
1119	0.929
1460	1.001
1460	1.113
1460	1.213

HTTP/3_example.csv

tcp_bytes	*quic_bytes*	*real_time*
0		0.830
517		0.834
	1350	0.840
	1350	0.927
0		0.928
517		1.093
	1258	1.119

QUIC++

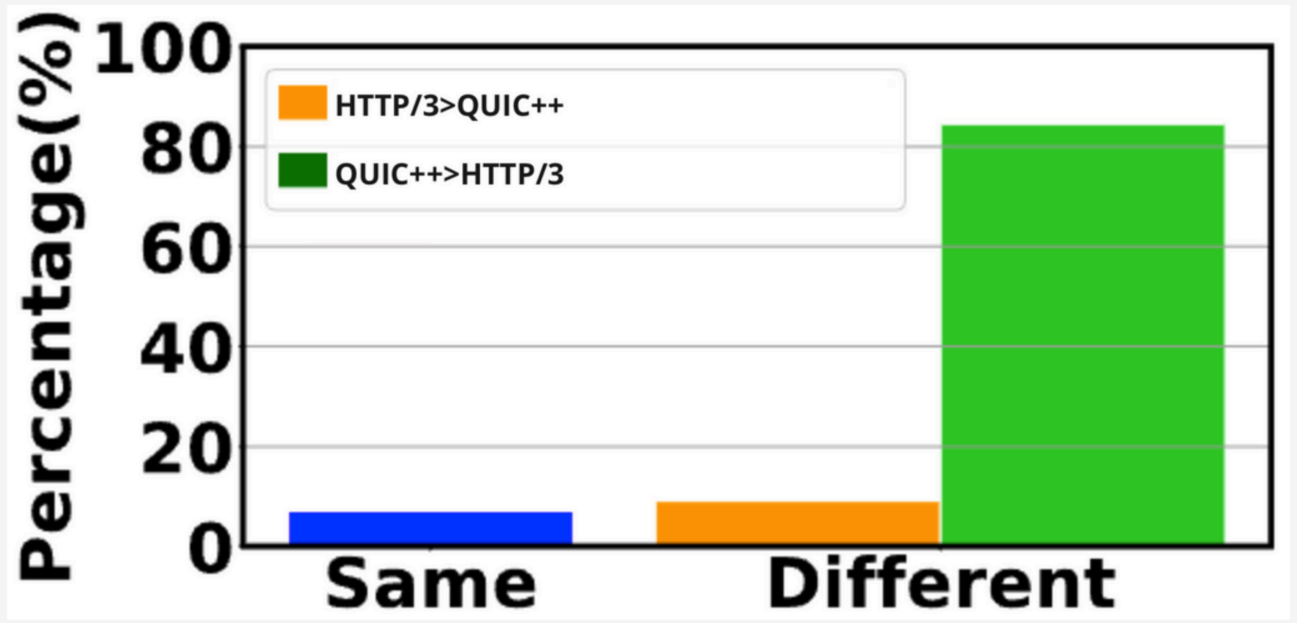
tcp_bytes	*quic_bytes*	*real_time*
0		0.830
517		0.834
	1350	0.840
	1350	0.927
1119		0.929
1460		1.001
1460		1.113
	1258	1.119

Data Collection

- We first collected network traces(pcaps) for both QUIC and TCP streaming sessions using Wireshark software.
- Ran bash scripts to convert these pcaps to csv format using tshark.
- After this, we ran a python file to generate our new trace file for stitched QUIC.
- Converted this stitched QUIC and the original QUIC traces to mahimahi trace files.
- We ran these trace files on mahimahi emulator to generate pcaps and logs.
- These logs were used to evaluate and compare the performance of stitched QUIC vs the original QUIC.

Evaluation

- The application logs collected during our data processing were used to compute Quality of Experience(QoE).
- The instantaneous QoE at time t was determined as follows: QoE = Avg. Bitrate - Avg. Bitrate Variation - 4.3 × Avg. Stall.
- Used this QoE to compare the performance of our simulated QUIC++(stitched QUIC) against the performance of HTTP/3(original QUIC).
- It was observed that QUIC++ outperformed HTTP/3 in 84% of the cases.
- This substantiates that a transfer of congestion state from QUIC to TCP stands to significantly improve the performance of QUIC.



Challenges

- Processing our extensive dataset was time-consuming.
- Task distribution across multiple systems requires setup with Chromium and Mahimahi software.
- Time-consuming
- Turned to Docker containers to encapsulate our entire experimental setup.
- Streamlined the process, eliminating software dependency issues and significantly improving efficiency.

Ebpf

- "Extended Berkeley Packet Filter,"
- Allows for dynamic and efficient analysis of network packets and system events within the Linux kernel
- Here are some key points about eBPF:
    - Programmable Kernel-Level Tool
    - Performance and Safety
    - Network Packet Analysis
    - System Tracing and Profiling

Conclusion and Future Work

- Due to the lack of direct access to the YouTube server, we couldn't perform an actual transfer of congestion state from user space to kernel space. Instead, we simulated this transfer using a trace file.
- We suggest that the Extended Berkeley Packet Filters (eBPF) framework could potentially achieve this transfer in real-time.
- Previous research has explored eBPF's dynamic kernel space programmability.
- eBPF could enable browsers using QUIC to communicate congestion state data to the TCP layer in the kernel, but its comprehensive implementation and performance testing require further analysis, including security considerations.