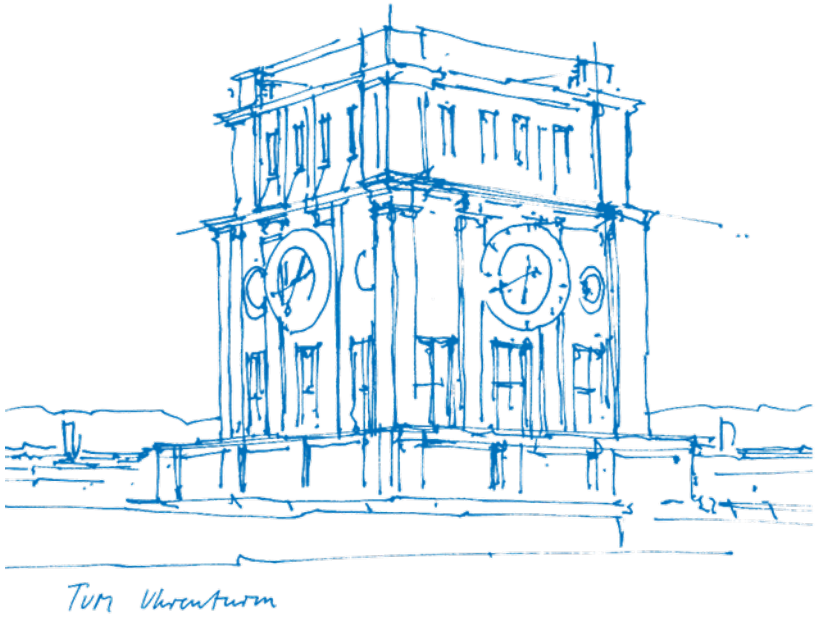
Connected Mobility
School of Computation
Technische Universität München

Jasper Rühl
Technische Universität München
School of Computation
Connected Mobility
9. Oktober 2024

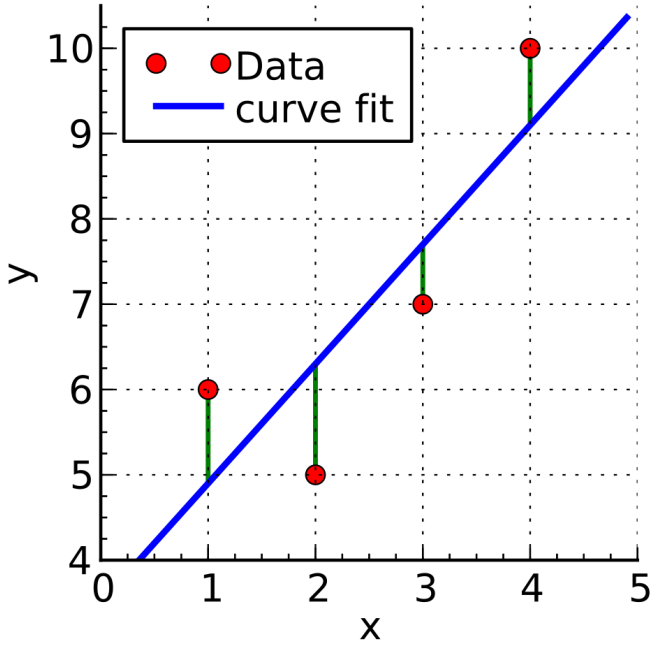


Introduction

- Background
  - Linear Regression
  - QUIC
  - Congestion Control
  - quic-go
- What we want
- First Solutions
- Implementation
- Evaluation
- Future Work

Linear Regression

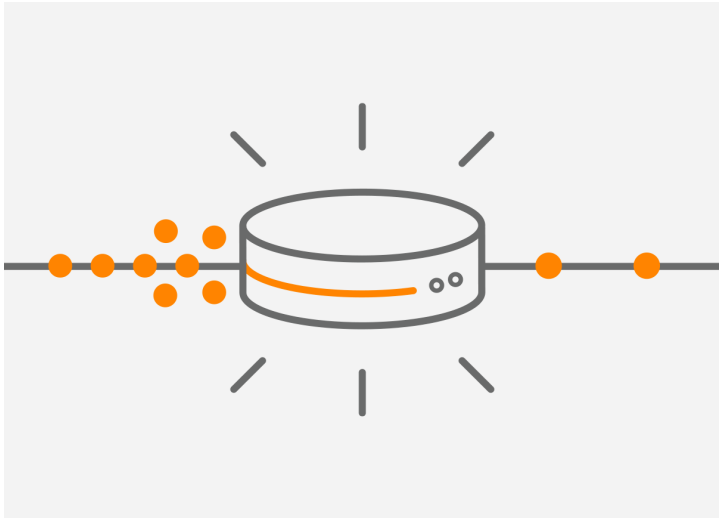
Given some observed samples, fit a linear function best describing the samples



Quick question

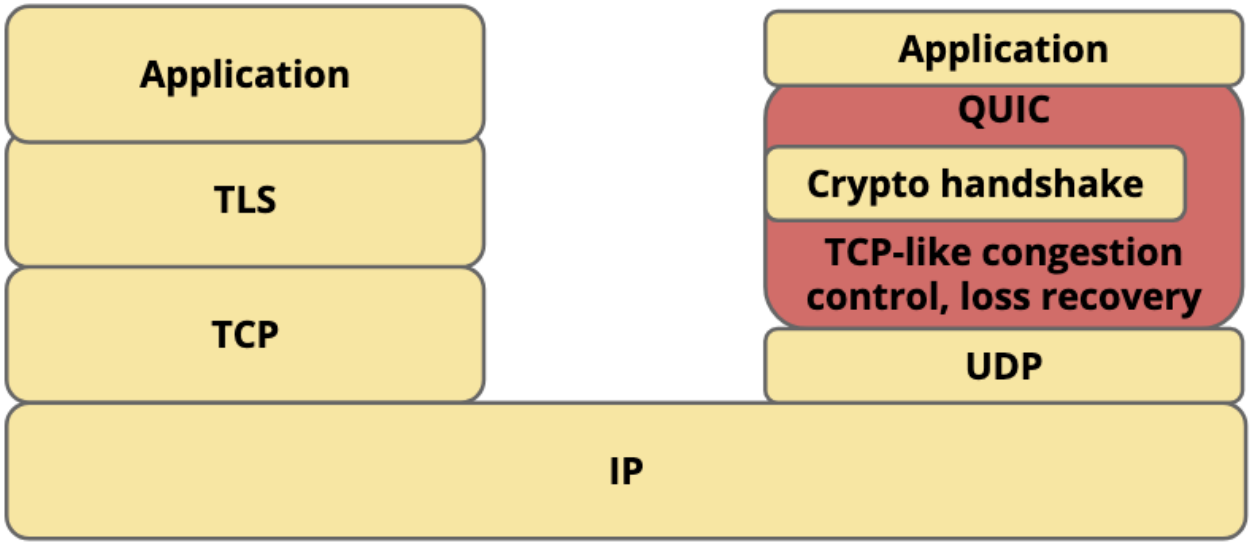
Congestion Control

- Takes care of the volume of data to be send
- Estimates available bandwidth
- Goal: utilize the available bandwidth while avoiding sending too much data
- Measurable critiria: Loss, Delay, ...



QUIC

Transport protocol designed by Google



Connection Prioritization

Some data is more urgent than other:

+ Livestreaming
+ Video Conferencing
+ Control commands
- File Transfer

This prioritization can already be done by switches. (Quality of Service)
But it can not be done by an application itself!

Connection Prioritization

Some data is more urgent than other:

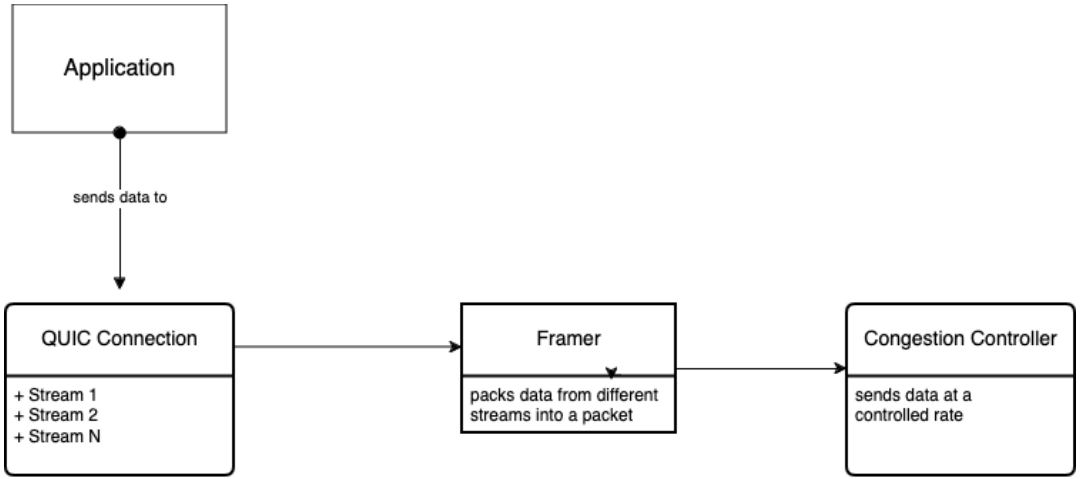
+ Livestreaming
+ Video Conferencing
+ Control commands
- File Transfer

This prioritization can already be done by switches. (Quality of Service)
But it can not be done by an application itself!

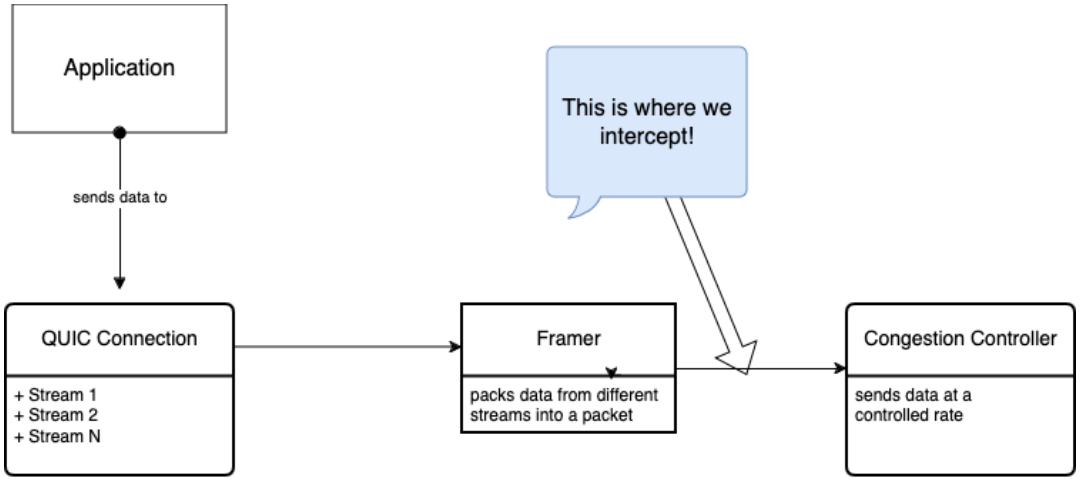
This is what we change!

We add a stream prioritization mechanism to quic-go, allowing an application to prioritize specific streams.

How does quic-go work?



How does quic-go work?



It should look something like this:

```
if stream not in prio:
    if not canSend(stream):
        continue
```

First Solution

```
def canSend(streamID, length):
    if bytesSentWithinLast(1second) < THRESHOLD:
        recordSentBytes(time.now(), length)
        return True
    else:
        return False
```

Drawbacks?

First Solution

```
def canSend(streamID, length):
    if bytesSentWithinLast(1second) < THRESHOLD:      # hardcoded? we can do better...
        recordSentBytes(time.now(), length)
        return True
    else:
        return False
```

Our solution should be

- suitable for variable network environments
- react to changes in network capacity

→ No hardcoded limit!

First Solution

```
def canSend(streamID, length):
    if bytesSentWithinLast(1second) < THRESHOLD:      # hardcoded? we can do better...
        recordSentBytes(time.now(), length)
        return True
    else:
        return False
```

Our solution should be

- suitable for variable network environments
- react to changes in network capacity

→ No hardcoded limit!

We do not care about the low priority data!

We care about high priority data!

RateMonitor

Idea:

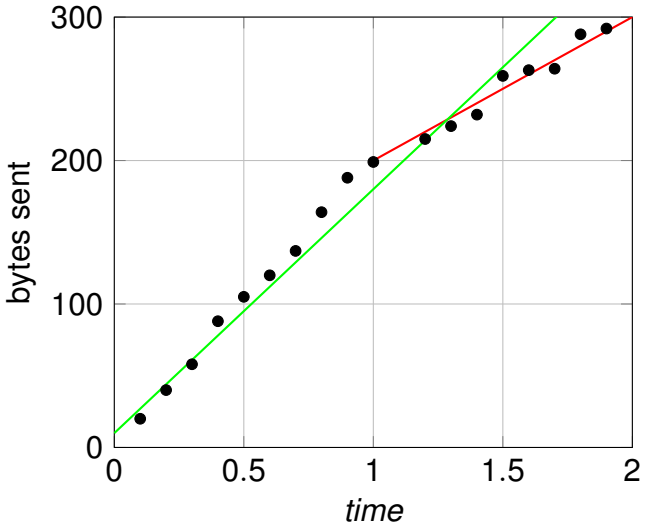
- estimate the sending rate of the high priority data
- monitor the development of the sending rate
- if everything is stable, everything is fine

How to detect a decrease?

- Look at different timeframes
- Simple division:
  - $ratescore = \frac{slope(1second)}{slope(2second)}$

Intuition:

- ratescore > 1: increasing, all good
- ratescore < 1: decreasing, bad



Three Criteria

To assess the connections quality, three factors are used

- Ratescore
  - indicates the slope of the sending rate
- Round Trip Time (RTT)
  - similar to the RateMonitor, Linear Regression is used
- Current throughput compared to maximum throughput
  - query throughput every second
  - divide this by the maximum observed value

But how do we do with all these criteria?

Rate Reduction

```
def canSend(streamID, length):
    if bytesSentWithinLast(1second) < threshold:
        recordSentBytes(time.now(), length)
        return True
    else:
        return False
```

Rate Reduction

```
def canSend(streamID, length):
    if bytesSentWithinLast(1second) < threshold:
        recordSentBytes(time.now(), length)
        return True
    else:
        return False

# in separate goroutine
while True:
    metrics = getMetrics()
    growFactor = metrics.deriveFactor()
    threshold *= growFactor
    sleep(1second)
```

Rate Reduction

Recall the ratescore: $ratescore = \frac{slope(1second)}{slope(2second)}$

- ratescore > 1: increasing, all good
- ratescore < 1: decreasing, bad

How should this affect the amount of allowed bytes?

- Small decline $\rightarrow$ small shrink
- Larger decline $\rightarrow$ large shrink!

$\rightarrow$ formula

$$growFactor(ratescore) = \begin{cases} ratescore^2 & \text{if } growFactor < 1 \\ 1 & \text{else} \end{cases}$$

Plugging the scores together

all together:

$$growFactor = 1.05 * growFactor(ratescore) * growFactor(RTT) * growFactor(currentThroughputToMaximum)$$

Exponential growth is fast...

To avoid overshooting quickly: $threshold = \begin{cases} threshold * \frac{9+growFactor}{10} & \text{if } threshold\_near\_last\_maximum() \\ threshold * growFactor & \text{else} \end{cases}$

Evaluation

Setup

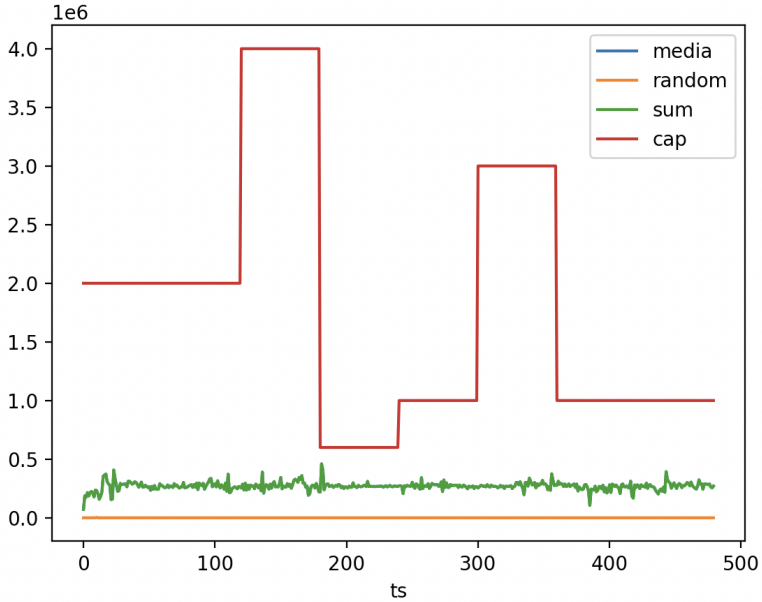
- Virtualized Ubuntu
- Small Program utilizing GStreamer[1]
  - Stream Video from Server to Client using RTP
  - Open a new Stream per RTP frame
  - uses separate stream to send randomly generated bytes at fixed rate
- use Mininet[2] to limit the link capacity

[1] https://gstreamer.freedesktop.org/

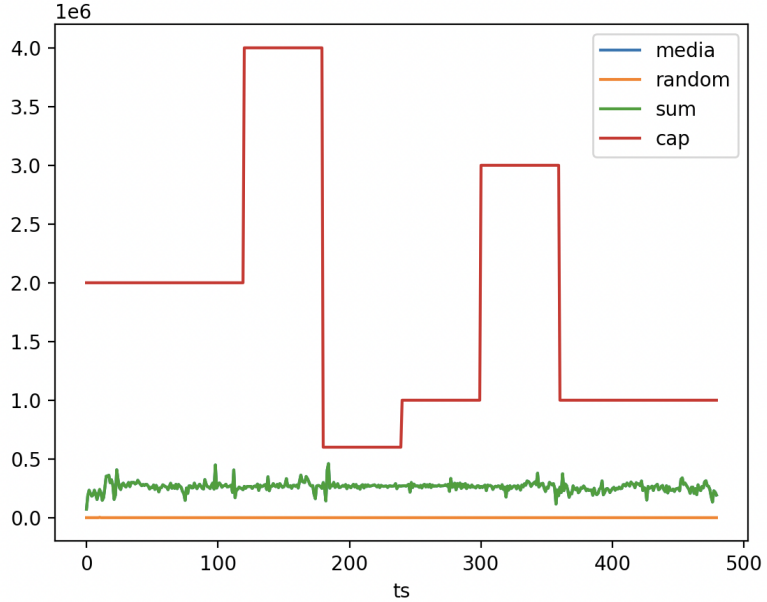
[2] mininet.org

0Kbit/s random data

modified quic-go

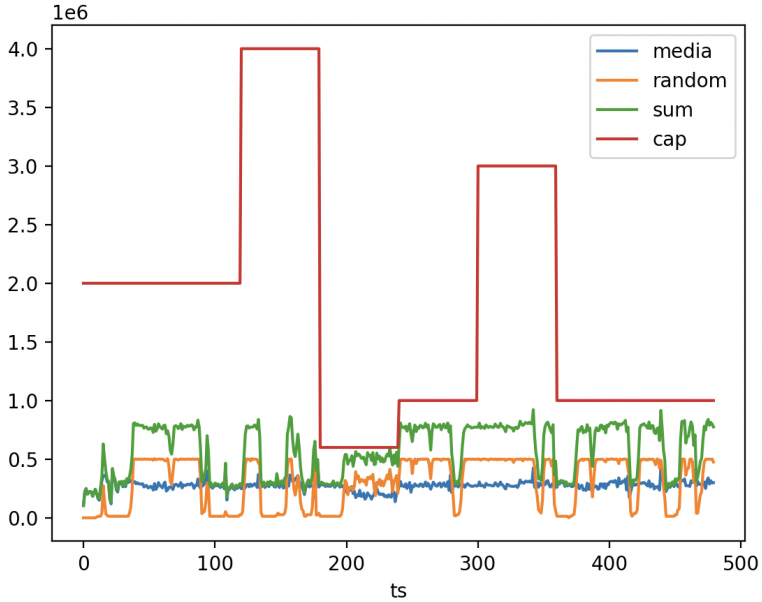


original quic-go

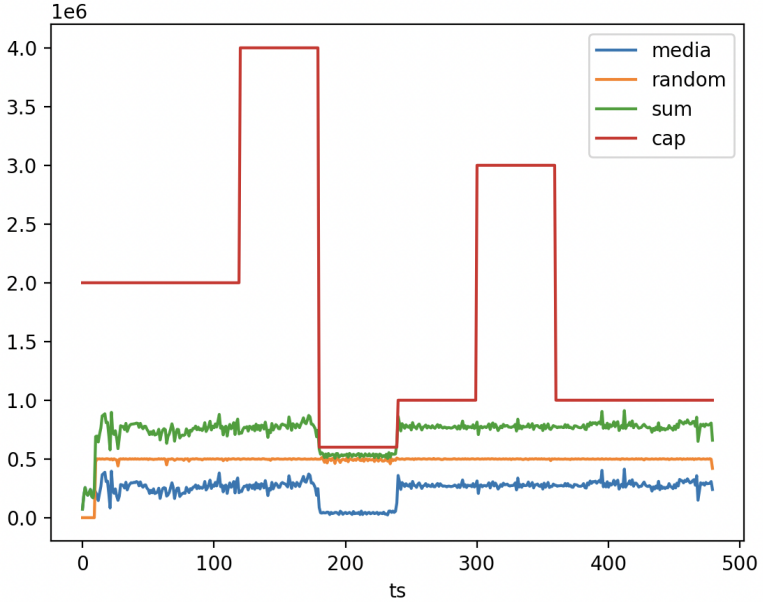


500Kbit/s random data

modified quic-go

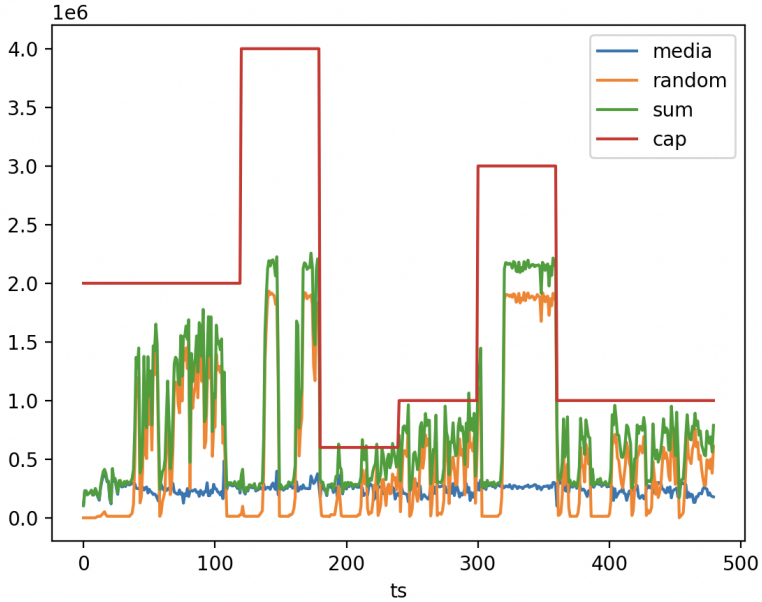


original quic-go

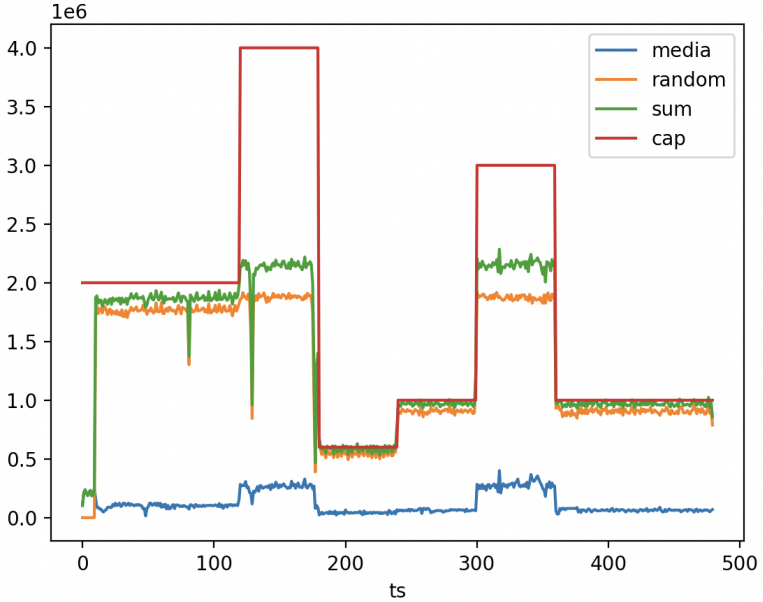


2MBit/s random data

modified quic-go



original quic-go



Flexible Priority-based Stream Schedulers in QUIC

Fátima Fernández
Ikerlan Technology Research Center
Arrasate/Mondragón, Spain
ffernandez@ikerlan.es

Mihail Zverev
Ikerlan Technology Research Center
Arrasate/Mondragón, Spain
mzeverev@ikerlan.es

Luis Diez
Universidad de Cantabria
Santander, Spain
ldiez@tlmat.unican.es

José R. Juárez
Ikerlan Technology Research Center
Arrasate/Mondragón, Spain
jrjuarez@ikerlan.es

Anna Brunstrom
Karlstad University
Karlstad, Sweden
anna.brunstrom@kau.se

Ramón Agüero
Universidad de Cantabria
Santander, Spain
ramon@tlmat.unican.es

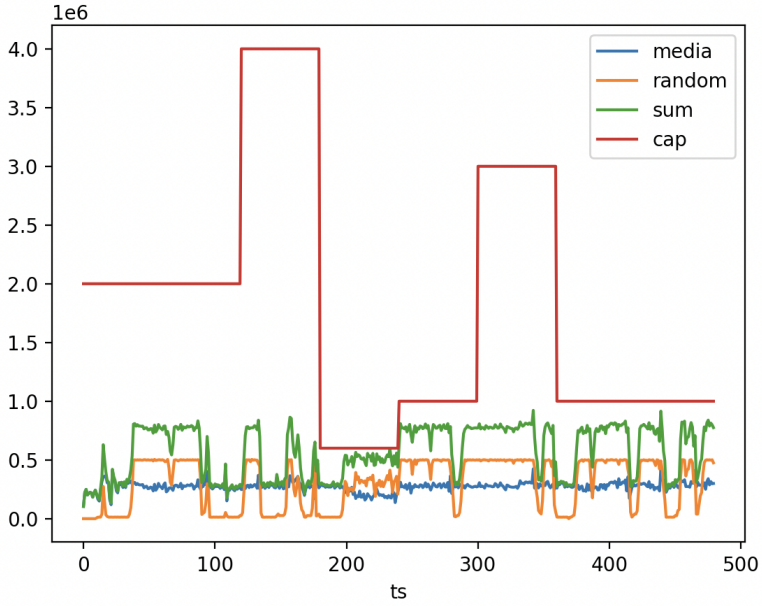
ABSTRACT

The advent of mobile technologies has led to the development of novel services for end-users, with stringent needs and requirements.

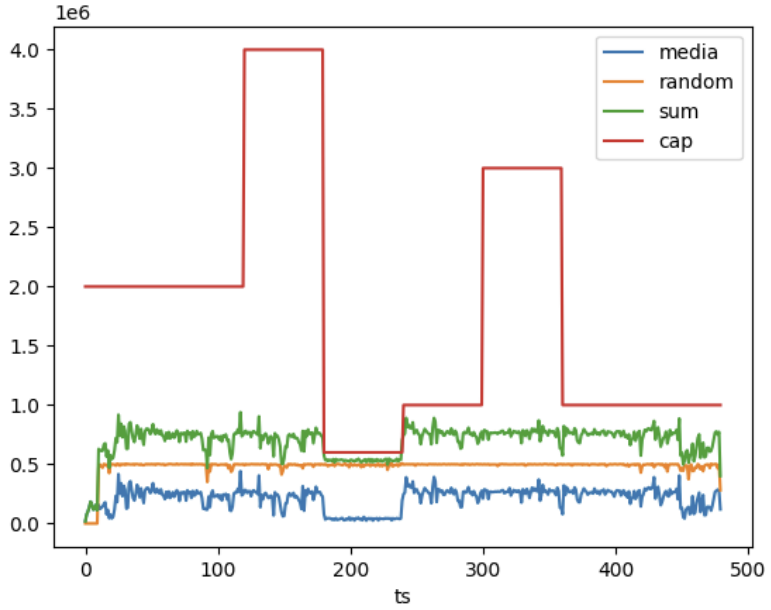
Wireless Ad Hoc, Sensor, & Ubiquitous Networks (PE-WASUN '23), October 30-November 3 2023, Montreal, QC, Canada. ACM, New York, NY, USA, 8 pages.
https://doi.org/10.1145/3616394.3618267

Comparison to strict prioritization

modified quic-go



only prioritization



What is going on here?

Recall the differences between our data!

- Live Video
  - roughly the same amount of data
  - 30 FPS camera: generated every 33ms
- File
  - 4GB of data
  - readily available on the disk

This mismatch leads to most available bandwidth being used by the low priority data!

When new video data becomes available, bandwidth has already been used up!

Key Takeaway: Do not just prioritize using ordering!

Take the volume, how much you have sent into account!

Future Work

- multiple priority classes
- implement as an actual congestion controller
- measure one way delay, ideally per class
- improve benchmarks
  - more frequent changes in capacity
  - measure computational performance
  - more realistic conditions

Meta Learnings

Maybe it wasn't about the results, but the insights we gained along the way

- If you are gonna do it, do it right
- *Good artists copy, great artists steal - Picasso*
- IETF is a complex community

Thank you!