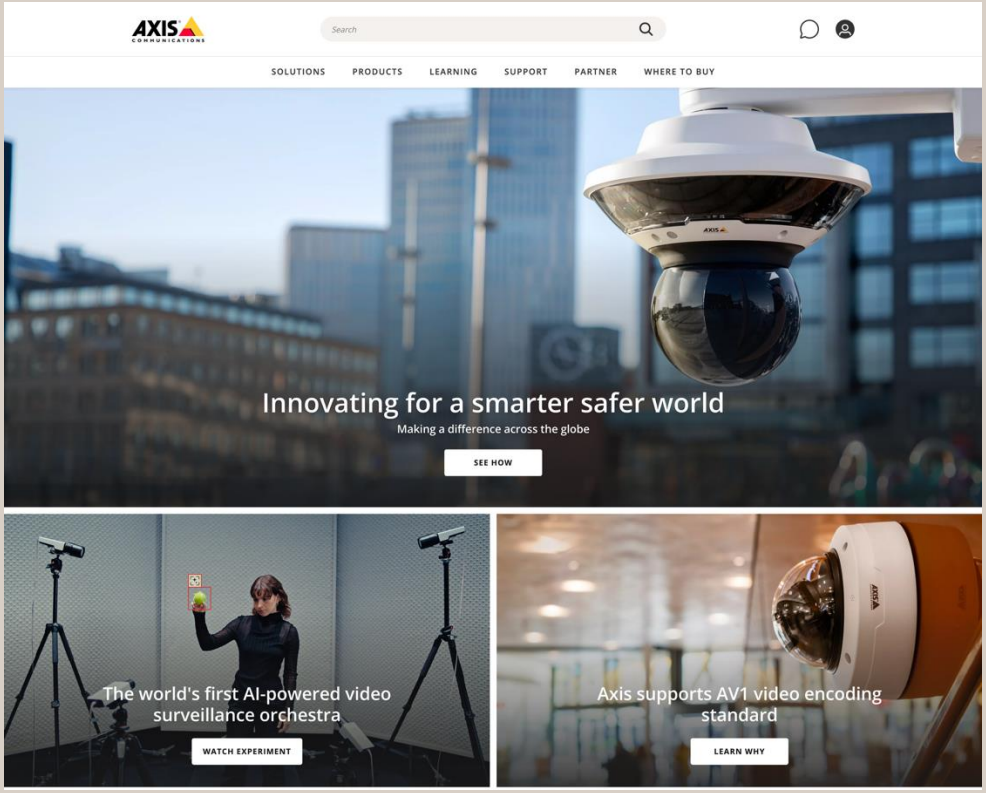
24 October 2025

Improving WebRTC data channel performance

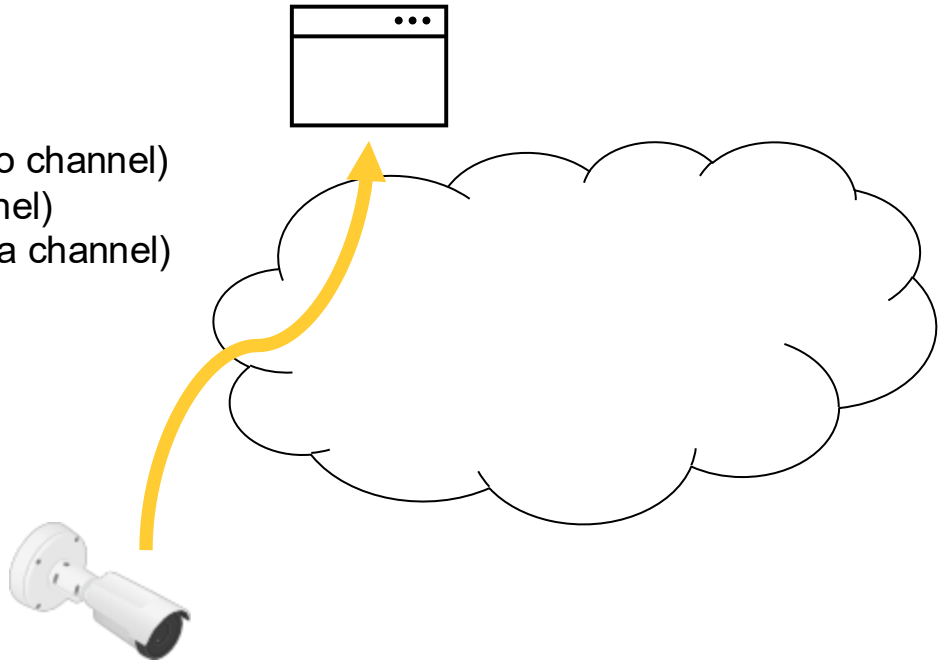
Emil Ljungdahl, Axis Communications

Axis Communications



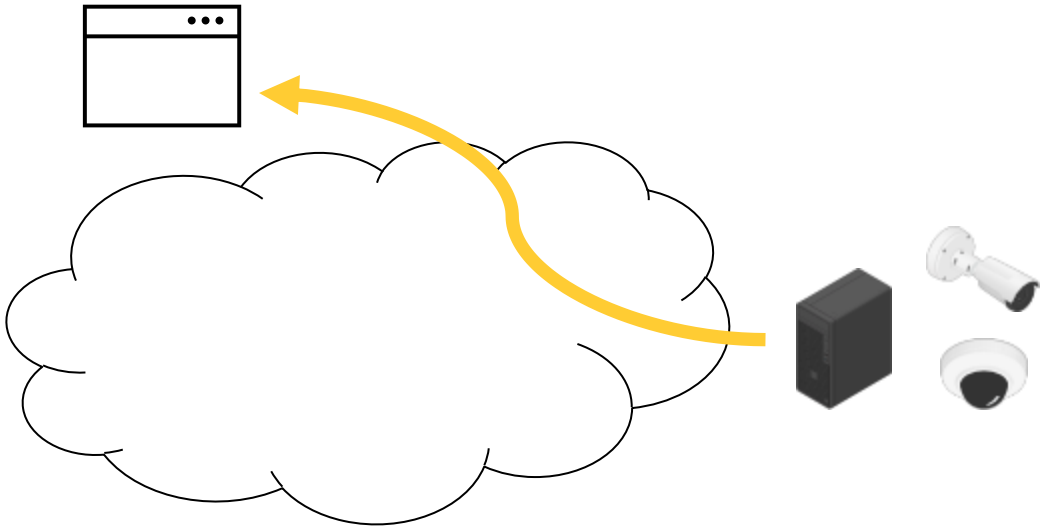
WebRTC at Axis

Live (video/audio channel)
PTZ (data channel)
Recordings (data channel)



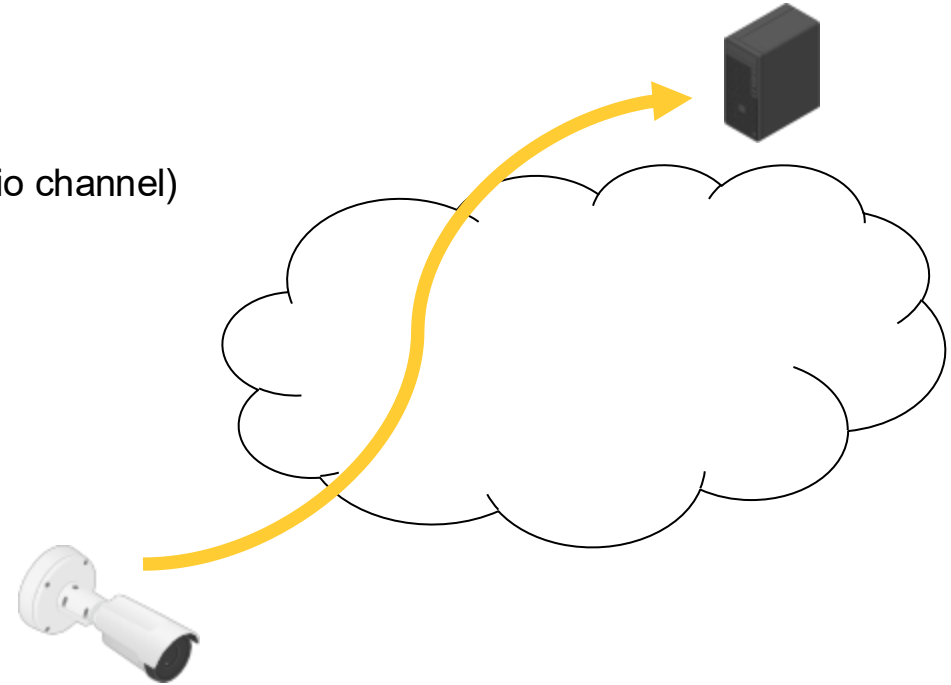
WebRTC at Axis

Live (video/audio channel)
PTZ (data channel)
Recordings (data channel)

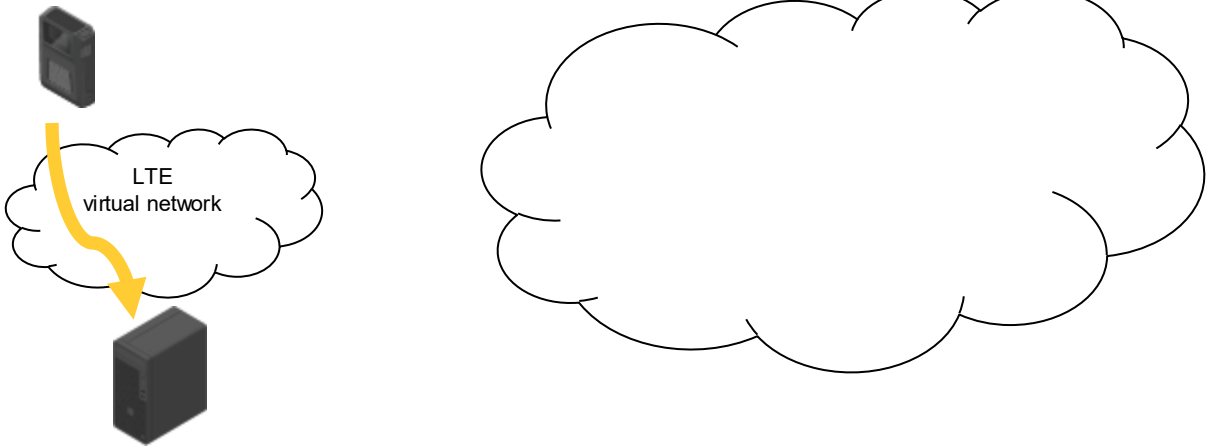


WebRTC at Axis

Live (video/audio channel)

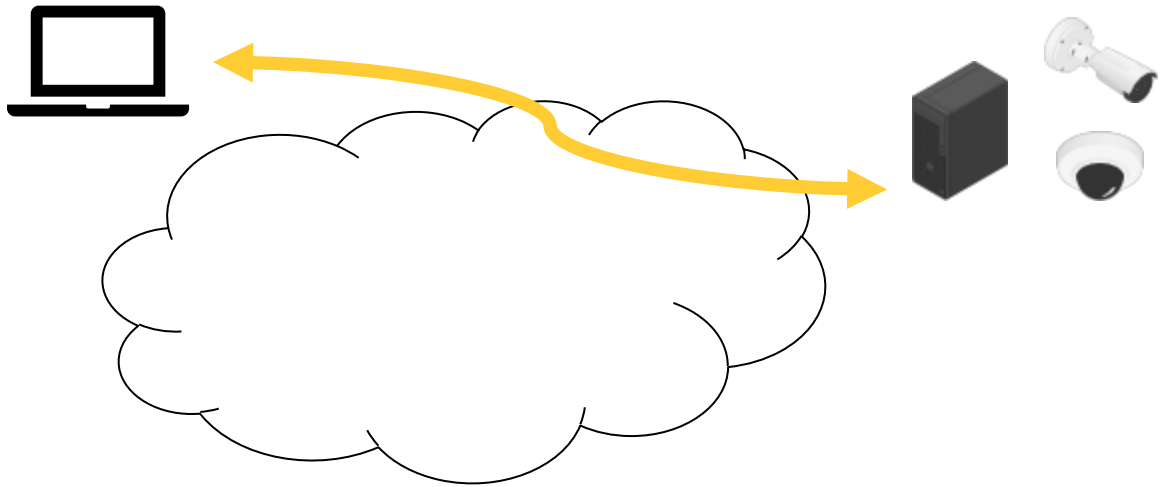


WebRTC at Axis



WebRTC at Axis

Tunnels (data channel)



WebRTC data channels

- For all non-media data
- Ordered vs unordered messages
- Reliable vs Unreliable messages
- Text & Binary message types
- SCTP
  DTLS
  ICE/UDP
- 1 WebRTC data channel = 1 SCTP stream
- Congestion control & loss detection similiar to TCP
  - Selective ACK (SACK)
  - Mostly sender side logic

SCTP in GStreamer

> usrSCTP

> Wrapper code around usrSCTP

Testing data channel throughput

- Different environments brings different challenges
  - P2P vs TURN
  - Latency
  - Jitter
  - Packet re-ordering
- What to test?
  - Environments similar to what we think the users are in!
- In Axis case: most data sent FROM GStreamer code

Testing data channel throughput

> Focus on Axis use case

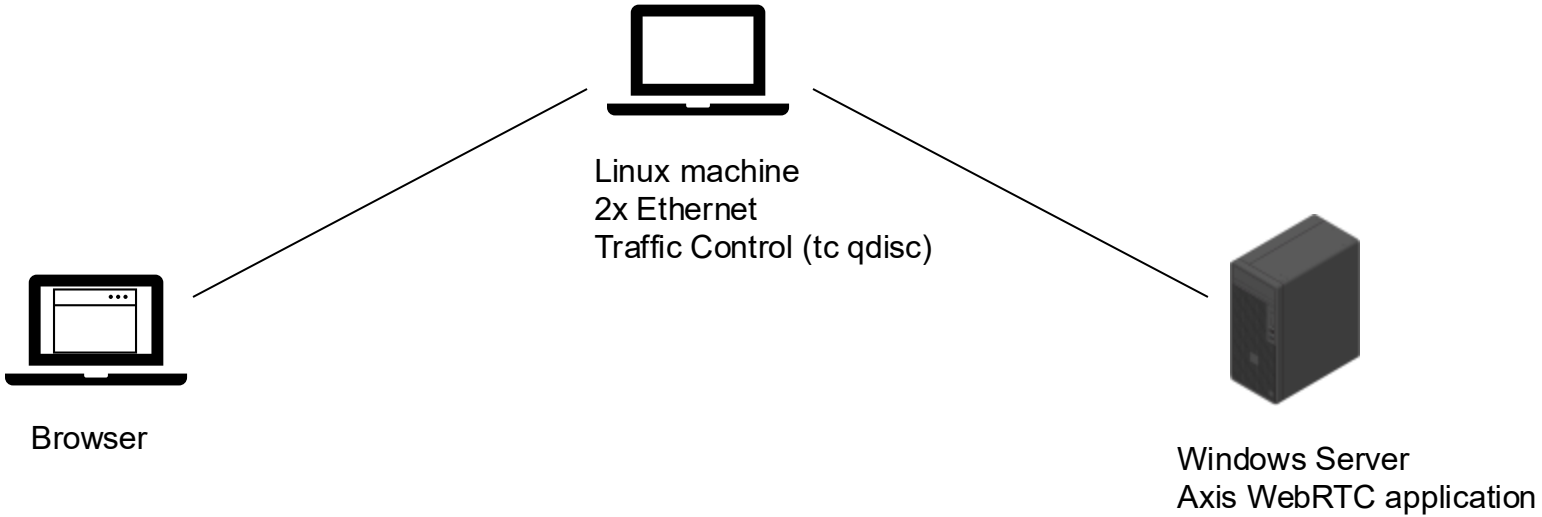
> Simplified

> Absolute numbers not so important!

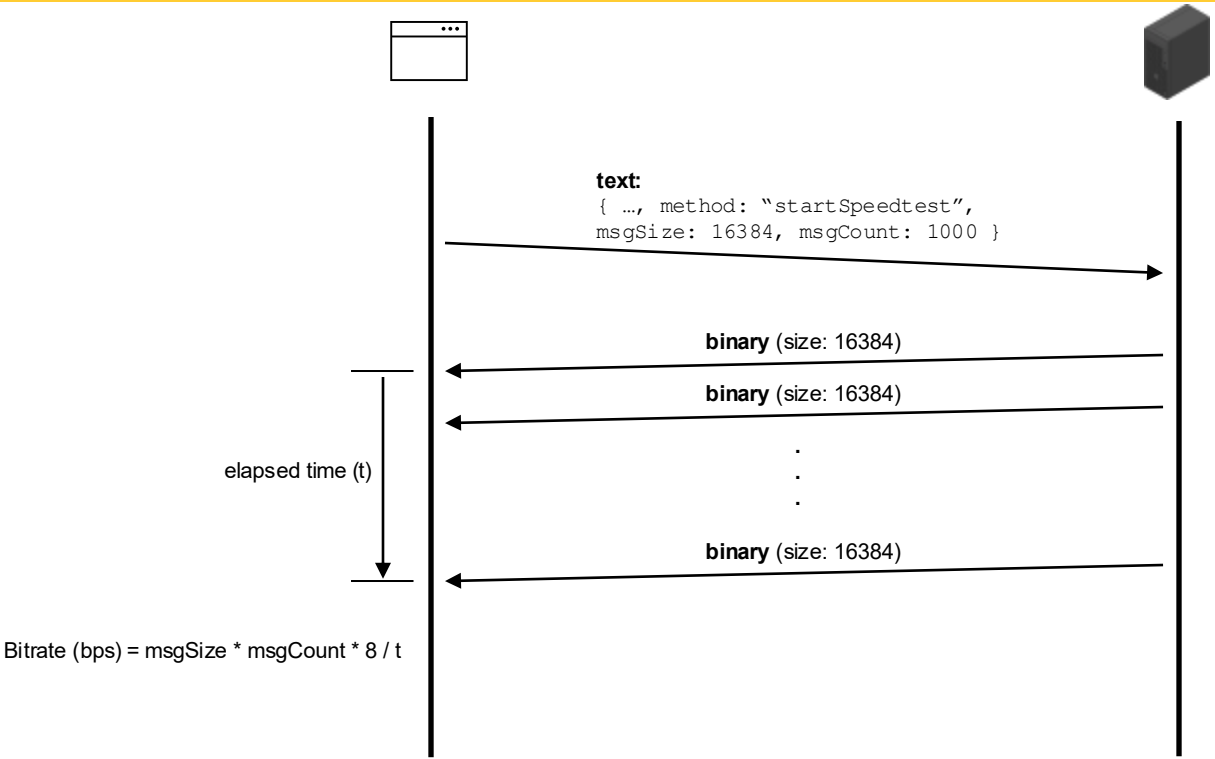
Testing data channel throughput

- Local network
  - Low latency
  - No packet re-ordering
- Long distance link
  - High latency
  - No (or almost no) packet re-ordering
- Mobile network
  - High latency
  - Packet re-ordering (typically caused by low layer re-transmits)

Test setup: environment



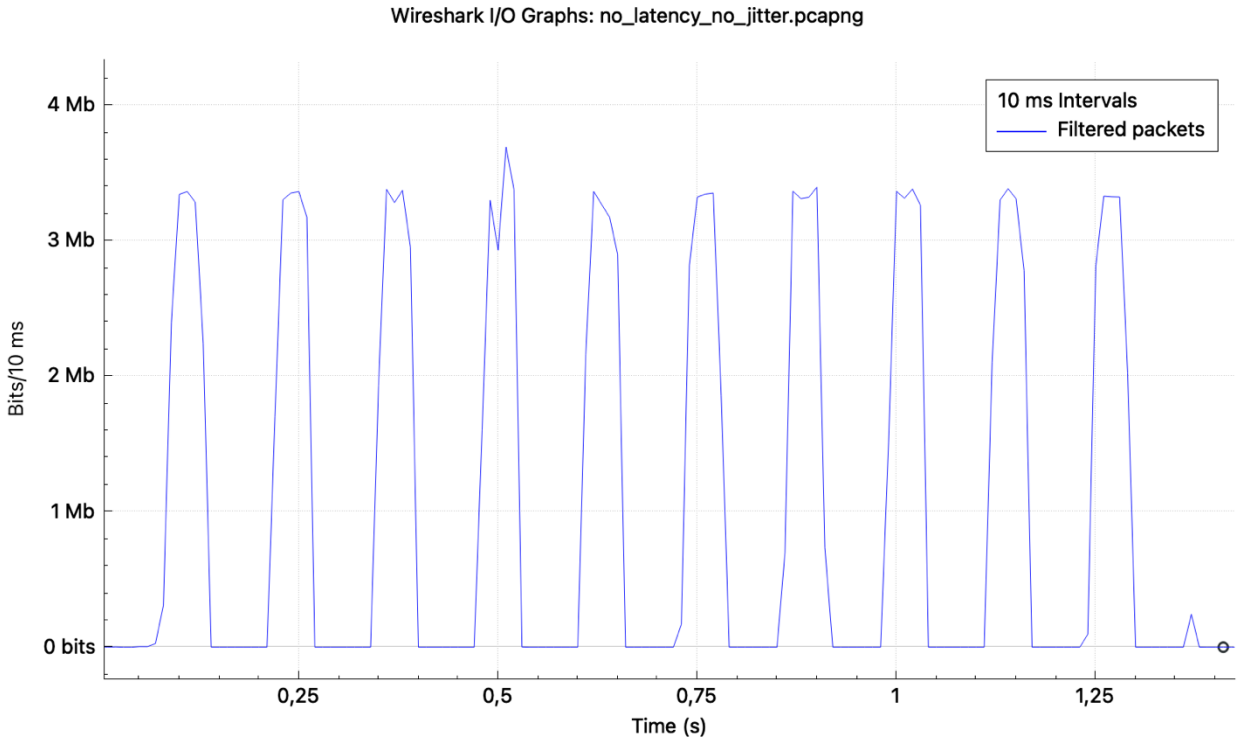
Test setup: data channel protocol



Improvement #1: Before

- Local network
  - Traffic control in bypass mode
- Result: ~100 Mbit/s
- TCP comparison: ~300 Mbit/s

Improvement #1: Before



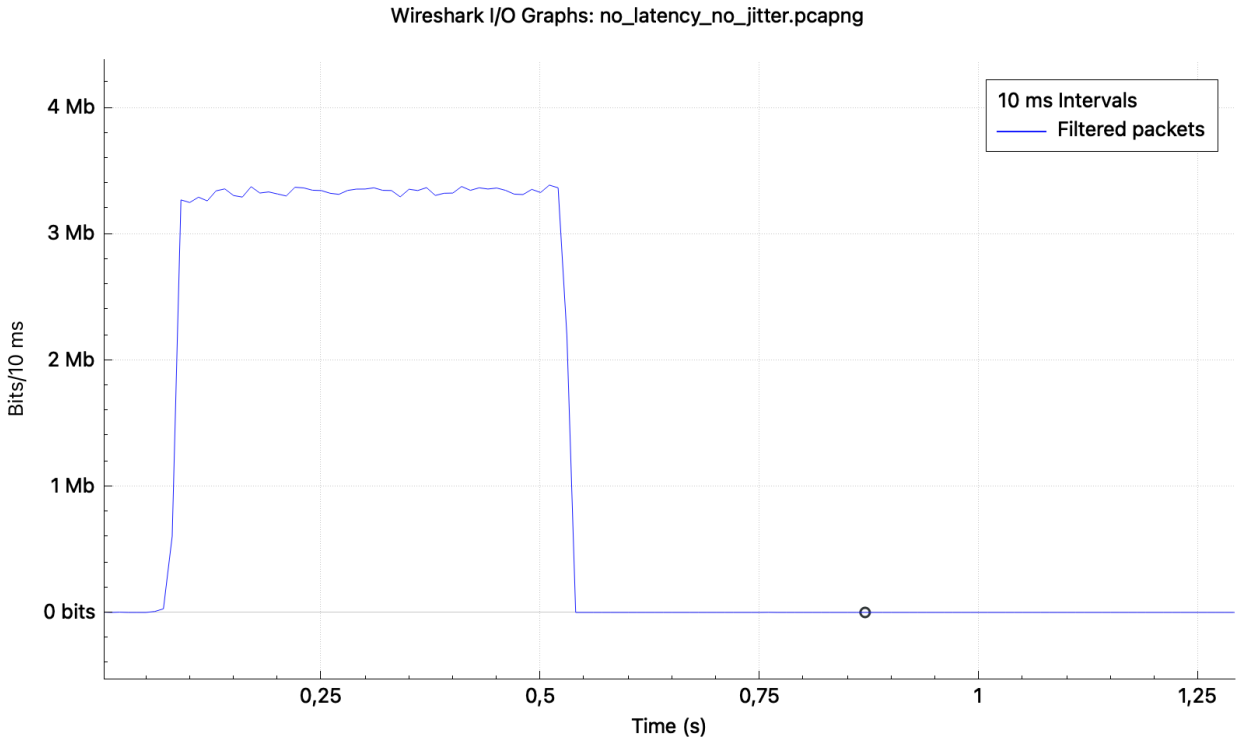
Improvement #1: The fix

> In gstsctpenc.c, gst_sctp_enc_sink_chain():

```
…
#define BUFFER_FULL_SLEEP_TIME 100000
…
gint64 end_time = g_get_monotonic_time () + BUFFER_FULL_SLEEP_TIME;
…
/* The buffer was probably full. Retry in a while */
g_cond_wait_until (&sctpenc_pad->cond, &sctpenc_pad->lock, end_time);
…
```

> Change BUFFER_FULL_SLEEP_TIME to 1000

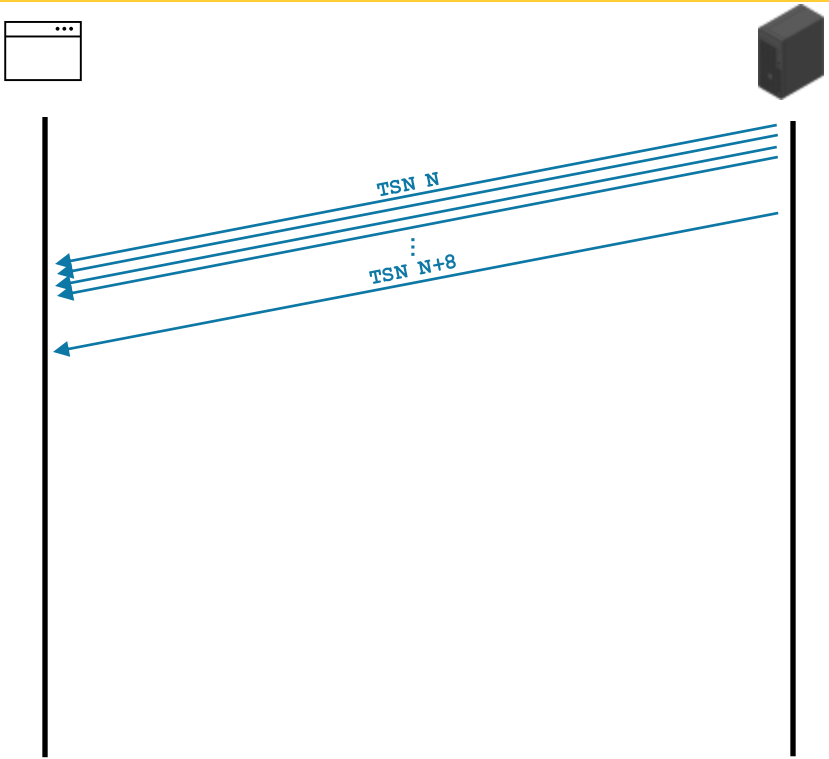
Improvement #1: After



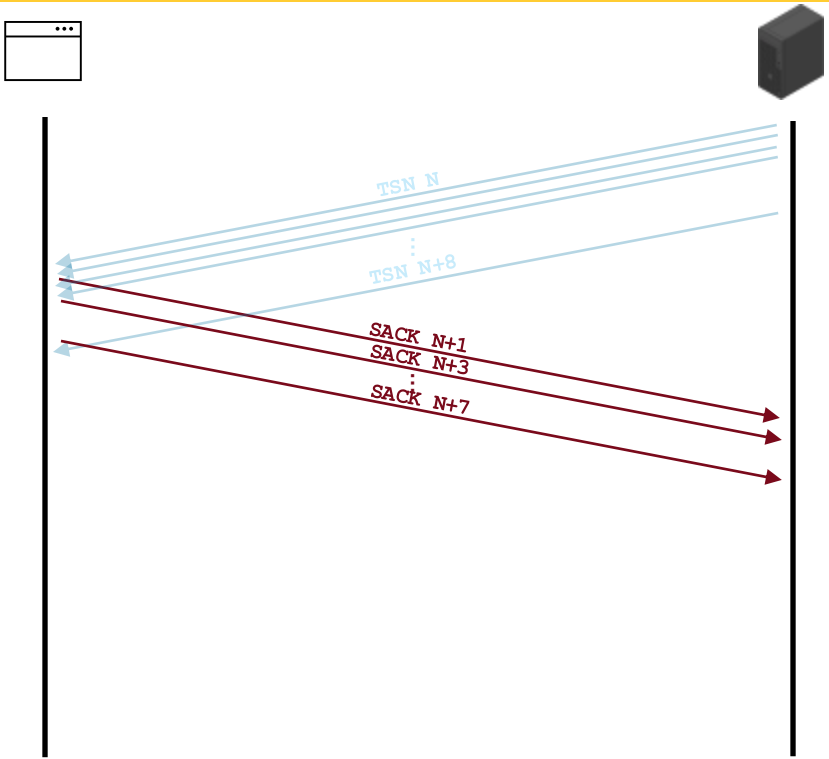
Improvement #2: Before

- Long distance link (RTT ~80ms)
  - `tc qdisc add dev eth0 root netem delay 40ms`
  - `tc qdisc add dev eth1 root netem delay 40ms`
- Result: ~8 Mbit/s in Chrome (~12 – 55 Mbit/s in Firefox)
- TCP comparison: ~90 Mbit/s

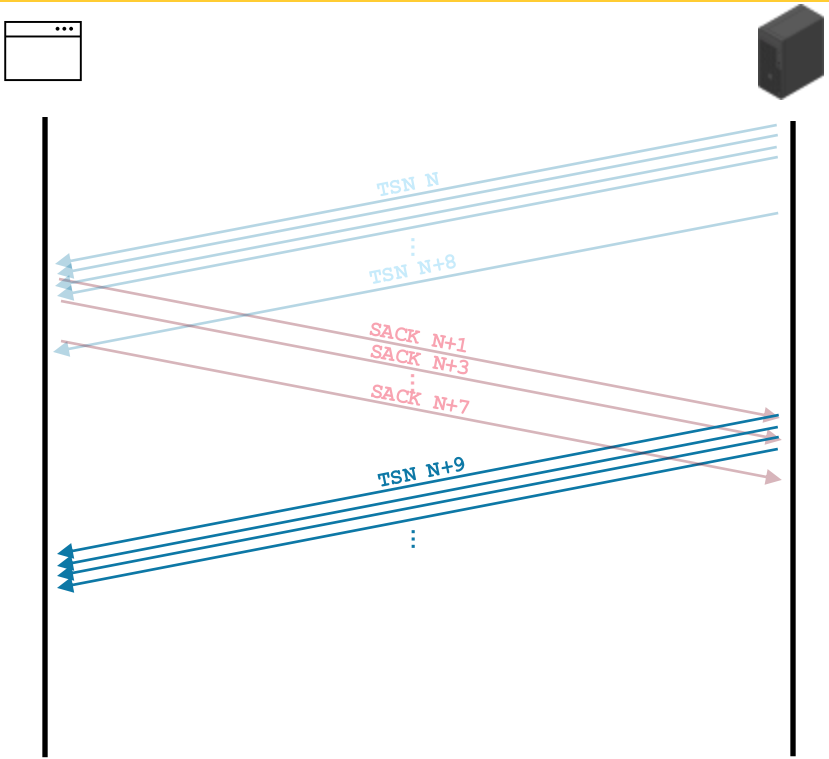
Improvement #2: The problem



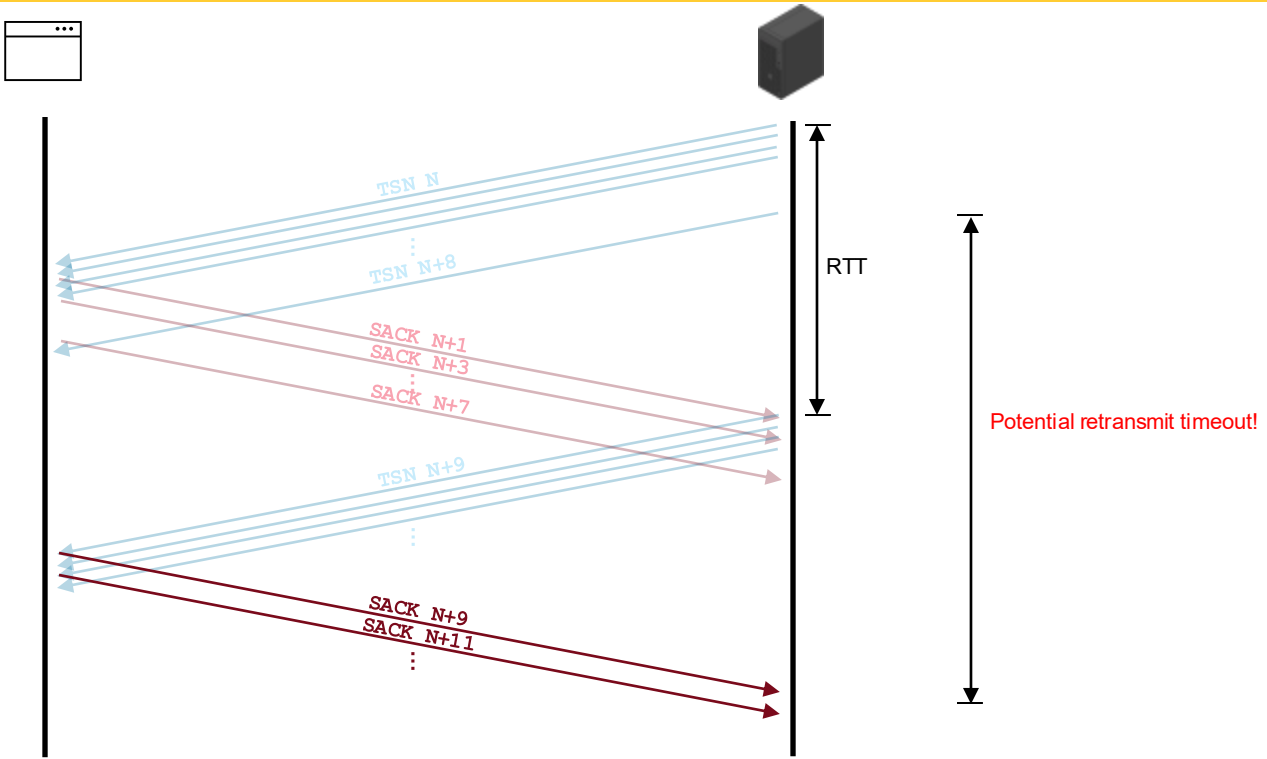
Improvement #2: The problem



Improvement #2: The problem



Improvement #2: The problem



Improvement #2: The solution

- SACK-IMMEDIATELY (RFC-7053)
- Set SACK-IMMEDIATELY flag if: `IN_FLIGHT + MTU >= CWND`
- Similar fix in PION back in 2020 (https://github.com/pion/sctp/issues/62)
- Drawback: A few more SACKs

Improvement #2: Result

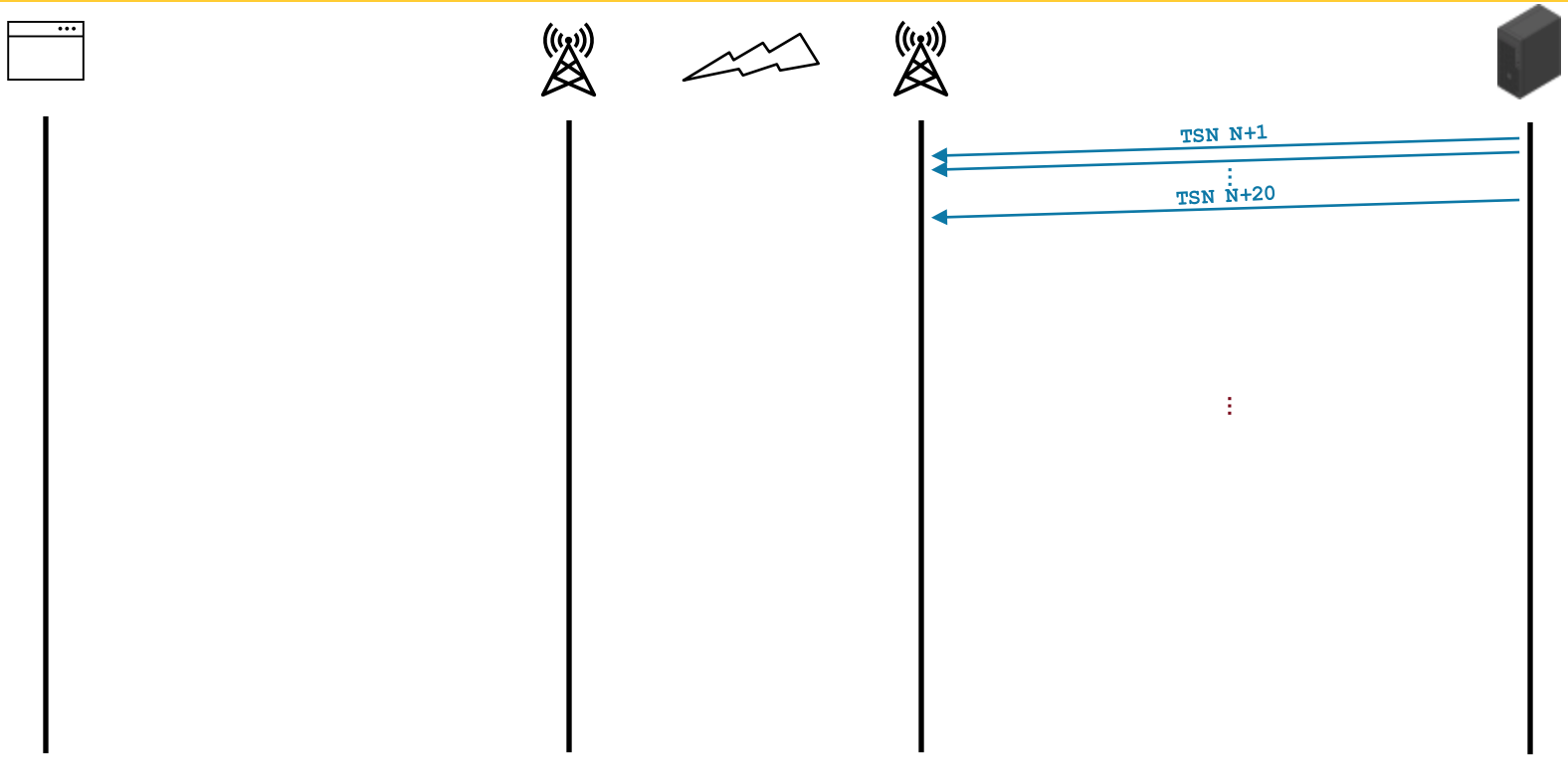
> Chrome: ~15 Mbit/s

> Firefox: stable at ~55 Mbit/s

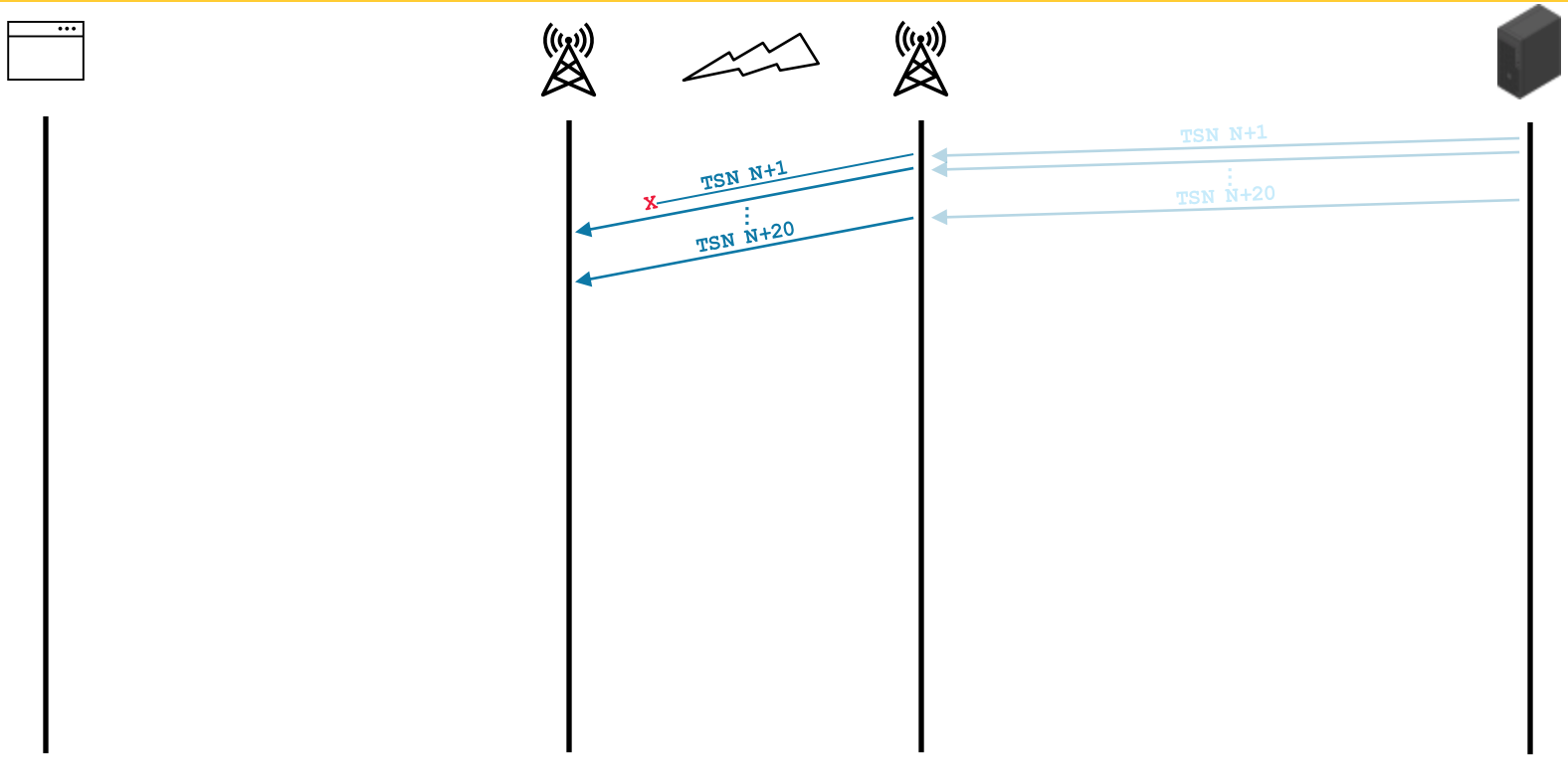
Improvement #3: Environment

- Mobile network (RTT ~80ms, with packet reordering)
  - `tc qdisc add dev eth0 root netem delay 40ms 15ms distribution normal`
  - `tc qdisc add dev eth1 root netem delay 40ms`
- Result: ~600 kbit/s
- TCP comparison: ~1.4 Mbit/s Windows (~13 Mbit/s Linux!!!)

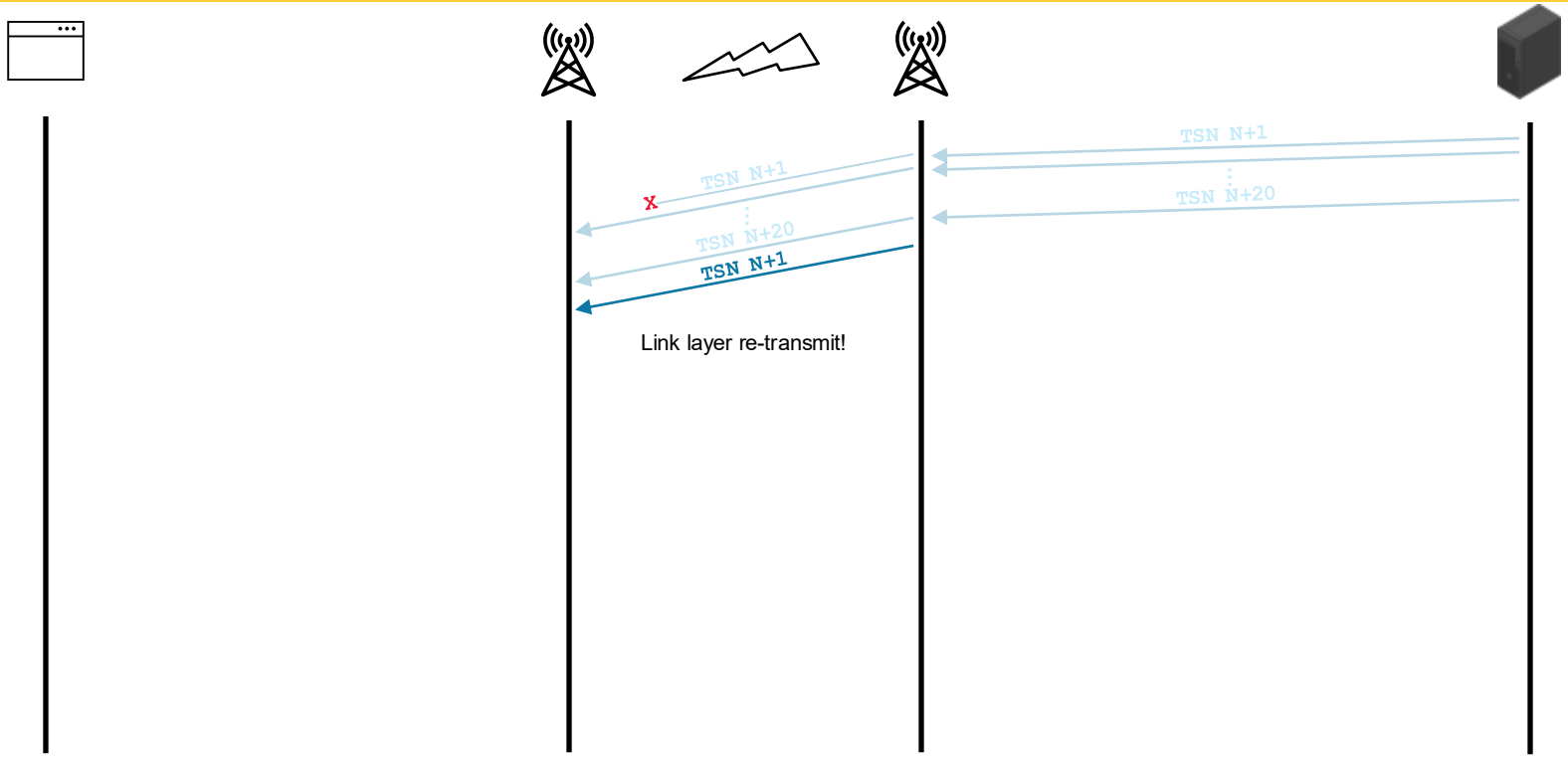
Improvement #3: The problem



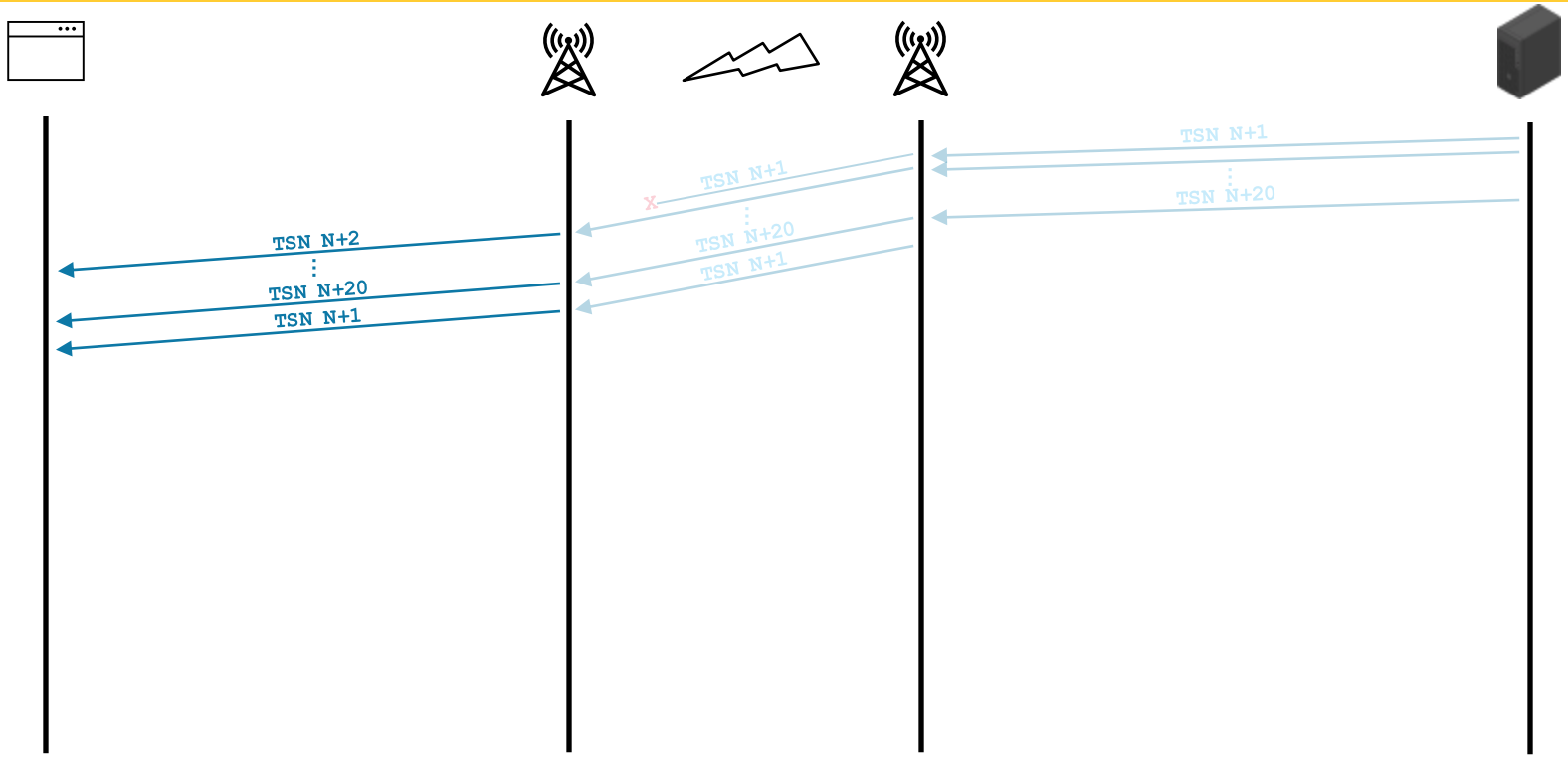
Improvement #3: The problem



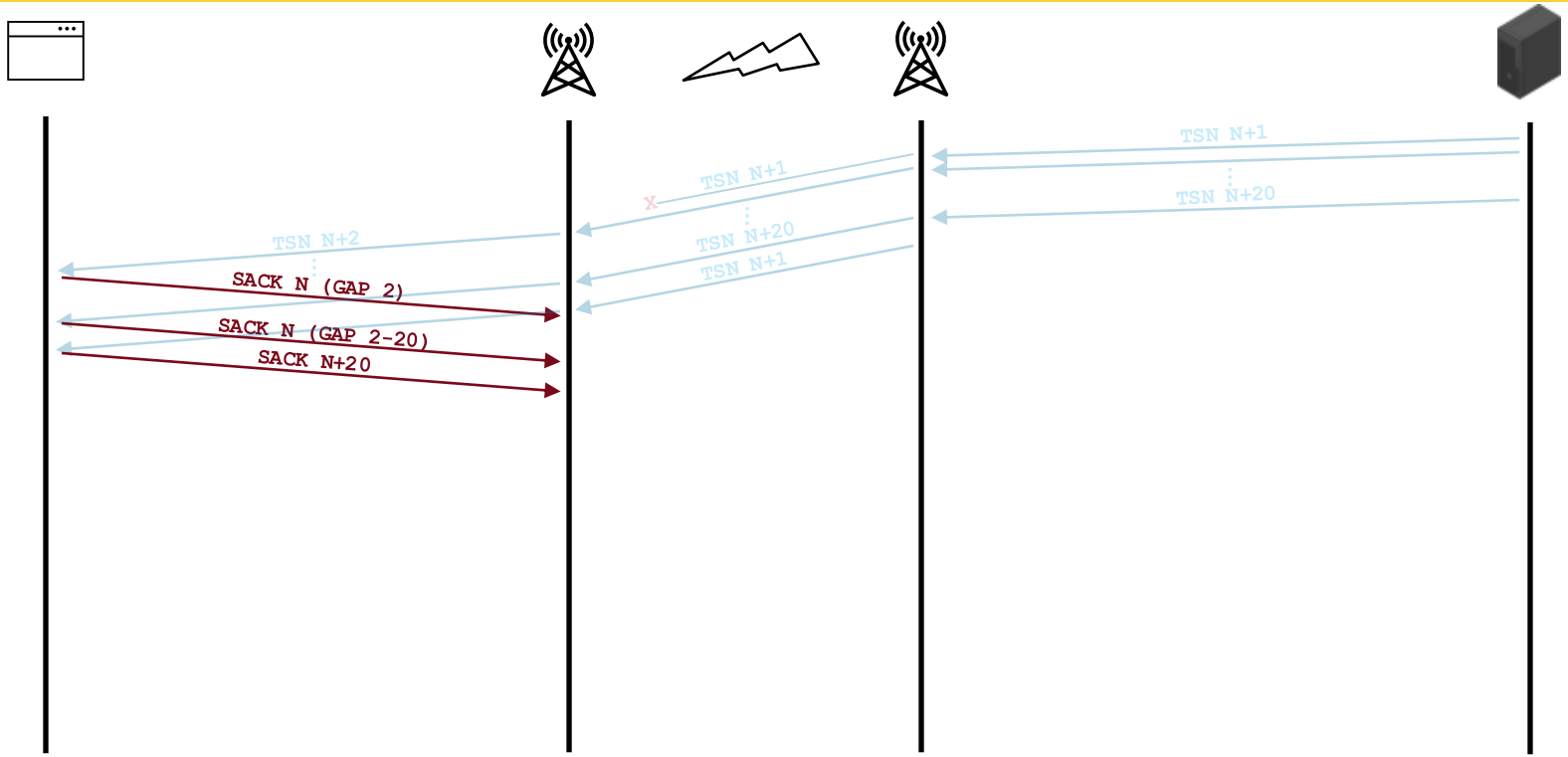
Improvement #3: The problem



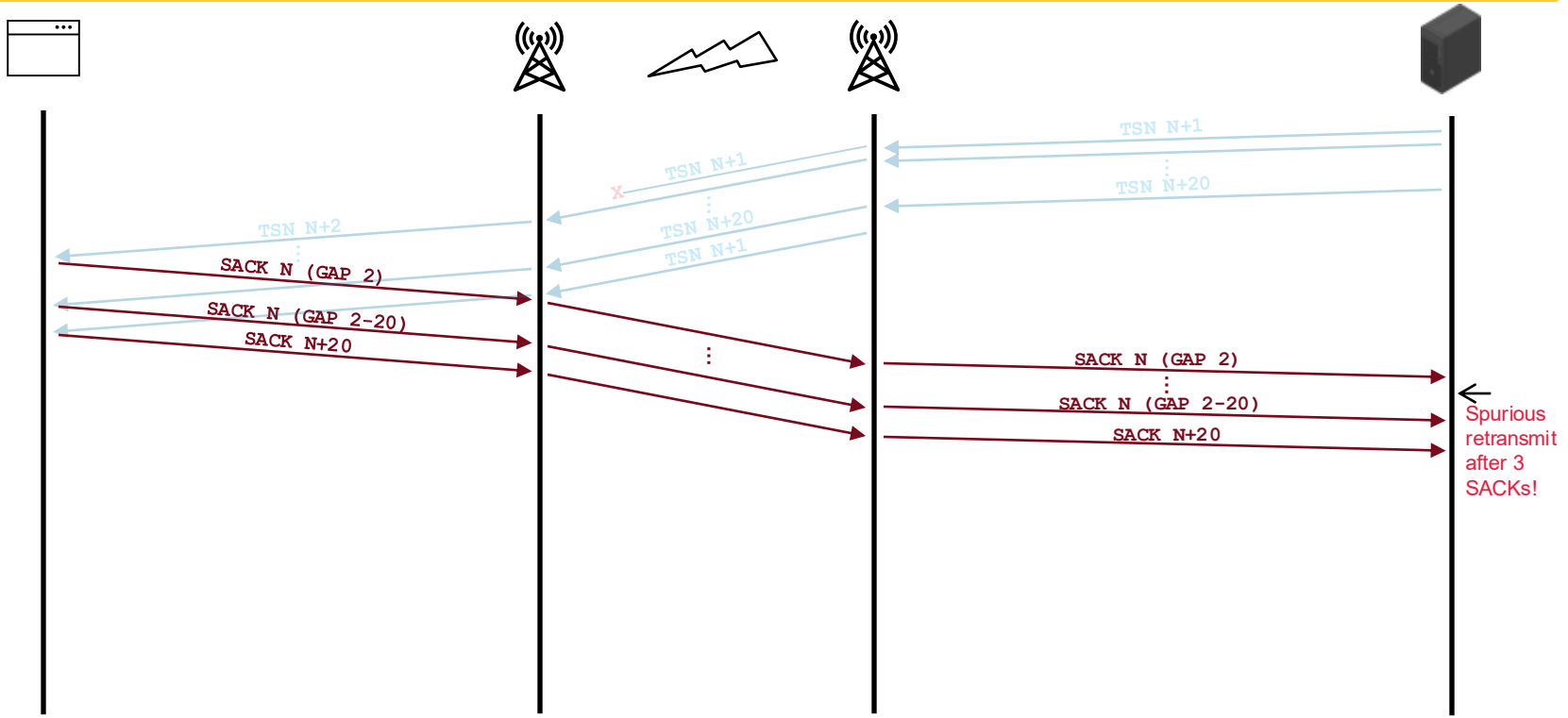
Improvement #3: The problem



Improvement #3: The problem



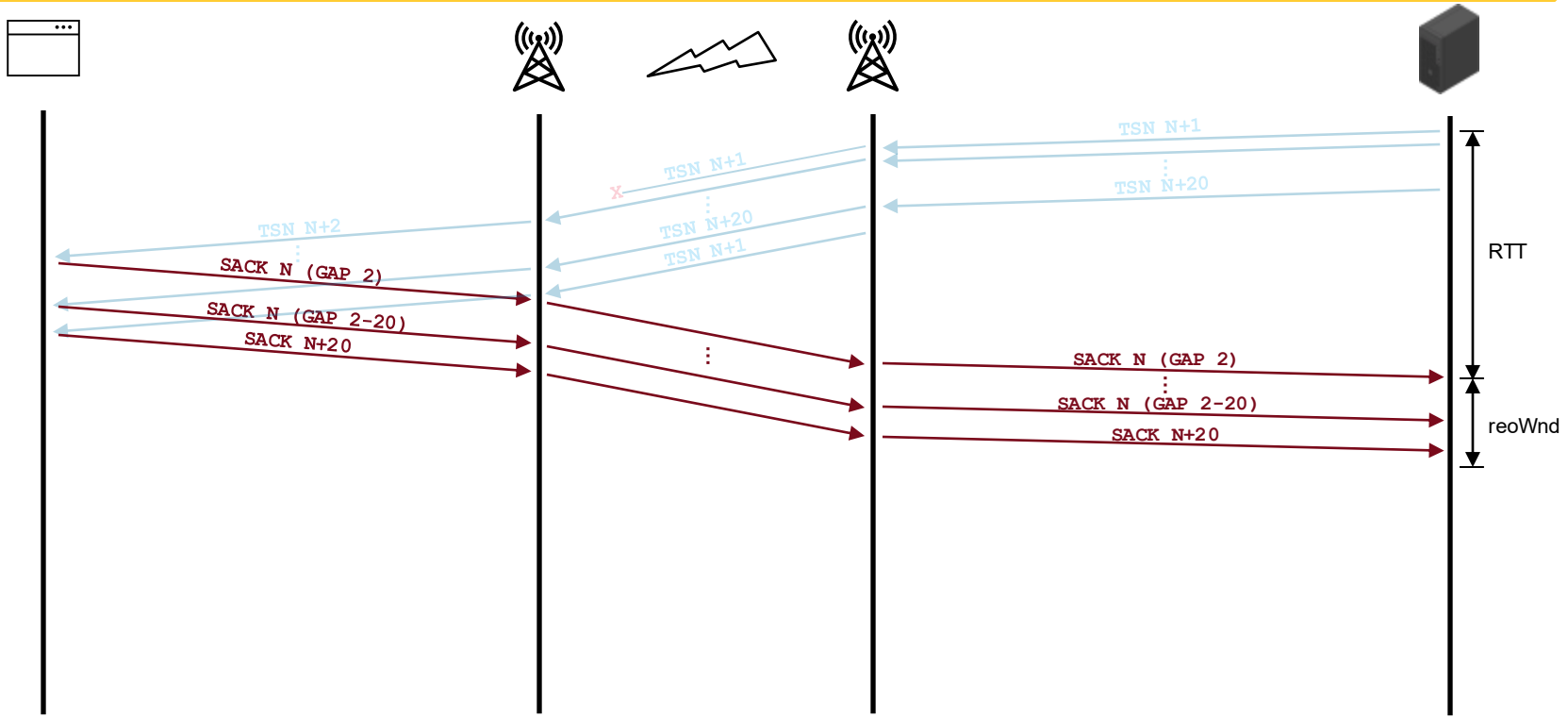
Improvement #3: The problem



Improvement #3: RACK, Possible solution

- RACK for SCTP (Felix Weinrank, Michael Tüxen, 2020)
  - Based on The RACK-TLP Loss Detection Algorithm for TCP (RFC 8985)
  - Uses a ”reordering window” instead of “DupACK count”
  - No retransmit until RTT + reoWnd

Improvement #3: RACK, Possible solution



Improvement #3: RACK, Possible solution

- Implementation in usrSCTP in progress
- Preliminary results: looking good so far
  - ~6 Mbit/s (before: ~600 kbit/s)
- Important to avoid decreased performance in other network setups
- Upstream to usrSCTP as well

Future

- Upstream improvement #1 + #2 ASAP!
- Finalize RACK implementation
- dcSCTP
  - YES! But probably no silver bullet in terms of performance
- What about TURN TCP / TURNS?
  - 2 layered congestion algorithms, is that a problem?

Questions?

Thank you!