Security Analysis of the Micro Transport Protocol with a Misbehaving Receiver

Florian Adamsky, Syed Ali Khayam, Rudolf Jäger
and Muttukrishnan Rajarajan

florian@adamsky.it
http://florian.adamsky.it

The 4th International Conference on Cyber-Enabled Distributed Computing and Knowledge Discovery 2012

Contents

Contents

Introduction

- BitTorrent is the most widely used Peer-to-Peer (P2P) application
- Before December 2008, TCP was its default transfer protocol
- Since this date, uTorrent switched from TCP to the Micro Transport Protocol (uTP) which is based on UDP

What's the problem with TCP and BitTorrent?

- BitTorrent is in most cases background traffic
- BitTorrent uses multiple connection at once
  - TCP distributes the available bandwidth evenly across connections
  - The more connections one application uses, the larger the share of bandwidth
- Old Solution: Configure bandwidth limit
  - Requires knowledge about the own Internet connection
  - Often lets big head room which is unused
  - Bad configured BitTorrent client can harm other people in the network

$\Rightarrow$ Novel transport protocol based on UDP: Micro Transport Protocol (uTP)

The next Internet meltdown?

The Register®
Biting the hand that feeds IT

Original URL: http://www.theregister.co.uk/2008/12/01/richard_bennett_utorrent_udp/

Bittorrent declares war on VoIP, gamers

The next internet meltdown

By **Richard Bennett**

Posted in Networks, 1st December 2008 12:29 GMT

Gamers, VoIP and video conference users beware. The leading BitTorrent software authors have declared war on you - and any users wanting to wring high performance out of their networks. A key design change in the P2P application promises to make the headaches faced by ISPs so far look like a party game. So what's happened, and why does it matter?

Upset about Bell Canada's system for allocating bandwidth fairly among internet users, the developers of the uTorrent P2P application have decided to make the UDP protocol the default transport protocol for file transfers. BitTorrent implementations have long used UDP to exchange tracker information – the addresses of the computers where files could be found – but the new release uses it in preference to TCP for the actual transfer of files. The implications of this change are enormous.

As BitTorrent implementations follow uTorrent's lead – and they will, since uTorrent is owned by BitTorrent Inc, and is regarded as the canonical implementation – the burden of reducing network load during periods of congestion will shift to the remaining TCP uses, the most important of which are web browsing and video streaming.

By most estimates, P2P accounts for close to half of internet traffic today. When this traffic is immune to congestion control, the remaining half will stumble along at roughly a quarter of the bandwidth it has available today: half the raw bandwidth, used with half efficiency, by 95% of internet users. Oops.

When your internet bandwidth is divided by four, you're going to notice. Even the downloading fiends who haunt the message boards at Broadband Reports can see this, as several have noted [1]:

> **Is bypassing TCP congestion control a good thing for the users of the network? Why should one persons [sic] non-interactive file sharing generating a dozen to a hundred streams be more important than my interactive VoIP call or gaming experience?**
>
> **Using it as a feature, maybe, but enabling this behavior by default is just wrong and will lead to continuing counter, counter**

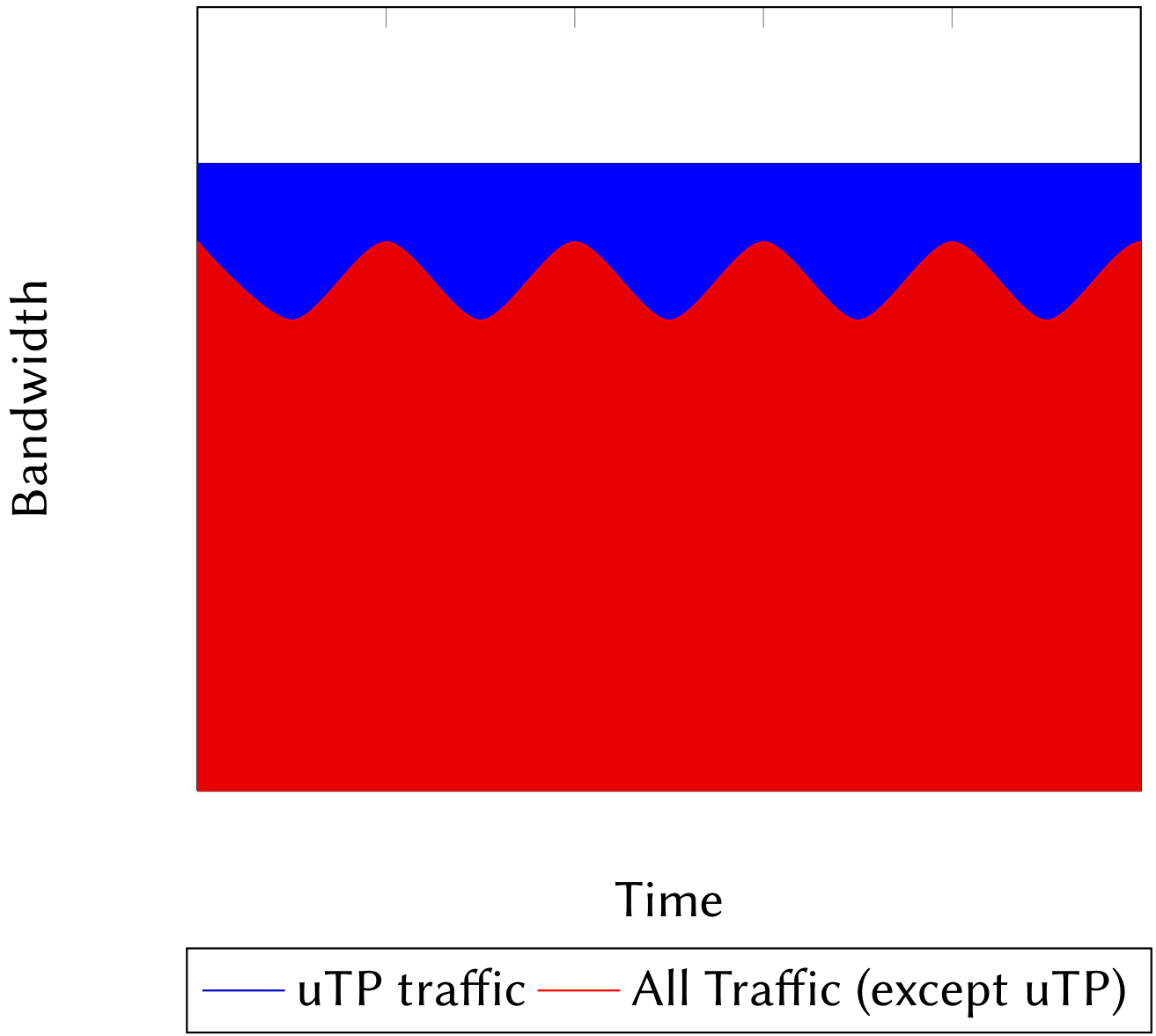
Why the sky isn't falling

http:

Micro Transport Protocol (uTP)

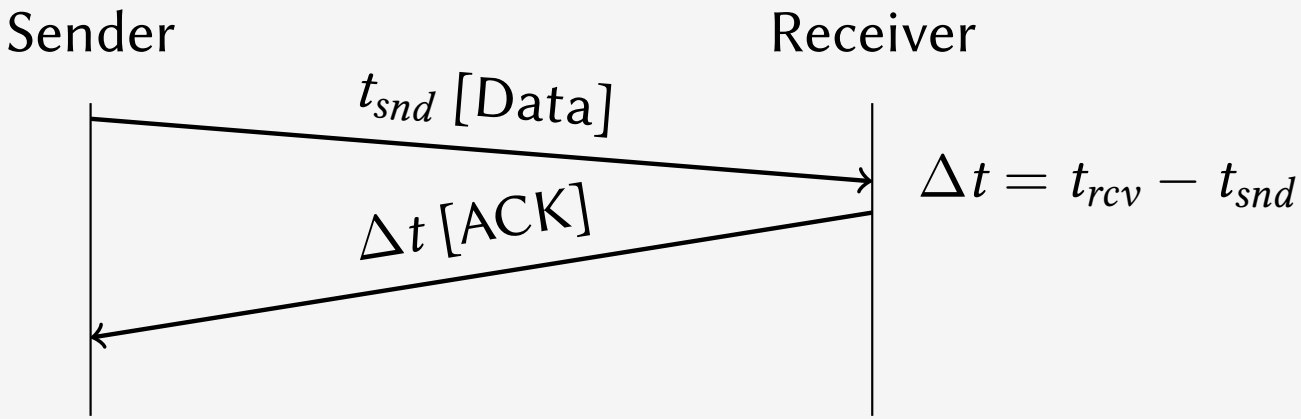
- uTP is a reliable transport protocol which makes use of UDP
  - Introduces a new congestion control: Low Extra Delay Background Transport (LEDBAT)
- Similarities to TCP
  - Window based flow control
  - Sequence numbers and ACK numbers
- Differences to TCP
  - Sequence numbers and ACKs refer to packets, not bytes
  - SACK uses a 32 bitmask where each bit represents a packet
  - No congestion control (Slow-start, congestion avoidance, ...) $\rightarrow$ LEDBAT
- Advantages:
  - Detects foreground traffic and automatically slows down
  - Scales better when sending over many connections at once

Traffic



LEDBAT (Congestion Control)

- LEDBAT uses one way delay as the main congestion measurement
  - Will be compared with previous values (not absolute values)



- And also:
  - Packet loss (max_window $=$ max_window $\times$ 0.78)
  - ACKs during one window
- IETF just tries to standardize LEDBAT (Draft 09[3])

[3] http://tools.ietf.org/id/draft-ietf-ledbat-congestion-09.txt

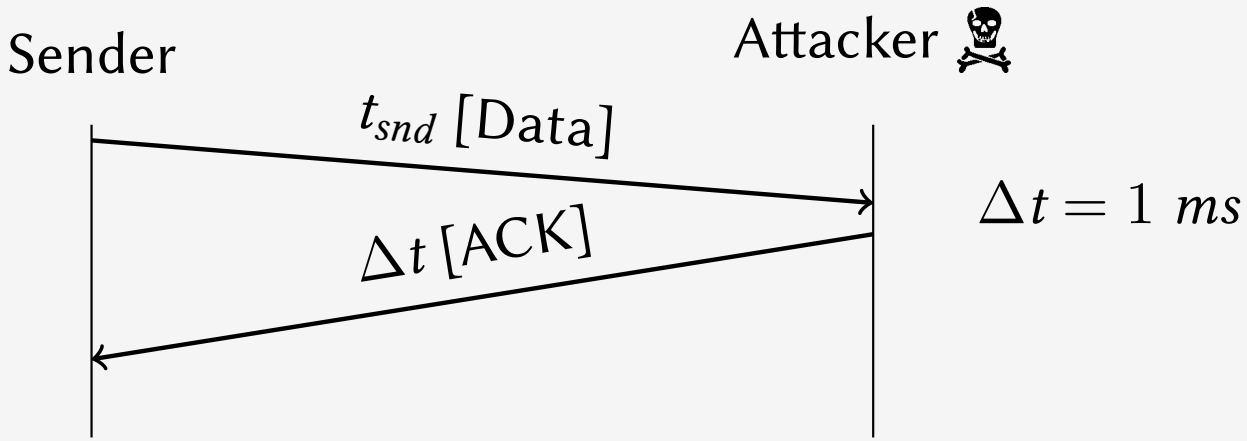
Contents

Attack Idea 1 (Delay Measurement)

Idea

Lying about the one way delay to induce the sender to send more and more packets into the network

How to do that?



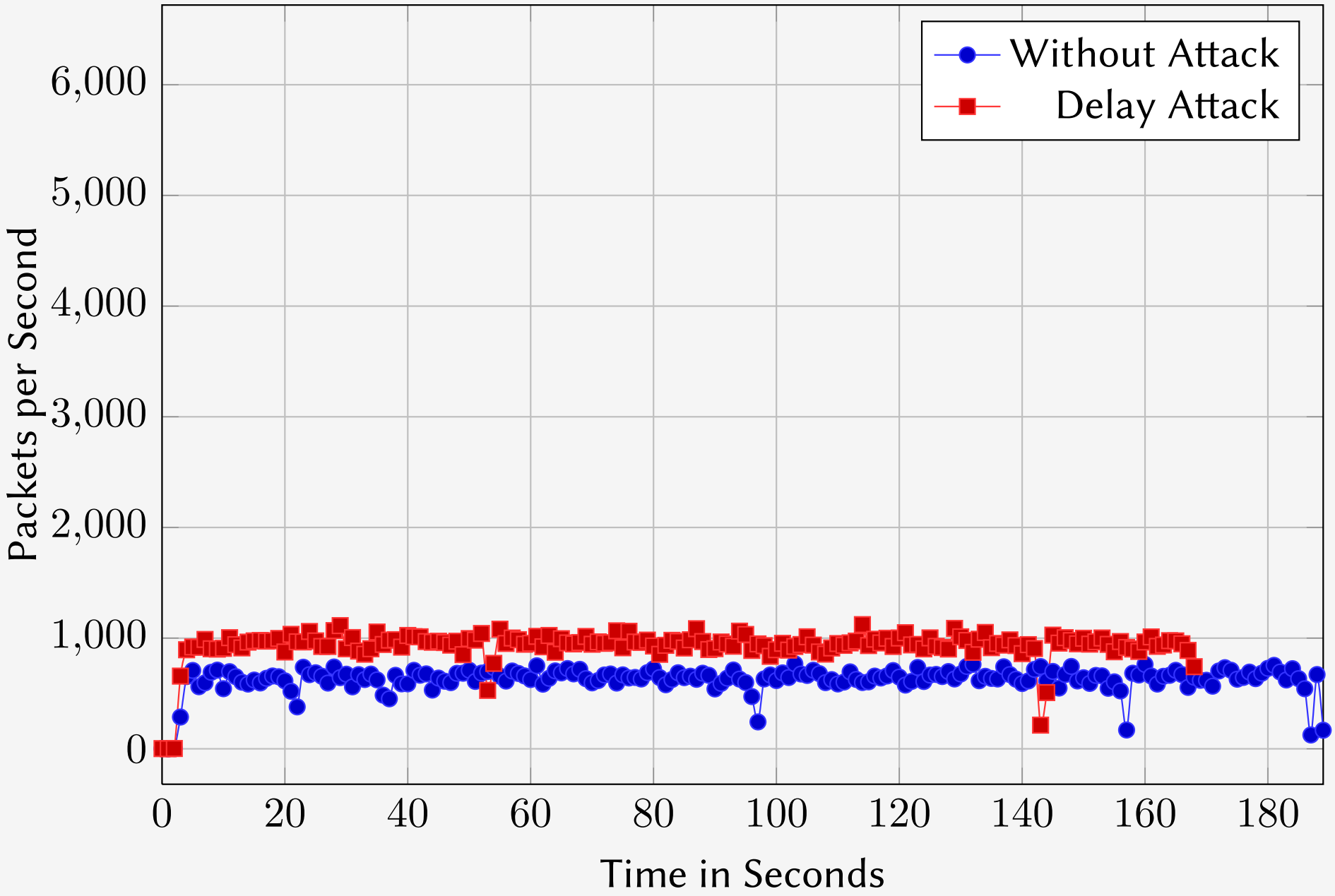
Code Snippet

```
1 scaled_gain    = MAX_CWND_INCREASE_PACKETS_PER_RTT
2                  * delay_factor * window_factor;

4 max_window     += scaled_gain;
```

Delay Calucation

$$\begin{aligned} \text{delay\_factor} &= \frac{100\ ms - min(\vec{our\_hist})}{100\ ms} \\ \text{delay\_factor} &= \frac{100\ ms - 0}{100\ ms} \\ \text{delay\_factor} &= 1 \end{aligned}$$

Bandwidth Usage (100 MiB File Transfer)



Attack Idea 2 (Packet loss)

Idea

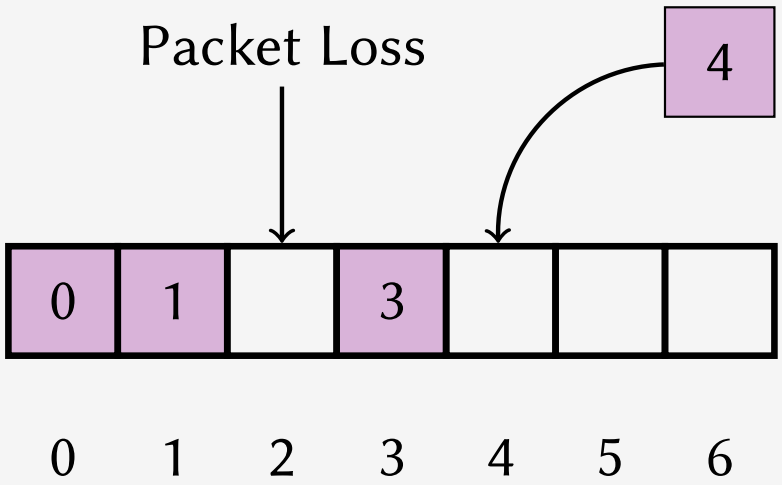
We do not inform the sender about packet loss

How to do that?

- As an attacker we're not interested in data integrity
- Save all packets we got sequentially and not according to there sequence number

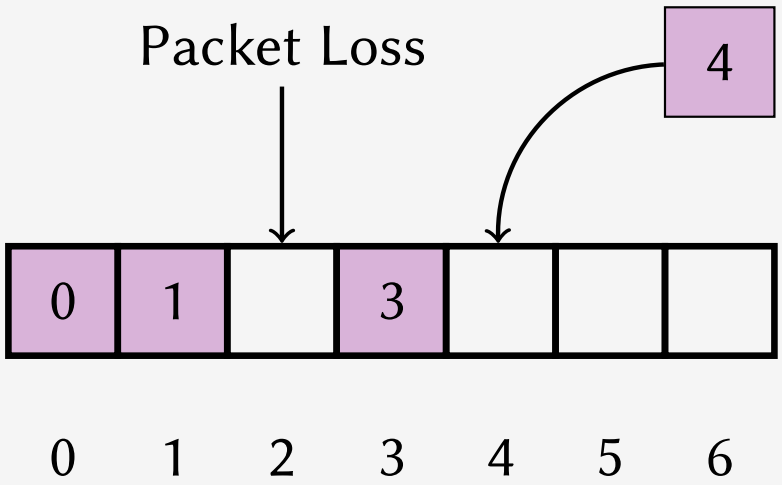
Attack Idea 2

Normal packet sorting

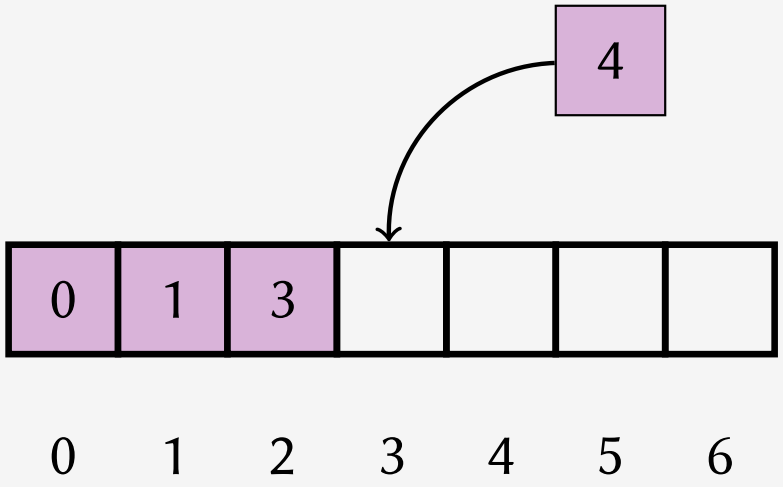


Attack Idea 2

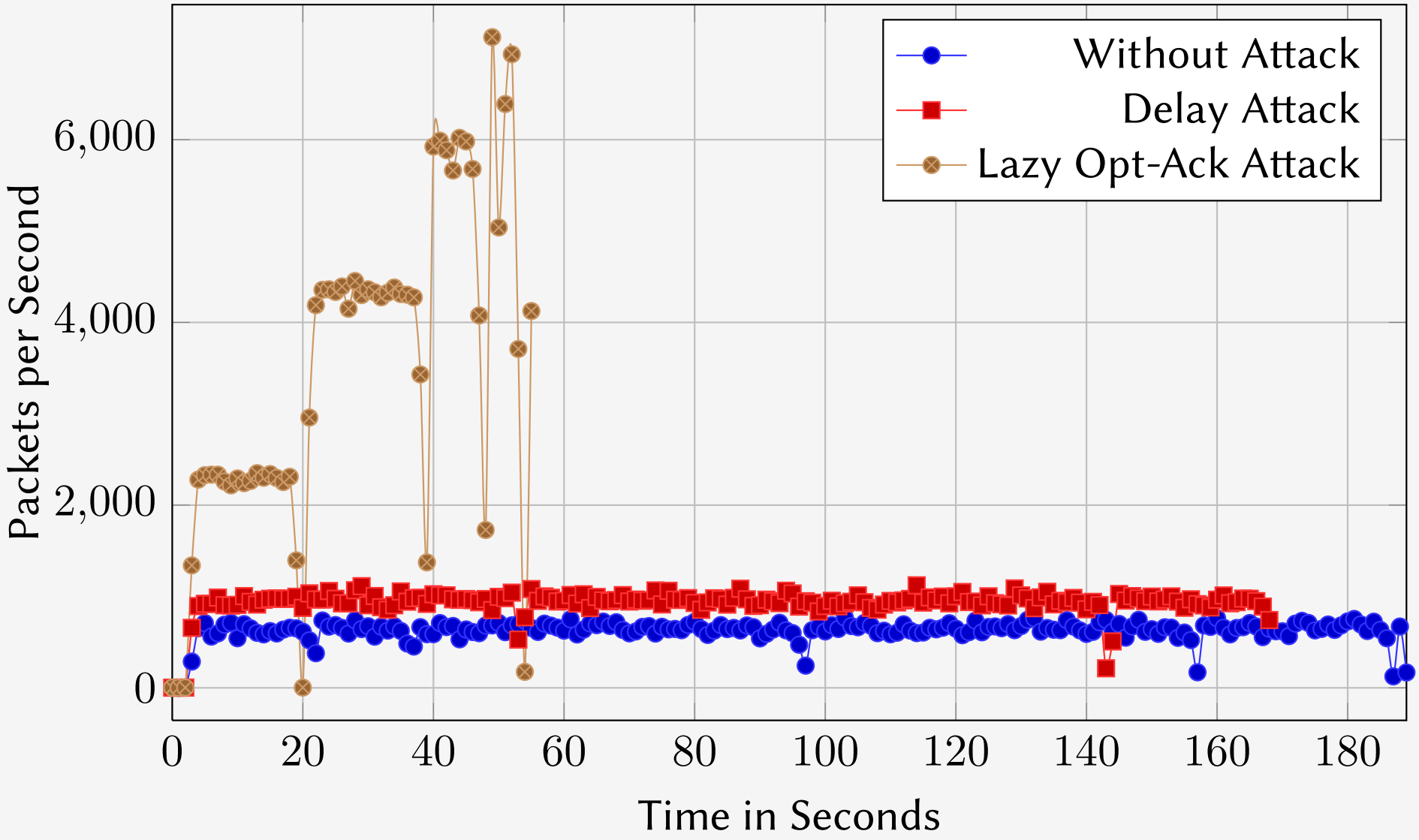
Normal packet sorting



Attack: Packet sorting sequentially



Bandwidth Usage with Loss Attack



Attack Idea 3 (Selective Acknowledgment)

Idea

Pretend that we got all packets always via SACK

How to do that?

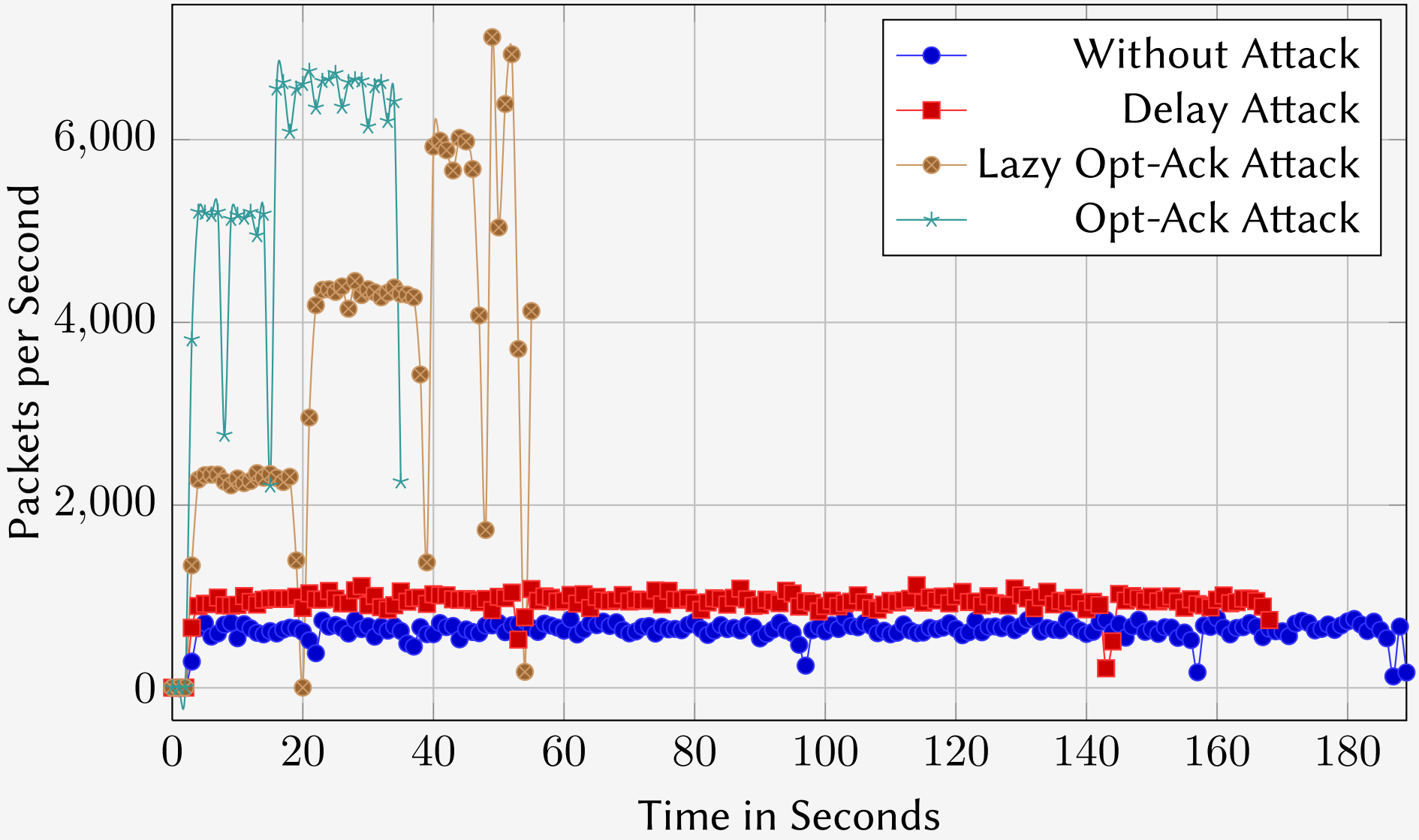
- Again: As an attacker we're not interested in data interested
- Set all bits of the SACK bitmask to the value 1
- Replace that code:

```
1       uint m = 0;             // SACK bitmask
```

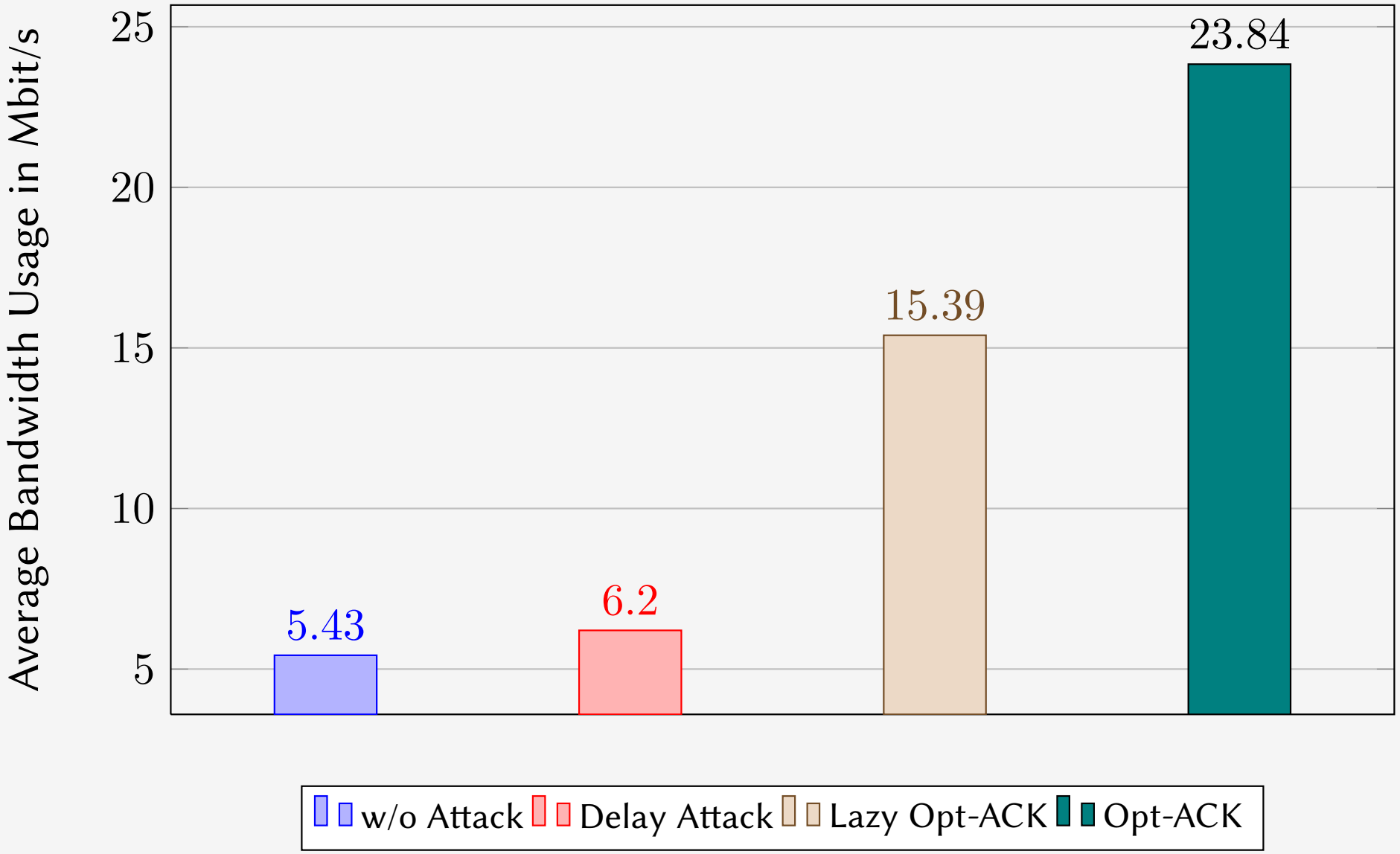
- with this code

```
1       uint m = UINT_MAX; // SACK bitmask
```

Bandwidth Usage with SACK Attack



Bandwidth from all Attacks



Can we create Congestion?

- Congestion Experiment
  - Constant 50 Mbit/s UDP stream with iPerf
  - Parallel to this file transfer via uTP

Can we create Congestion?

- Congestion Experiment
  - Constant 50 Mbit/s UDP stream with iPerf
  - Parallel to this file transfer via uTP

Results

Client	Packet loss
Without Attack	0 %
Delay Attack	0 %

Can we create Congestion?

- Congestion Experiment
  - Constant 50 Mbit/s UDP stream with iPerf
  - Parallel to this file transfer via uTP

Results

Client	Packet loss
Without Attack	0 %
Delay Attack	0 %
Lazy Opt-Ack Attack	7 %

Can we create Congestion?

- Congestion Experiment
  - Constant 50 Mbit/s UDP stream with iPerf
  - Parallel to this file transfer via uTP

Results

Client	Packet loss
Without Attack	0 %
Delay Attack	0 %
Lazy Opt-Ack Attack	7 %
Opt-Ack Attack	42 %

Contents

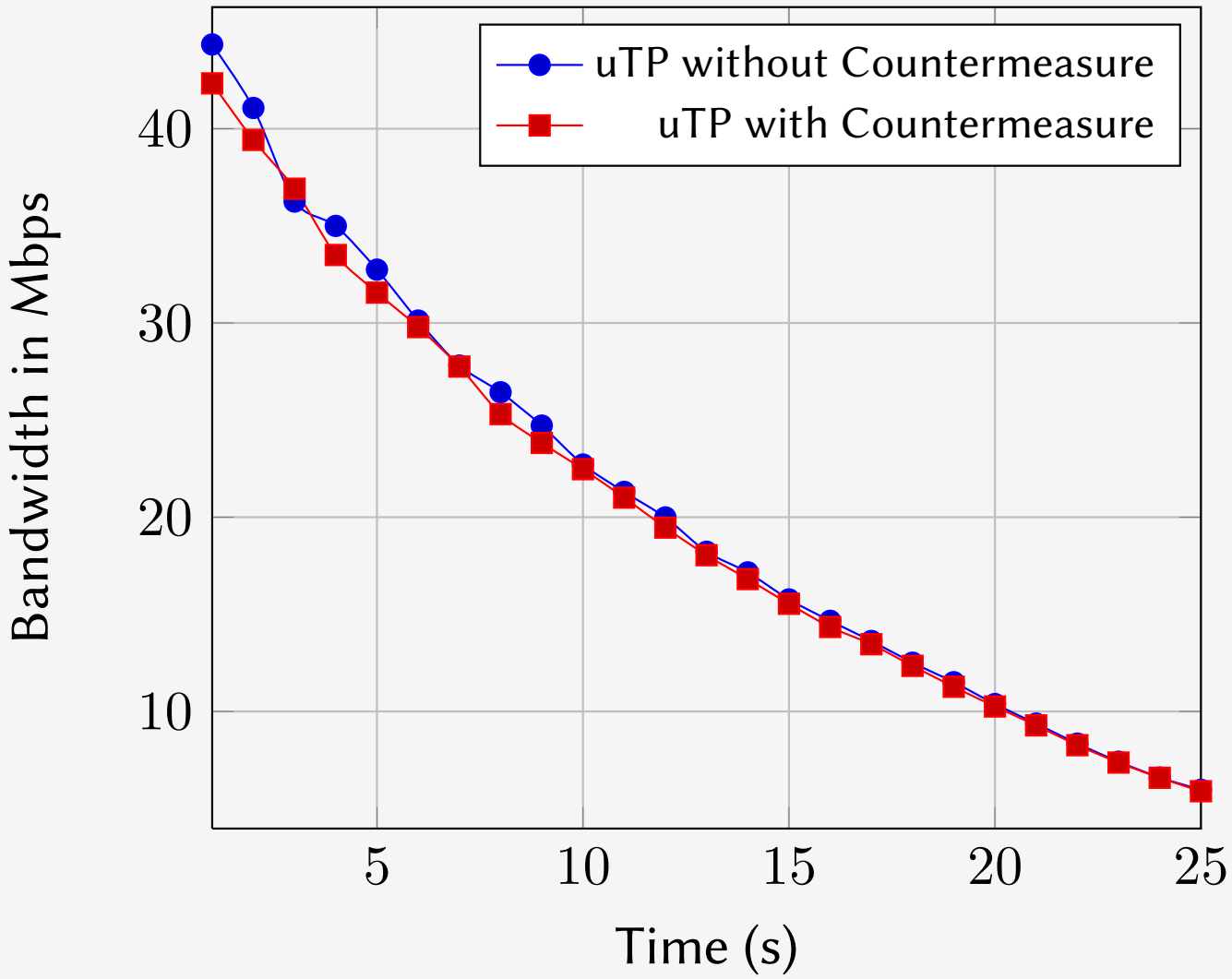
Randomly Skipped Packets

Idea

A sender randomly skips packets and checks if the receivers acknowledge those packets.

- Countermeasure only against the lazy opt-ack and opt-ack attack
- This countermeasure will reduce the performance a little bit

Performance Analysis



Contents

Conclusion

The main contributions of our paper are:

- Proposed the following three attacks against the novel uTP:
  - delay attack
  - lazy optimistic attack
  - optimistic attack
- Shown a detail evaluation and shown the severity of those attacks
- Presented a countermeasure against those attacks

Thank You

Questions?

florian@adamsky.it
http://florian.adamsky.it

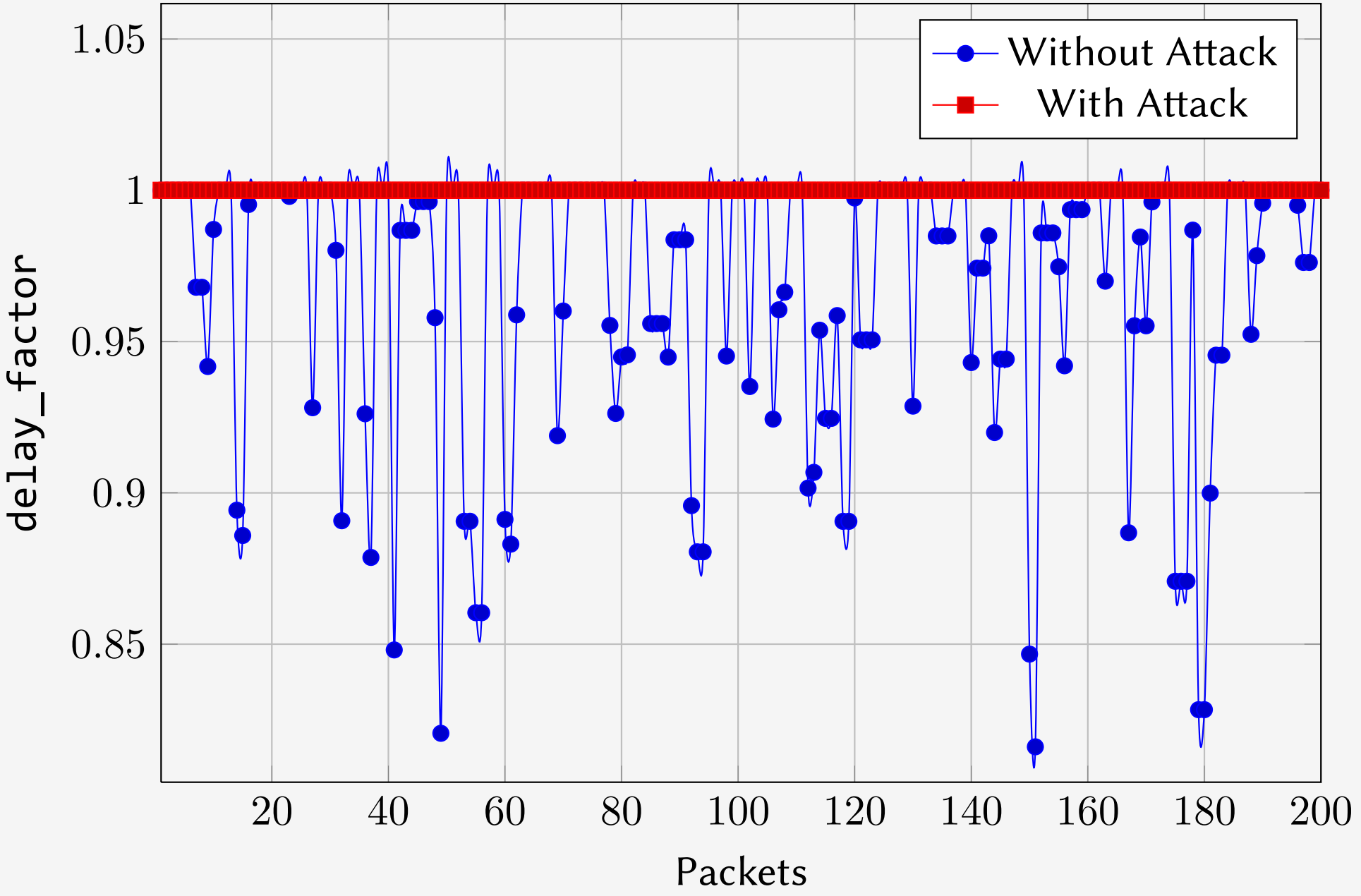
Institutions:

CITY UNIVERSITY LONDON

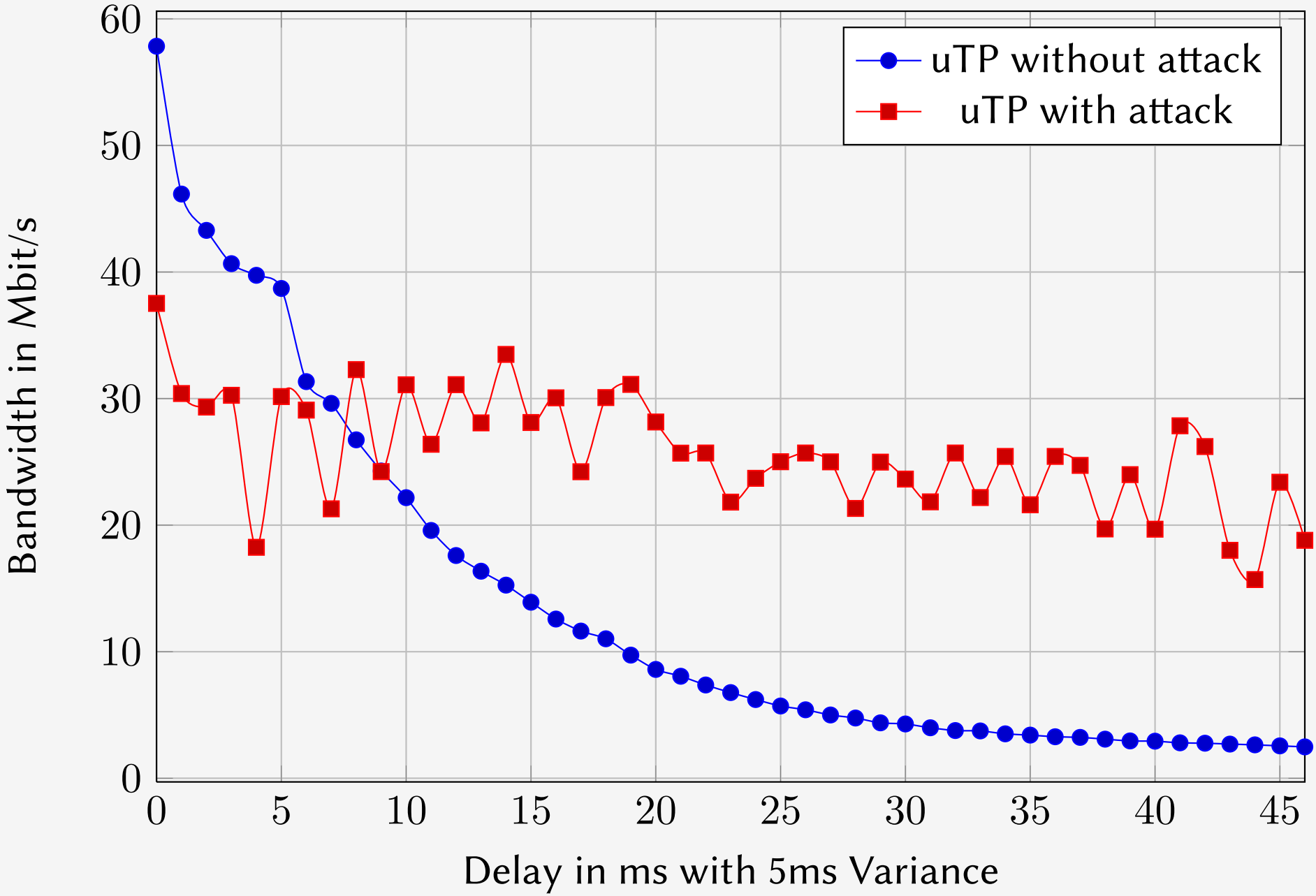
NUST
NATIONAL UNIVERSITY OF SCIENCES & TECHNOLOGY

THM
TECHNISCHE HOCHSCHULE MITTELHESSEN

Delay Comparison



Delay Comparison



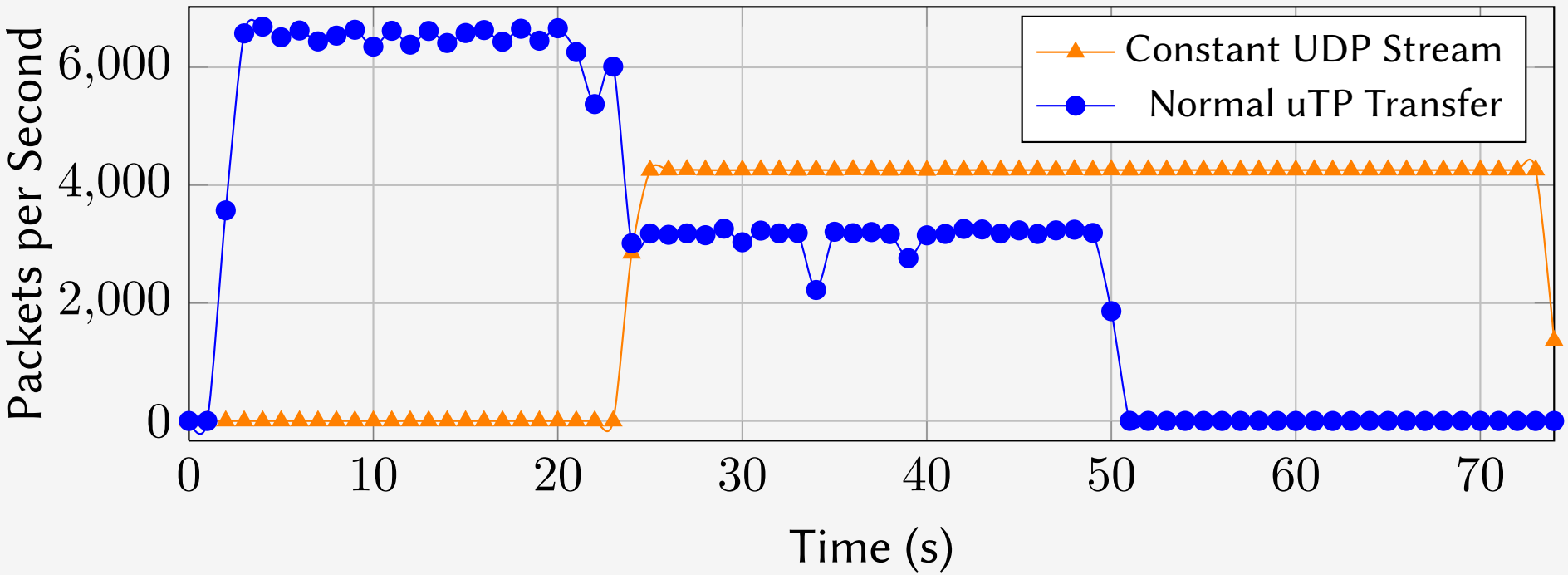
libutp

```
1       uint m = 0; // SACK bitmask

3       for (size_t i = 0; i < window; i++) {
4           if (inbuf.get(ack_nr + i + 2) != NULL) {
5               m |= 1 << i;
6           }
7       }
```

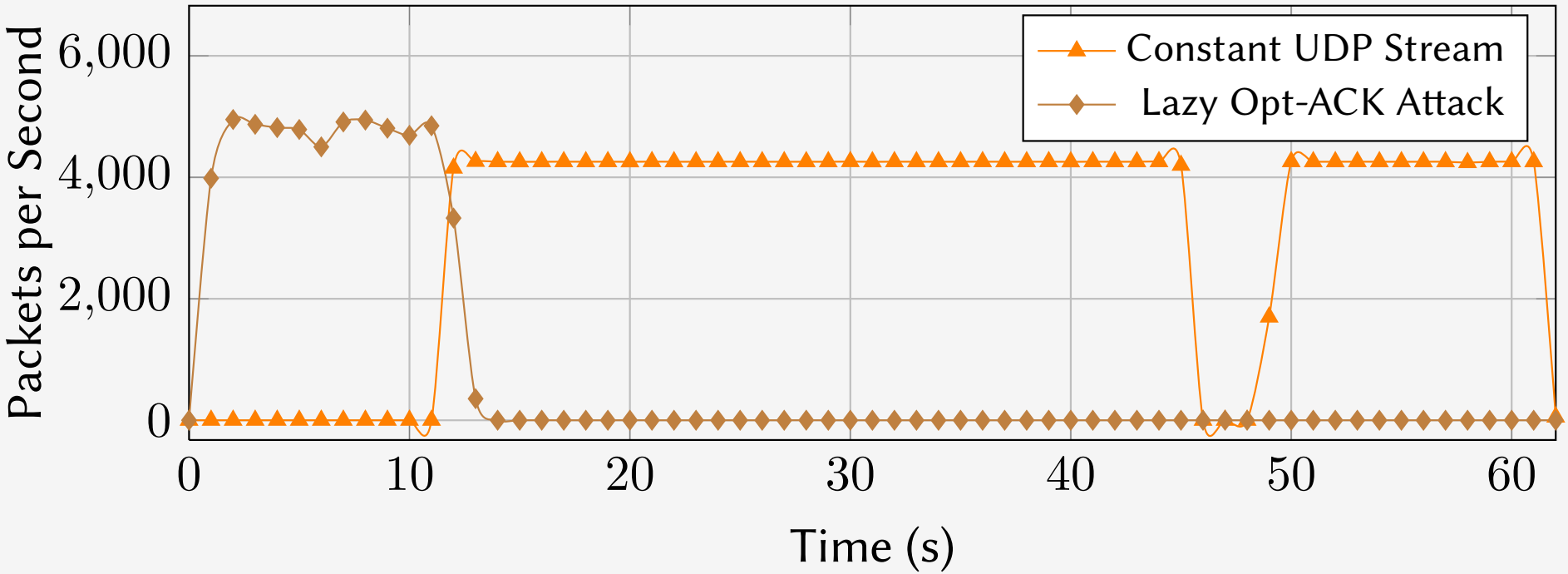
libutp

```
1       uint m = UINT_MAX; // SACK bitmask

3       for (size_t i = 0; i < window; i++) {
4           if (inbuf.get(ack_nr + i + 2) != NULL) {
5               m |= 1 << i;
6           }
7       }
```

Congestion Experiment (w/o Attack)



Congestion Experiment (Lazy Opt-ACK)



Congestion Experiment (Opt-ACK Attack)