TCP-aNCR

draft-zimmermann-tcpm-reordering-detection-00
draft-zimmermann-tcpm-reordering-reaction-00

Alexander Zimmermann <alexander.zimmermann@netapp.com>
Lennart Schulte <lennart.schulte@aalto.fi>

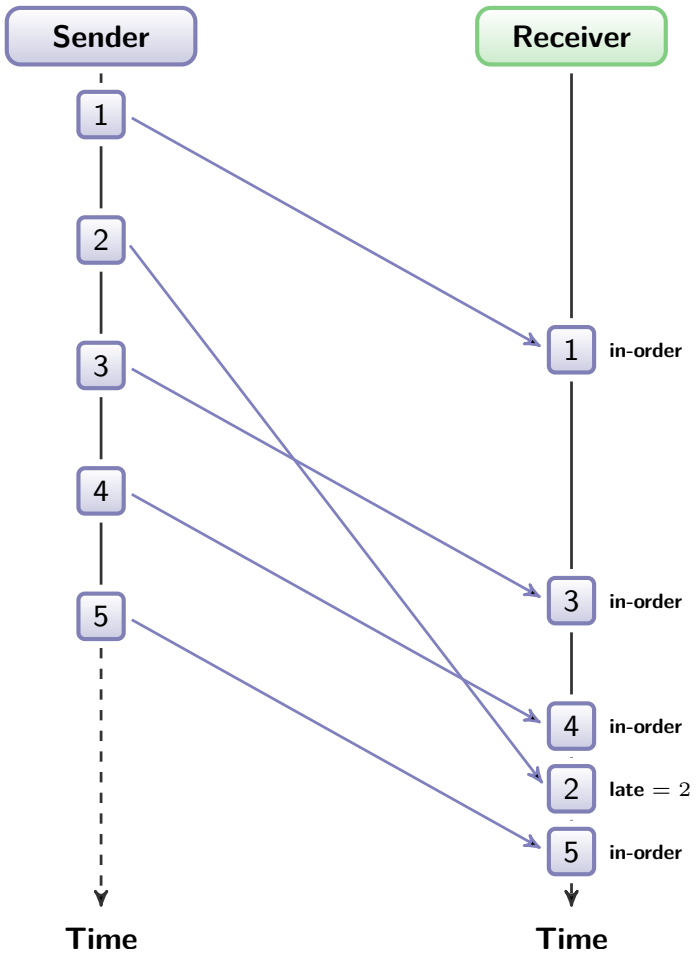
Motivation

- **Reordering detection draft**
  - Standardize Linux reordering behavior
    → detection with SACK / DSACK / Timestamps
  - Extent Linux reordering behavior (SACK scoreboard)
    → use reordering events below `SND.UNA`

- **Reordering reaction draft**
  - Maintain RFC 4653 (TCP-NCR): Bug fixing
    → premature leaving Slow-Start, burst protection could fail, performance issues if reordering is persistent
  - Making TCP-NCR adaptively robust to non-congestion events → “must have” for Linux Kernel integration

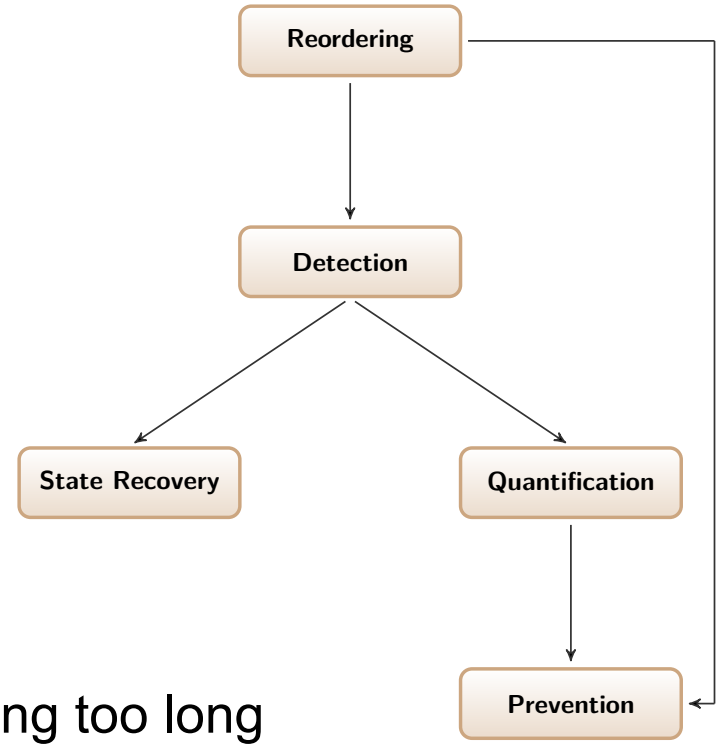
RFC 4737: Reordering Definition & Metric

- **Definition of reordering**
  - *Late arrival:* packet is received later than another with higher sequence number
- **Metric for quantifying**
  - *Reordering extent:* specifies how "late" a reordered packet is received in terms of packets



Approaches to Tackle Reordering

- **Detection & Quantification**
  - SACK / DSACK
  - TCP Timestamps option

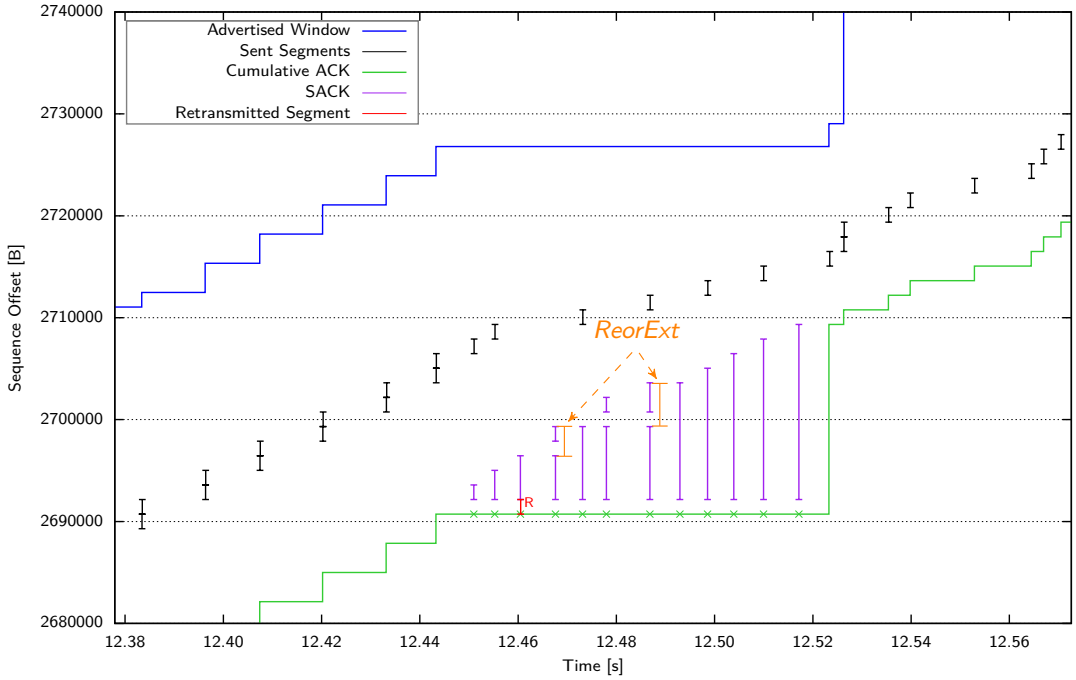


- **Reaction & Prevention**
  - *State recovery:* undoing CWND reduction
  - *Prevention strategies:* delay Fast Retransmit
    - Avoid unnecessary RTOs by waiting too long
    - Transmission of new data during Disorder

Detection with SACK (1/2)

- **Reordering detection**
  - Closed hole in SACK scoreboard (*SACK* closes hole)
  - For transmissions only → retransmission ambiguity problem

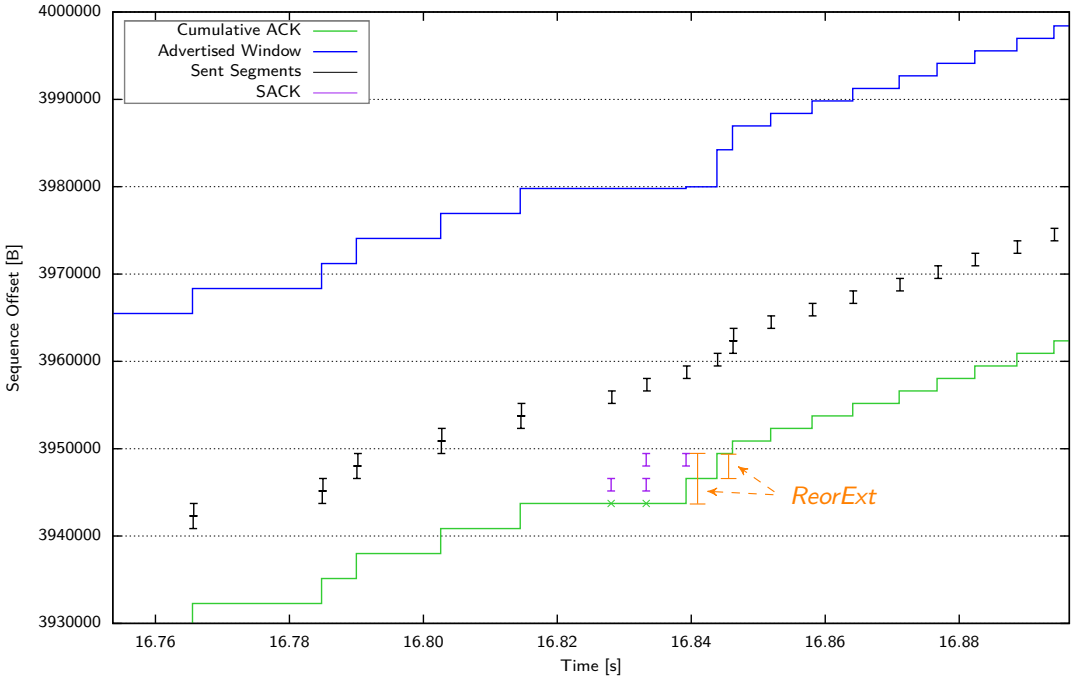
- **Quantification**
  - `ReorExt = SND.FACK - SEG.SEQ`



Detection with SACK (2/2)

- **Reordering detection**
  - Closed hole in SACK scoreboard (*ACK* closes hole)
  - For transmissions only → retransmission ambiguity problem

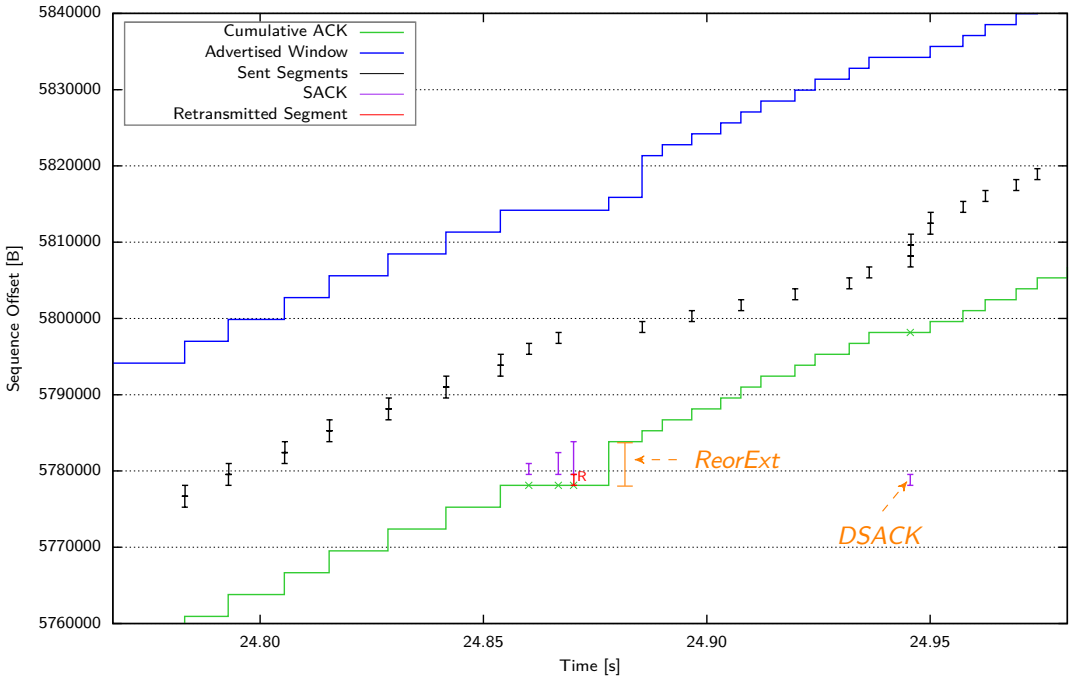
- **Quantification**
  - `ReorExt = SND.FACK - SEG.SEQ`



Detection with DSACK (1/2)

- **Reordering detection**
  - Using DSACK to detect spurious retransmissions (DSACK *below* `SND.UNA`)
  - RFC 3708

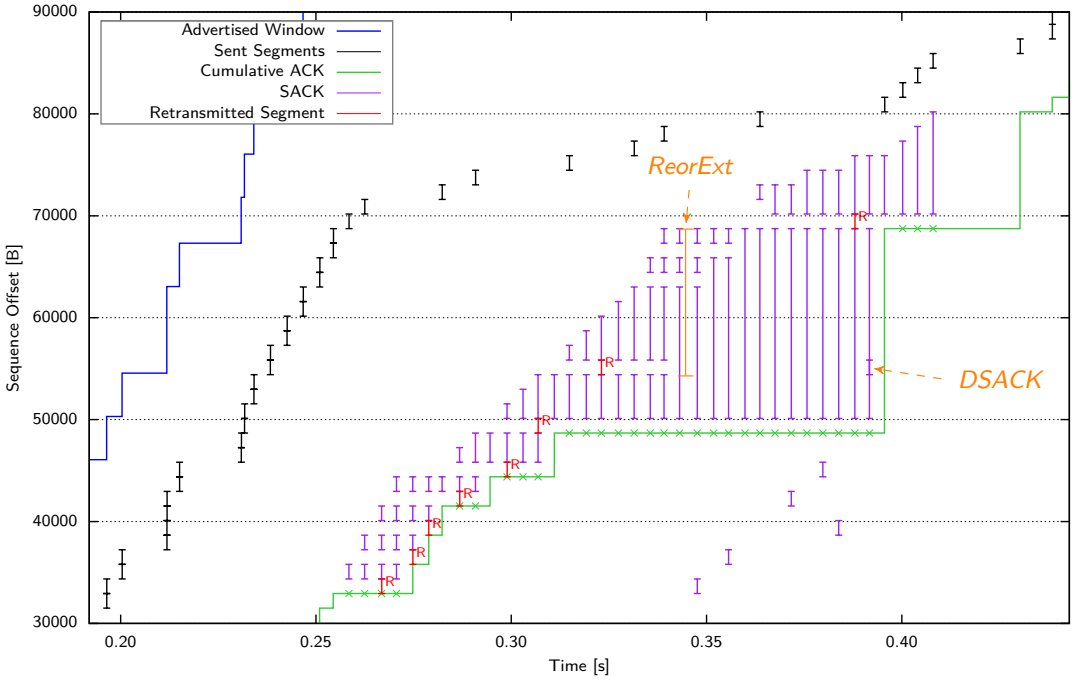
- **Quantification**
  - Equal to SACK
  - Reordering extent has to be stored



Detection with DSACK (2/2)

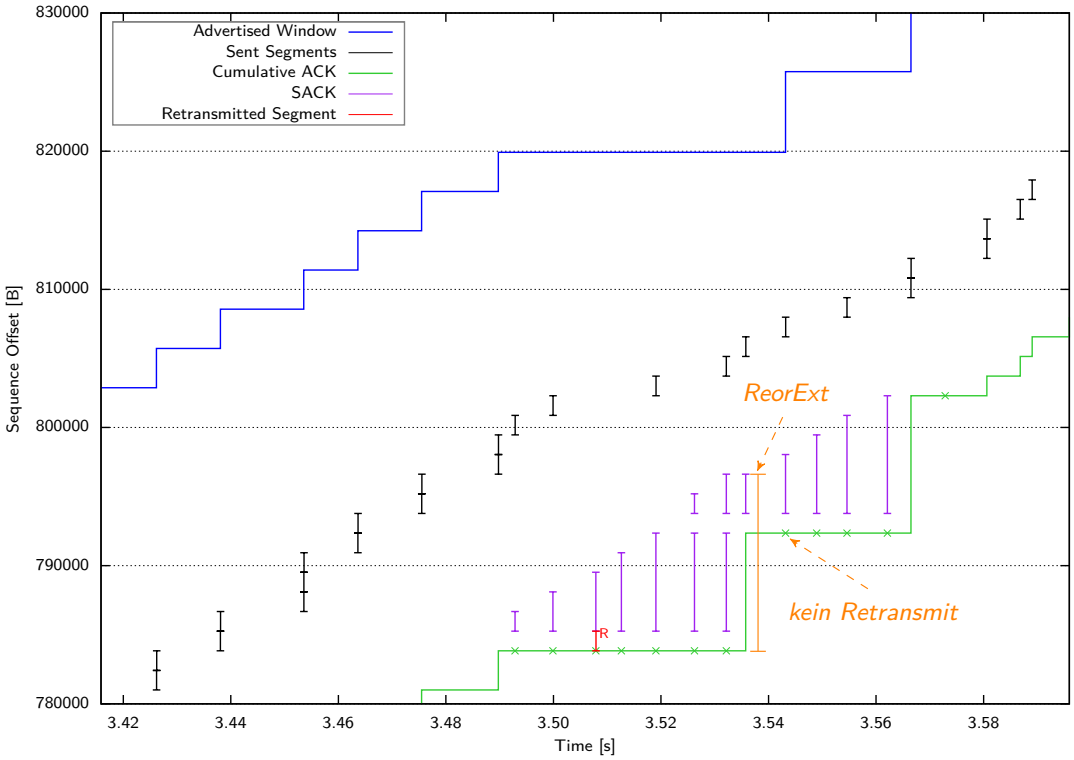
- **Reordering detection**
  - Using DSACK to detect spurious retransmissions (DSACK *above* `SND.UNA`)
  - RFC 3708

- **Quantification**
  - Equal to SACK
  - Reordering extent has to be stored



Detection with TCP Timestamps Option

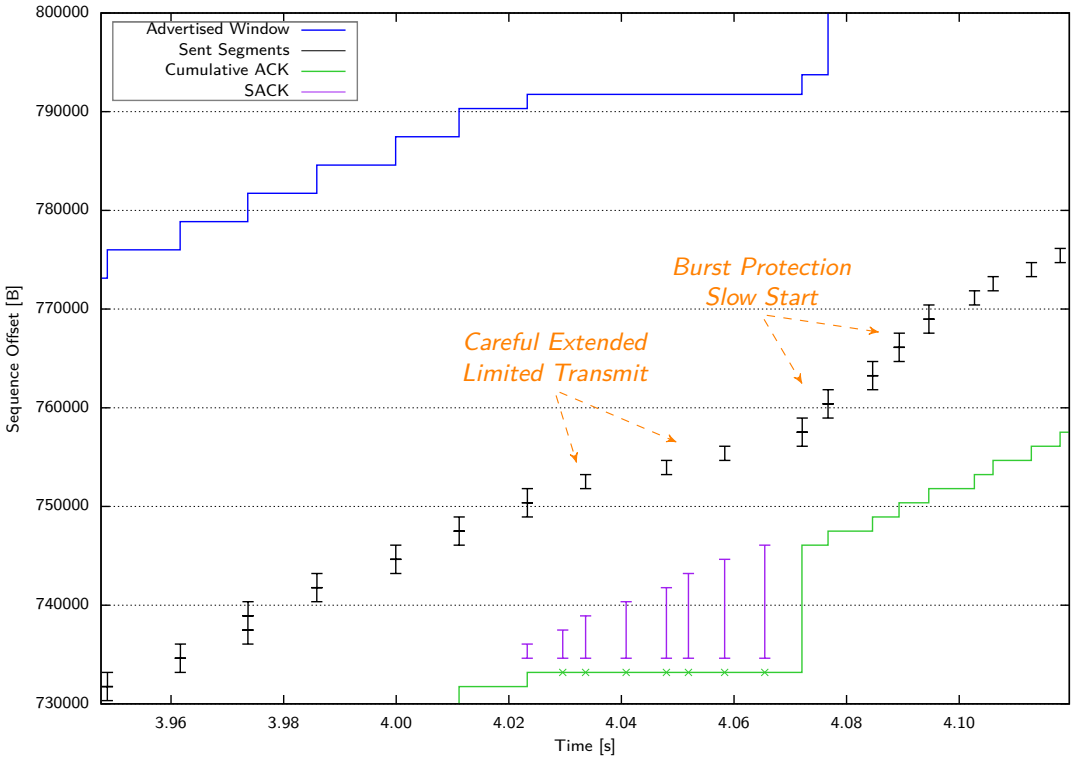
- **Reordering detection**
  - Using TS to detect spurious retransmissions
  - RFC 3522 (Eifel Detection)
- **Quantification**
  - Equal to SACK
  - Reordering extent has *not* to be stored



Why using TCP-NCR?

- **Basic idea**
  - Set `DUPTRESH` to `1*CWND` (≈ `1*RTT`)
  - Aggressive vs. Careful Limited Transmit

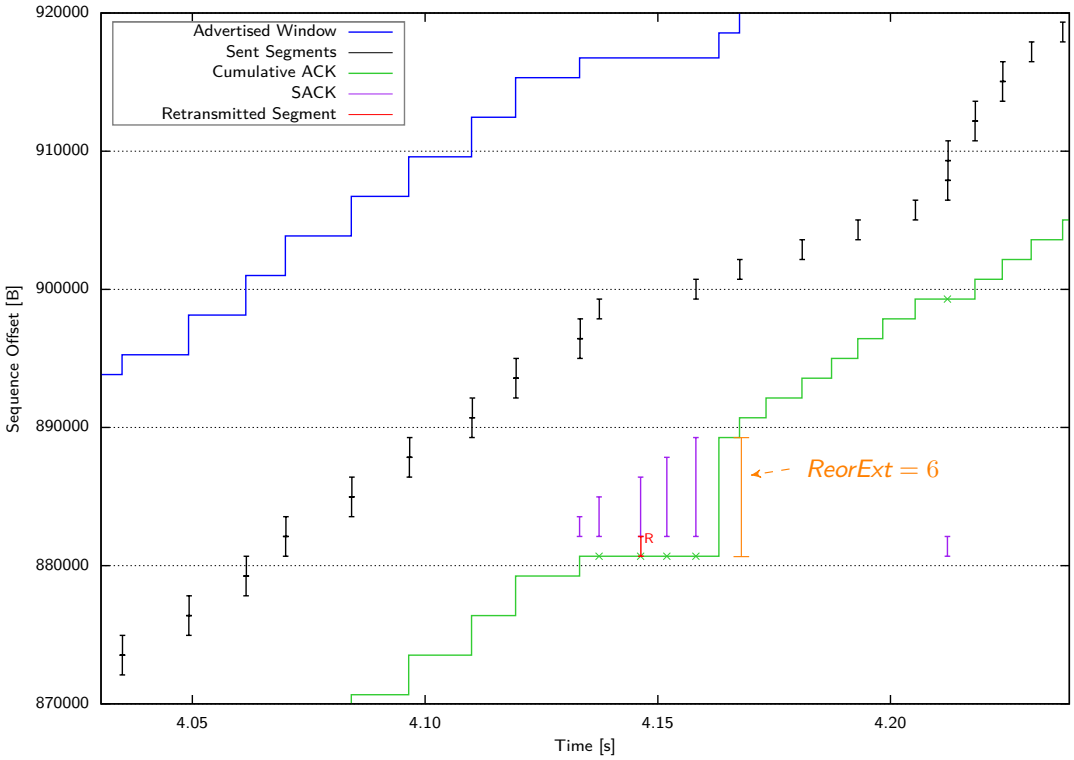
- **Strengths**
  - Avoid RTOs
  - Adapt `DUPTHRESH` on each SACK
  - Burst protection



Detection & Quantification in TCP-aNCR (1/2)

- **Status quo**
  - *Detection:* SACK, DSACK and TCP Timestamps option
  - *Quantification:* Reordering extent

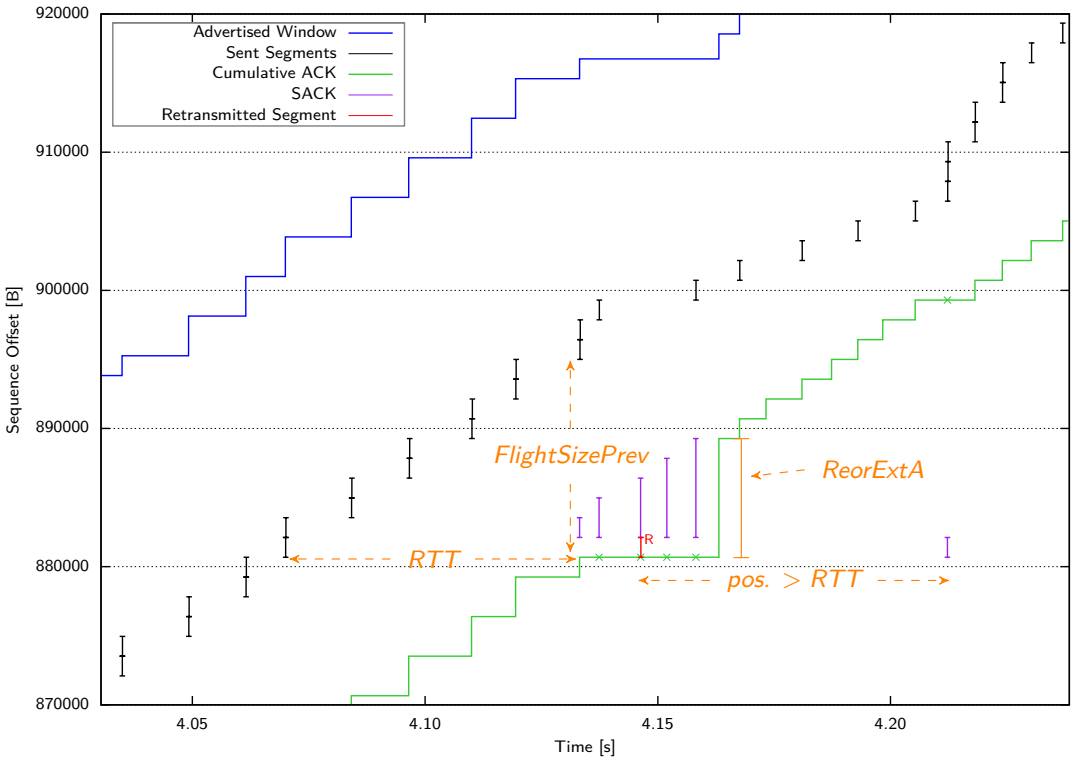
- **Disadvantage**
  - Reordering extent depends on TCP's sending rate



Detection & Quantification in TCP-aNCR (2/2)

- **TCP-aNCR's solution**
  - Store `FlightSize` when entering Disorder state
  - Relative reordering extent: `ReordExt/FightSizePrev`
  - Independent of TCP's sending rate

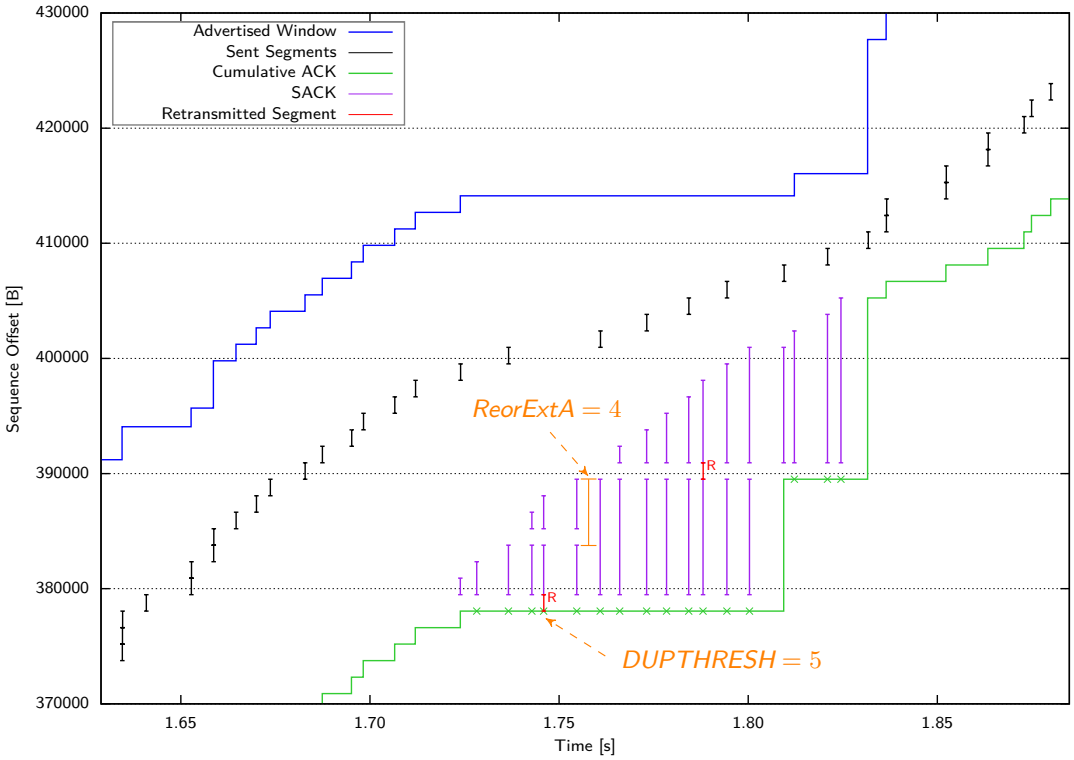
- **Challenge**
  - `FightSizePrev` is never updated if Disorder state is not left → Update it once per RTT



Reaction & Prevention in TCP-aNCR

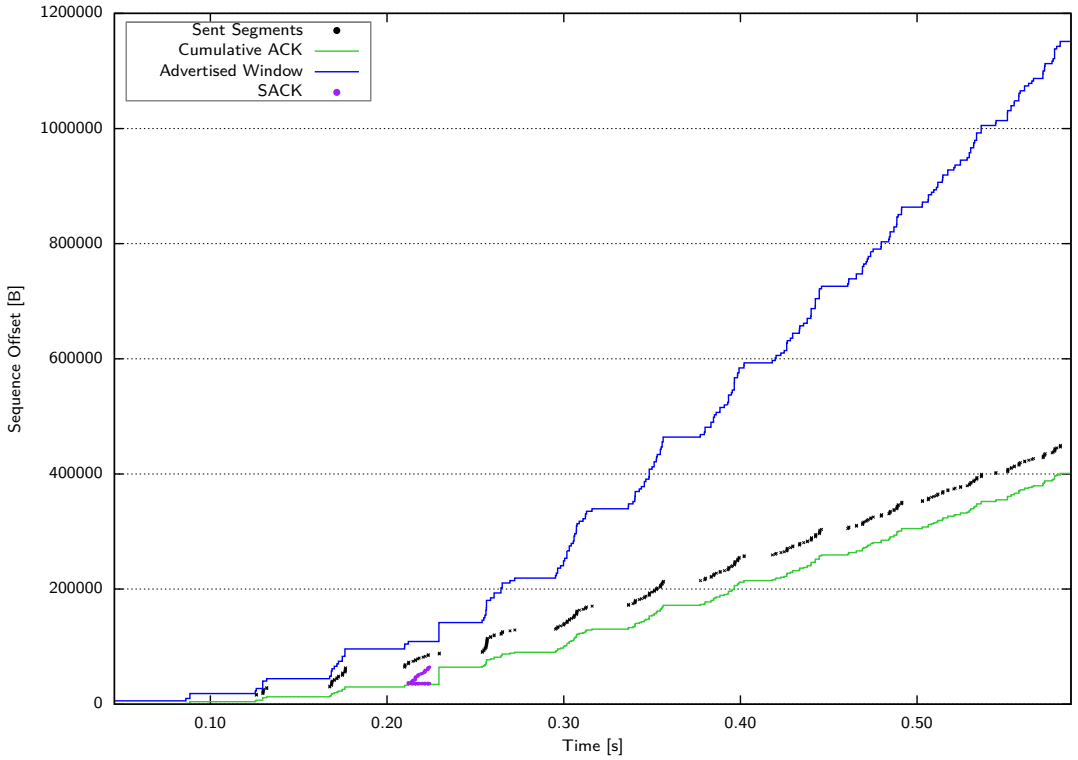
- **DUPTRESH formula**
  - $DUPTHRESH_{aNCR} = min(FlightSizePrev * ReorExtR_{max}, DUPTHRESH_{NCR})$

- **DUPTRESH strategy**
  - TCP-NCR avoids unnecessary RTOs → upper bound
  - Only delay as needed → Max. relative reordering extent
  - Resetting on RTO



Packet Reordering during Slow Start (1/2)

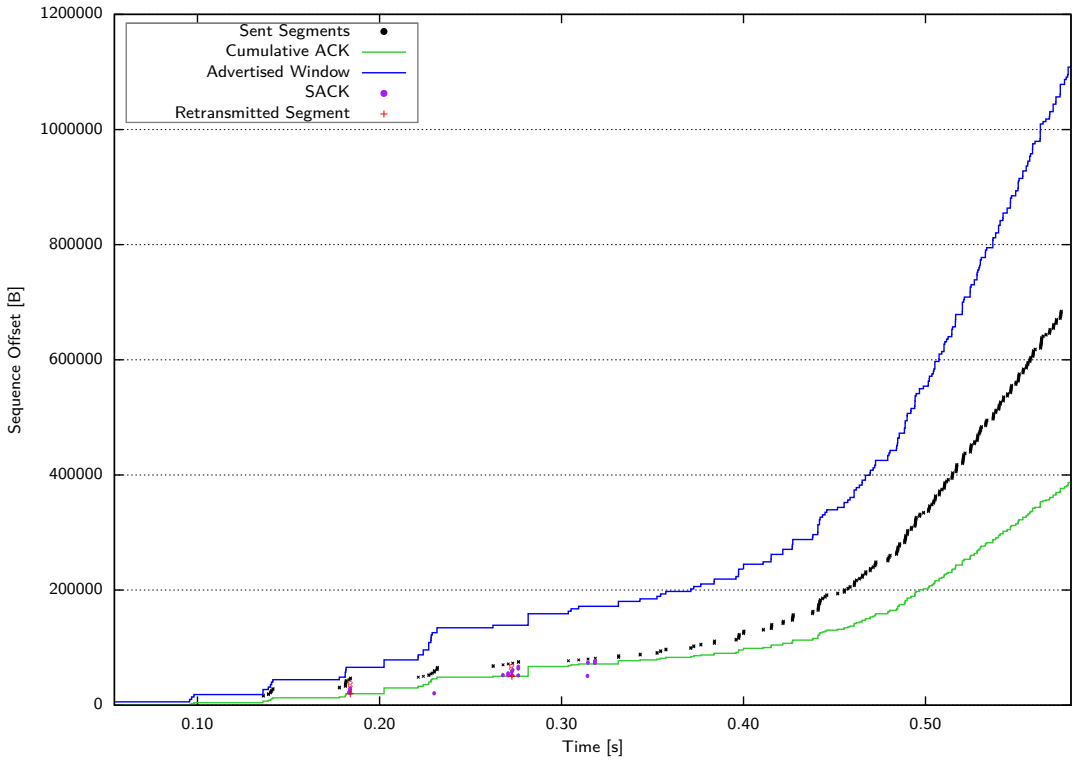
- **Problem of RFC 4653**
  - Slow start can be interrupted by mild reordering



Packet Reordering during Slow Start (2/2)

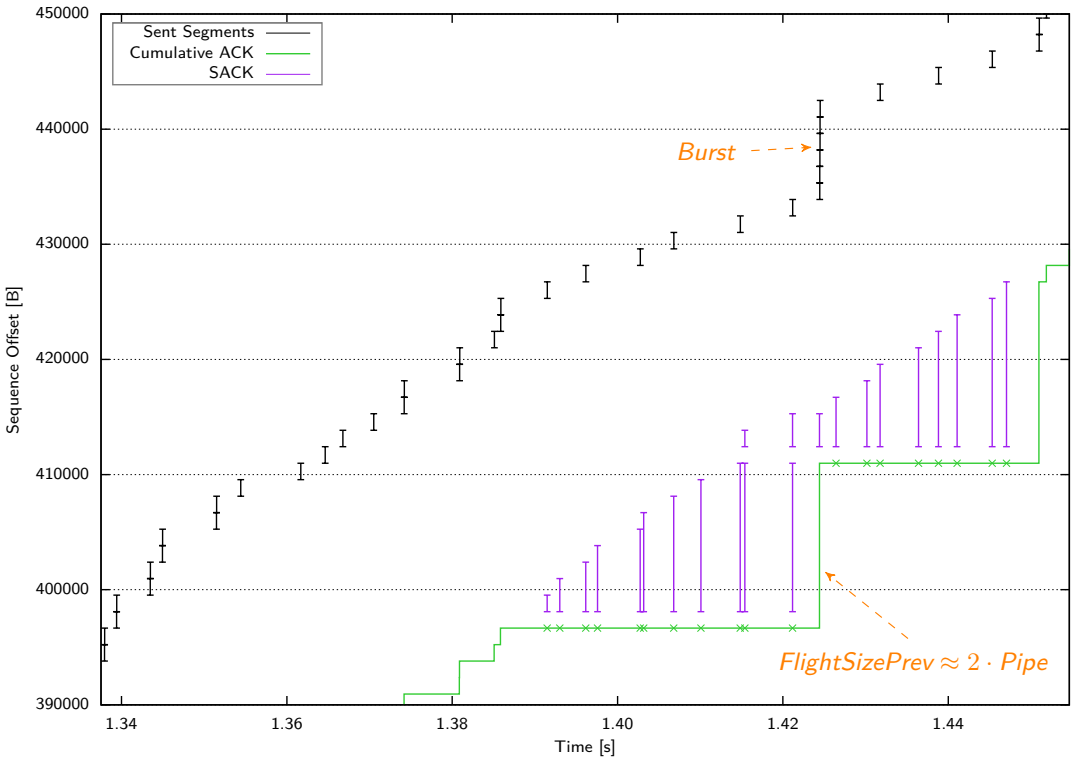
- **Problem of RFC 4653**
  - Slow start can be interrupted by mild reordering

- **TCP-aNCR solution**
  - Let slow start continue after the reordering event



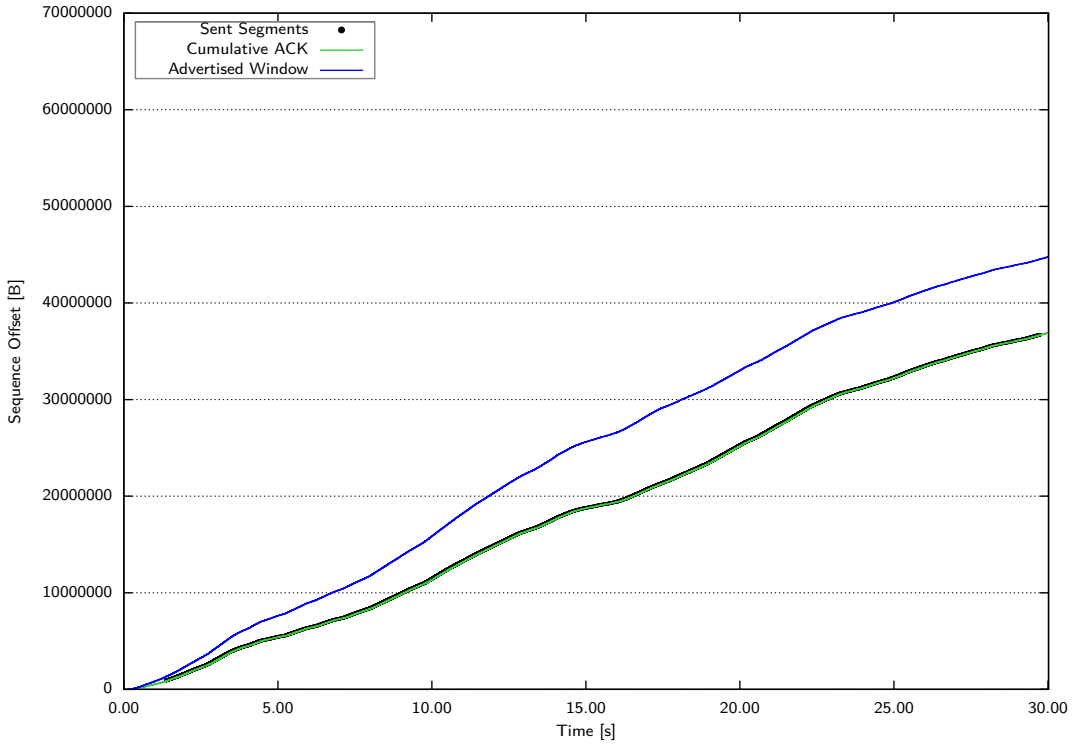
Avoiding of Data Bursts

- **Problem of RFC 4653**
  - Burst protection fails
    - Reordering or loss on reverse path
    - Re-start Extended Limited Transmit
- **TCP-aNCR solution**
  - Burst variable
  - Re-start vs. termination of ELT



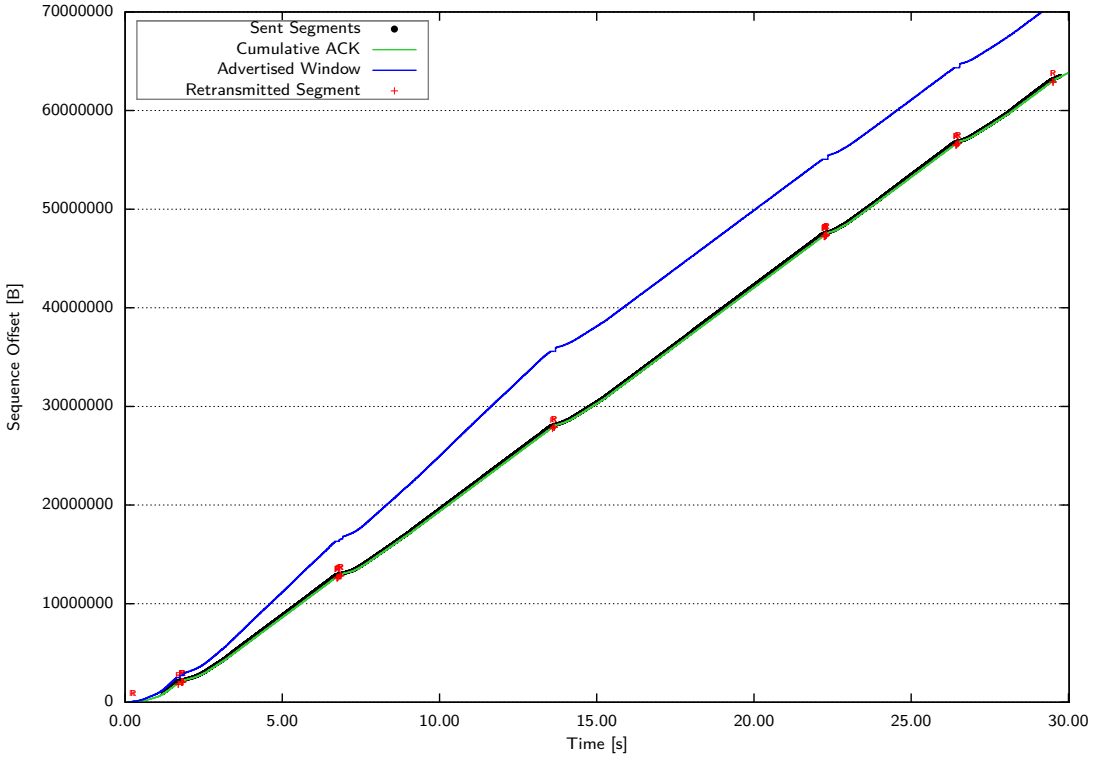
Persistent SACK Arrival (1/2)

- **Problem of RFC 4653**
  - Transmission of data depends on `FlightSizePrev`
  - `FlightSizePrev` is never updated



Persistent SACK Arrival (2/2)

- **Problem of RFC 4653**
  - Transmission of data depends on `FlightSizePrev`
  - `FlightSizePrev` is never updated
- **TCP-aNCR solution**
  - Apply CWND
  - Update whenever `Recover` is covered

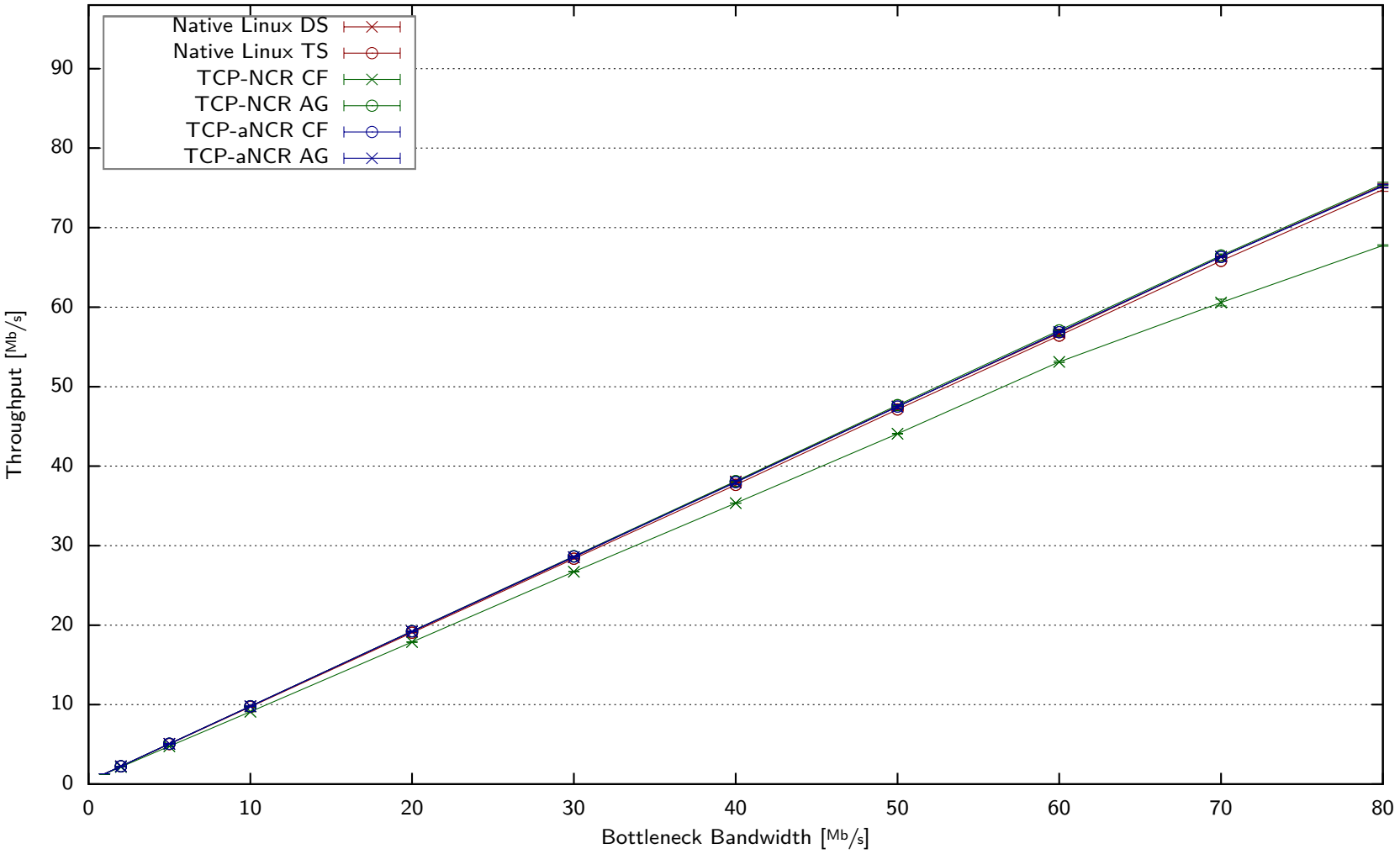


Evaluation Methodology

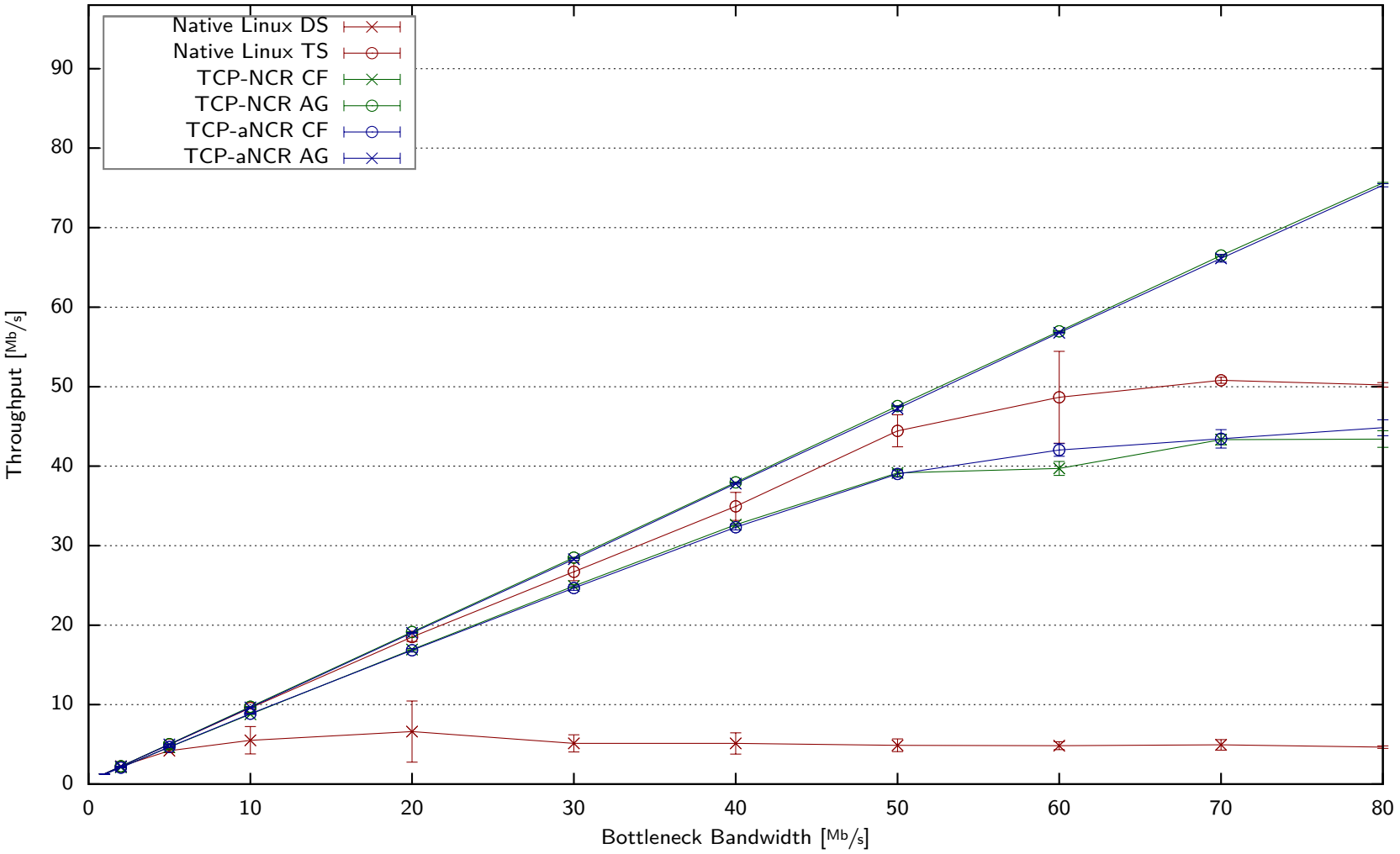
- **Traffic model**
  - One unidirectional long-lived bulk TCP flow
  - No competing flows; no cross traffic, no short-lived flows
  - Traffic generation: flowgrind
  - Network emulation: netem

- **Flows under test**
  - Linux 2.6.x (w/(o) Timestamps); TCP-(a)NCR (CF & AG)
  - Duration 120s; Repetition: 10x
  - Bottleneck Bandwidth 20 Mbps, RTT 40ms
  - Reordering Rate 2%, Reordering Delay 20ms

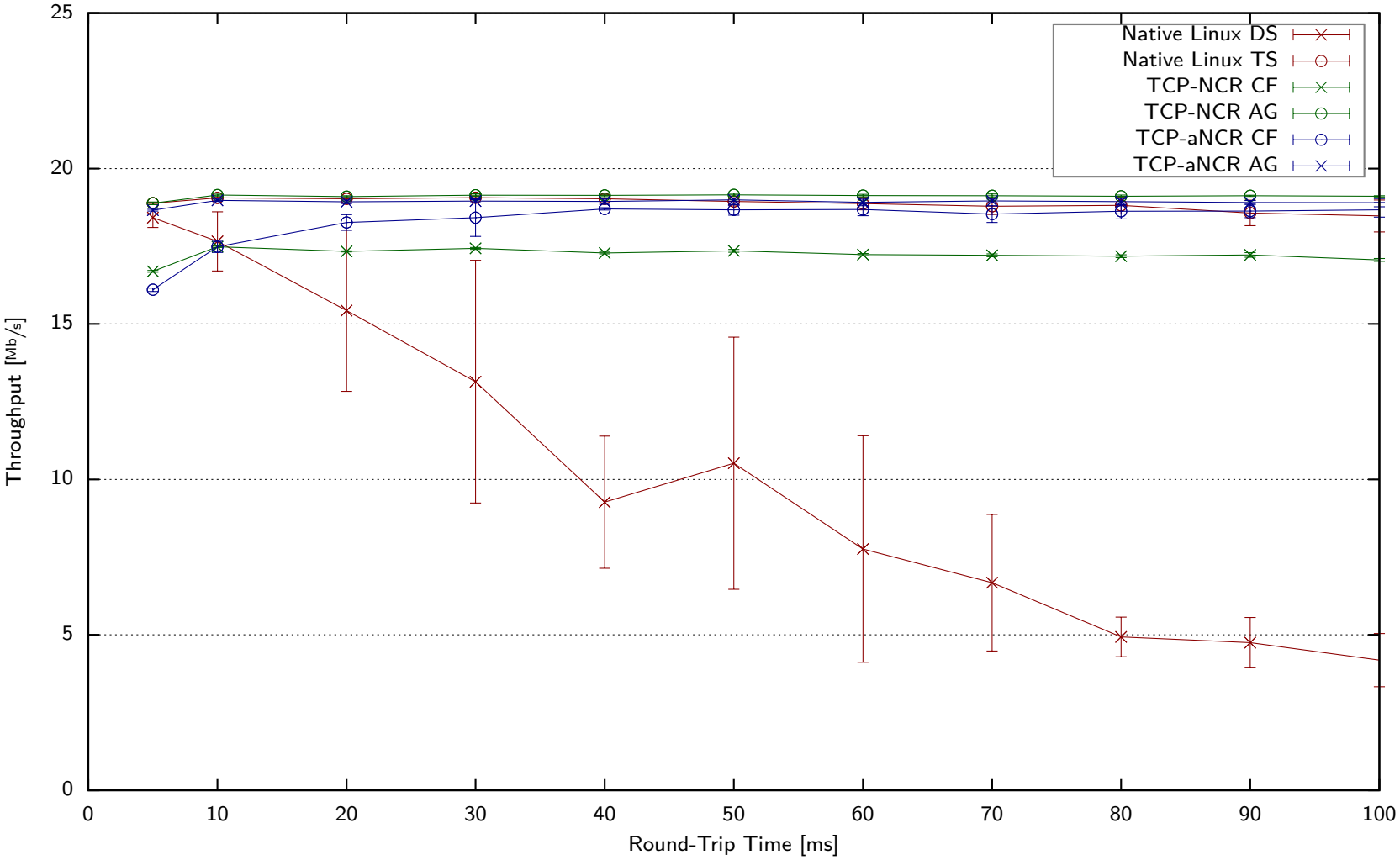
Bandwidth Variation – No Reordering



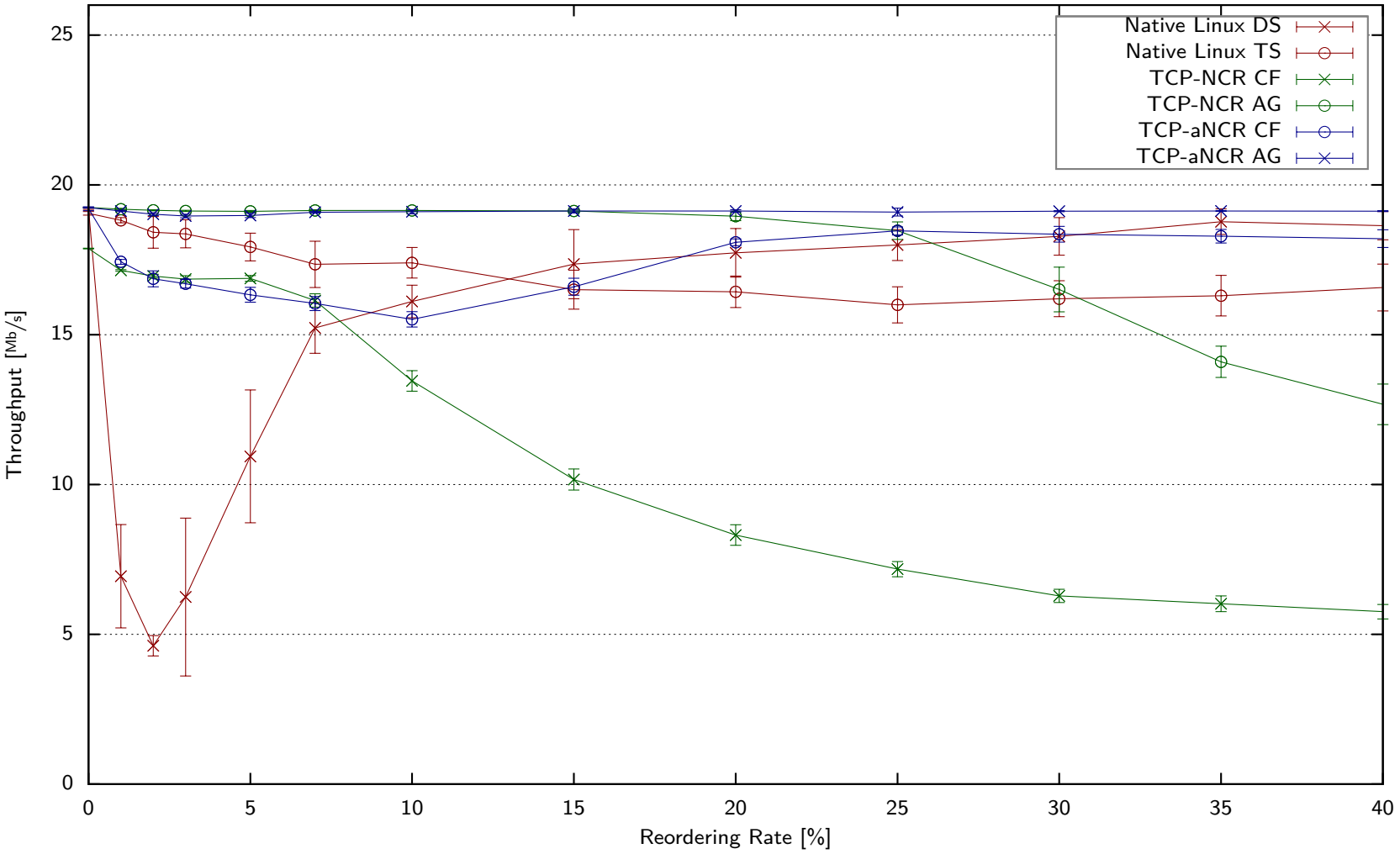
Bandwidth Variation – Reordering



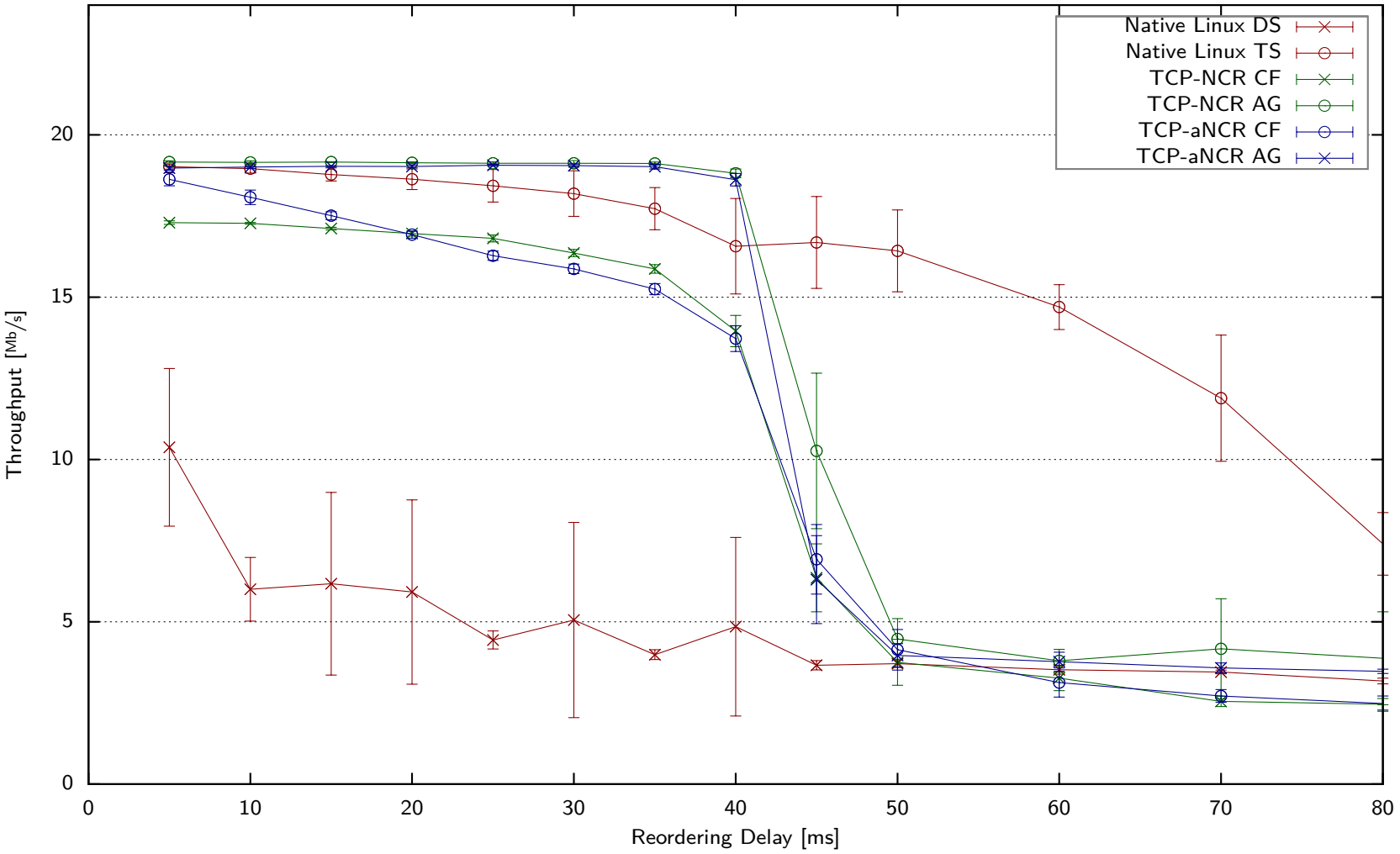
Round-Trip Time Variation



Reordering Rate Variation



Reordering Delay Variation



Summary

- **Conclusion**
  - Standards compatible sender side only modification
  - Utilizes knowledge about both history and current state
  - No change of standard behavior without reordering
  - Good compromise between reordering robustness (throughput), latency and receiver buffer load
  - Performs better than Linux and (Careful) TCP-NCR

- **Next steps**
  - Getting reviews
  - Update code base → Linux 3.15
  - More measurements