TCP Adaptation for MPI on Long-and-Fat Networks

Motohiko Matsuda, Tomohiro Kudoh
Yuetsu Kodama, Ryousei Takano
Grid Technology Research Center
National Institute of Advanced Industrial Science and Technology

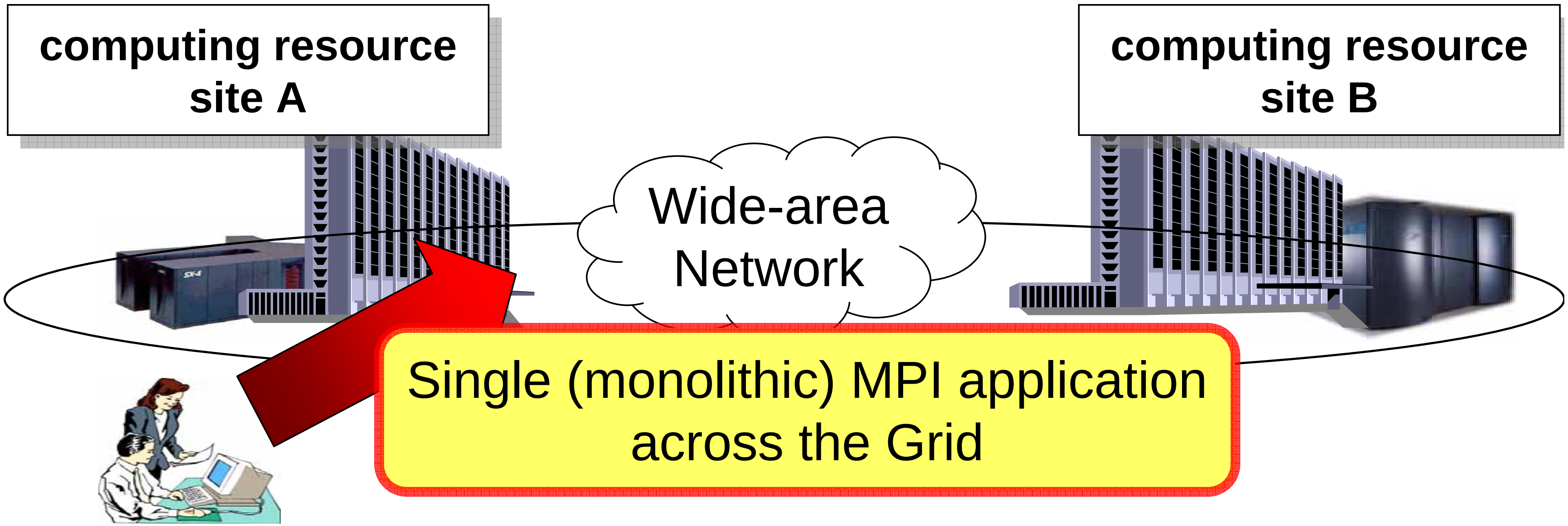
Yutaka Ishikawa
The University of Tokyo

Outline

- Background
- Review of TCP Basics
- TCP Behavior versus MPI Traffic
- Modifications to TCP and Implementation
- Evaluation; Simple Traffic and NPB Benchmarks
- Related Work
- Conclusion

Background

- Demands on high-performance MPI using TCP/IP
  - Commodity clusters, computing in the Grid, ...
- GridMPI Project
  - Run unmodified HPC applications in the Grid
  - Focus on metropolitan-area, high-bandwidth environment: ≥10Gpbs, ≤ 500miles (10ms one-way)

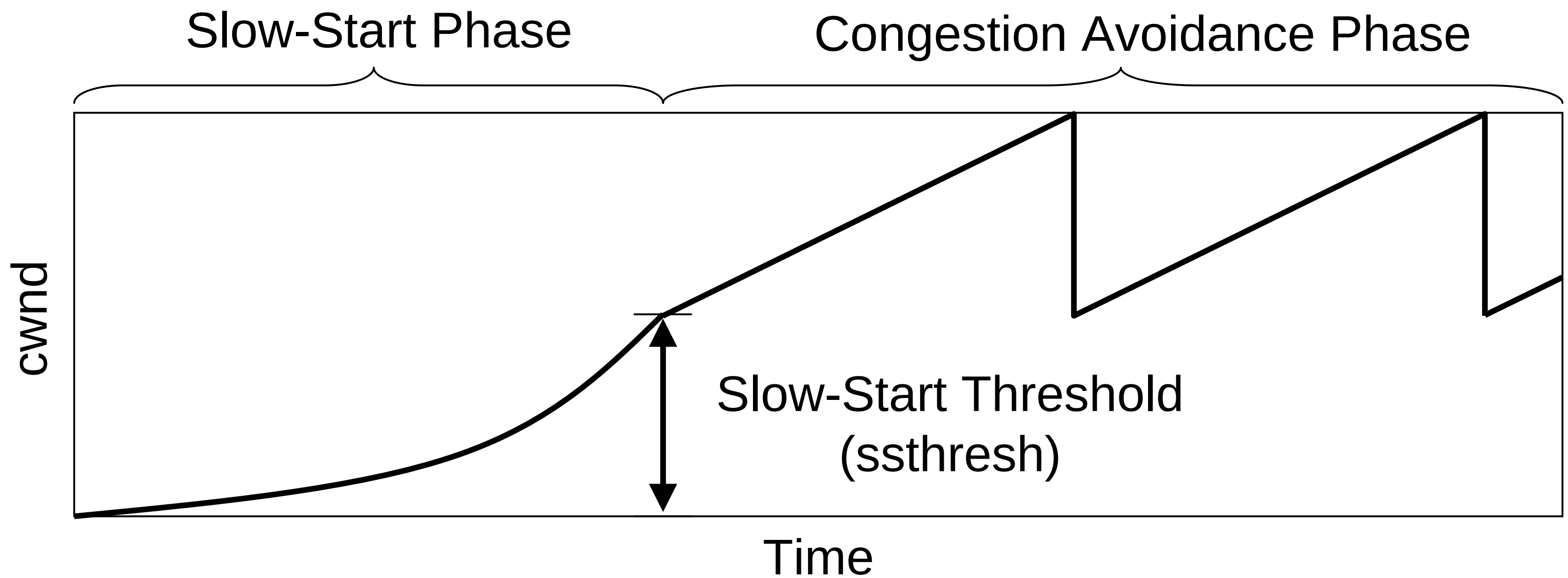


Background (cont.)

- MPI traffic is sub-optimal for TCP
  - TCP is designed for streams
- Typical MPI applications:
  - Repeat computation/communication phases
    - No communication during computation phases
  - Change traffic by communication patterns
    - e.g., one-to-one to all-to-all, and vice versa
- Known, old issue, but no fixing in actual implementations
  - Fix TCP behavior to non-contiguous traffic
  - Show impact in MPI applications

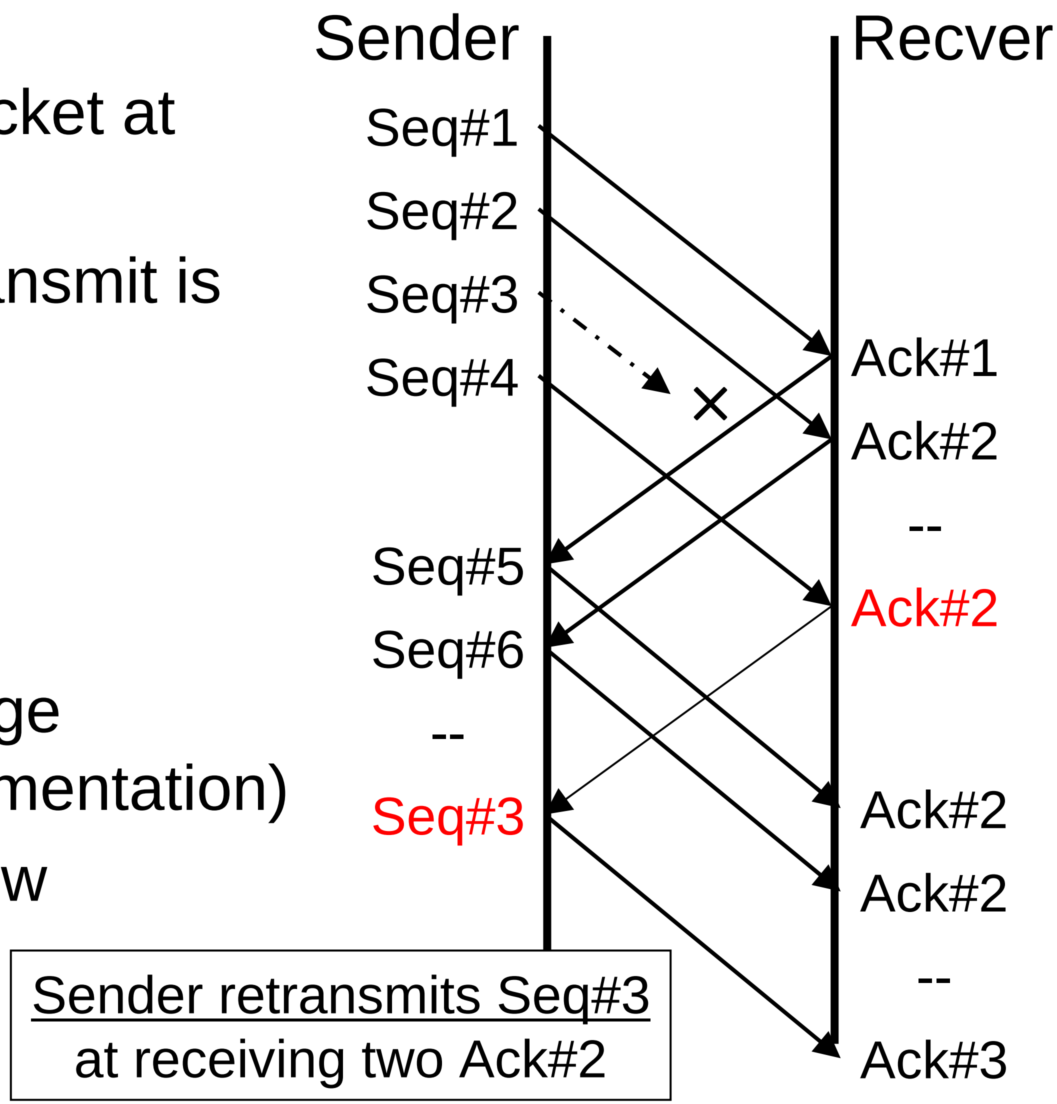
Review of TCP (Slow-Start)

- Basic behavior of TCP
  - cwnd = send buffer size = number of packets in-flight
    - estimated bandwidth-delay product
    - low cwnd value, low bandwidth utilization
  - <u>cwnd is low during Slow-Start</u>
  - <u>cwnd adapts slowly during Congestion Avoidance</u>



Review of TCP (Fast-Retransmit)

- Fast-Retransmit
  - Sender resends lost packet at duplicate ACK.
  - Recovery by Fast-Retransmit is fast
- Retransmit-Timeout (RTO)
  - Original mechanism of retransmission.
  - Timeout time is very large (≥200ms in Linux implementation)
  - Recovery by RTO is slow



TCP Behavior versus MPI Traffic

- Quick changes of communication patterns

  ⇒ cwnd mismatch

  - From all-to-all to one-to-one, vice versa
  - Adapting cwnd is slow, taking some round-trip time.
- Phases of computation/communication

  ⇒ Frequent Slow-Start

  - Long pause forces TCP to enter Slow-Start state
  - ≥200ms in Linux implementation
- Non-contiguous flow of packets

  ⇒ Long pause waiting for Retransmit-Timeout

  - Drop of a tail of a message cannot be recovered by Fast-Retransmit. Falls back to Retransmit-Timeout.

Three Modifications to TCP Behavior

Grid Technology Research Center AIST

Modifications	Addressed Problems
Pacing at Start-up	■ Frequent Slow-Start ■ Losing ACK-clocking
Reducing RTO (Retransmit-Timeout) time	■ Long pause waiting for Retransmit-Timeout
TCP Parameter Switching	■ Mismatch of cwnd value at communication phase changes

- Actual implementations diverge from the definition of TCP, but behave essentially the same at non-contiguous point.
- All problems frequently happen in MPI applications.

Implementation of Pacing at Start-up

- Pacing at Start-up
  - Pacing keeps constant pace during start-up.
  - Avoid burst traffic, to avoid router buffer overflow.
  - No ACK-clocking at start-ups, and pacing is needed.
- Target bandwidth
  - target bandwidth = (ssthresh * MSS) / RTT
  - ssthresh holds saved cwnd value
  - Target is checked every 10μsec. Packet is sent when pace is lower than the target
- Use 10μsec timer interrupt for clock generation
  - Use APC Timer of IA32 CPU
  - Timer stops at receiving ACK (within RTT)
  - Small overhead, 1% to 2% slowdown

Implementation of Parameter Switching

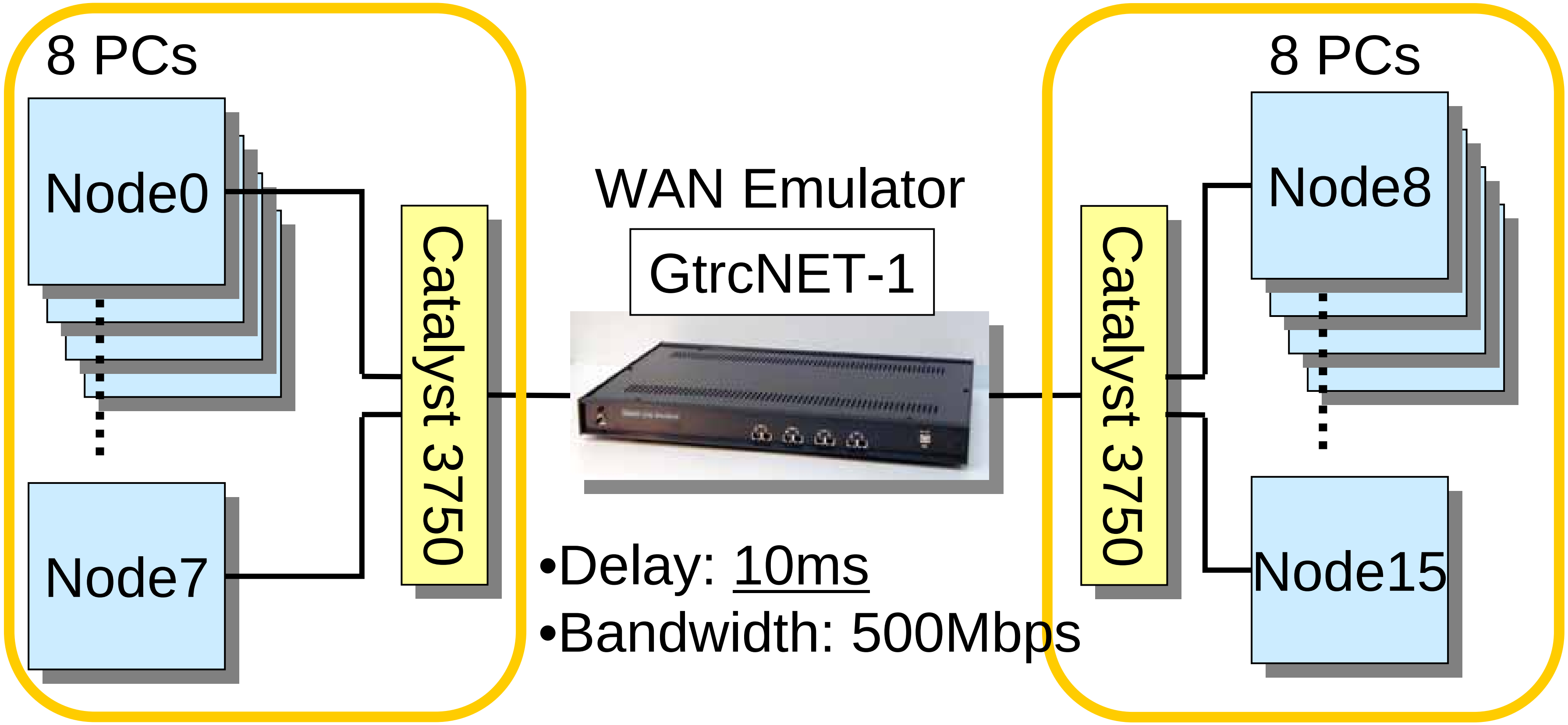
- TCP Parameter Switching
  - Save/restore ssthresh
  - ssthresh holds last cwnd value
  - ssthresh is stable, easy to save/restore
- Start communication at a rate by ssthresh
  - Use Pacing at Start-up
- Parameter Switching is activated at pattern changes
  - MPI library notifies pattern changes to TCP
  - Before/after collective communication calls
    - Two ssthresh values, one for point-to-point and one for collectives, are saved in a communicator.

Implementation of Reducing RTO Time

- Reducing RTO Time
  - Use the definition in RFC, without rounding up to 200ms
    - RTO = SRTT + 4 * RTTVAR

      (SRTT: Smoothed RTT, RTTVAR: RTT Variance)
- Linux implementation
    - RTO = SRTT + MAX(200, 4 * RTTVAR)
  - Roughly (200ms + RTT)
- NOTE: RTO Time is used for two purposes
    - Time to retransmission
    - Time to entering Slow-Start state
  - Reducing RTO has side-effect of frequently entering Slow-Start state.

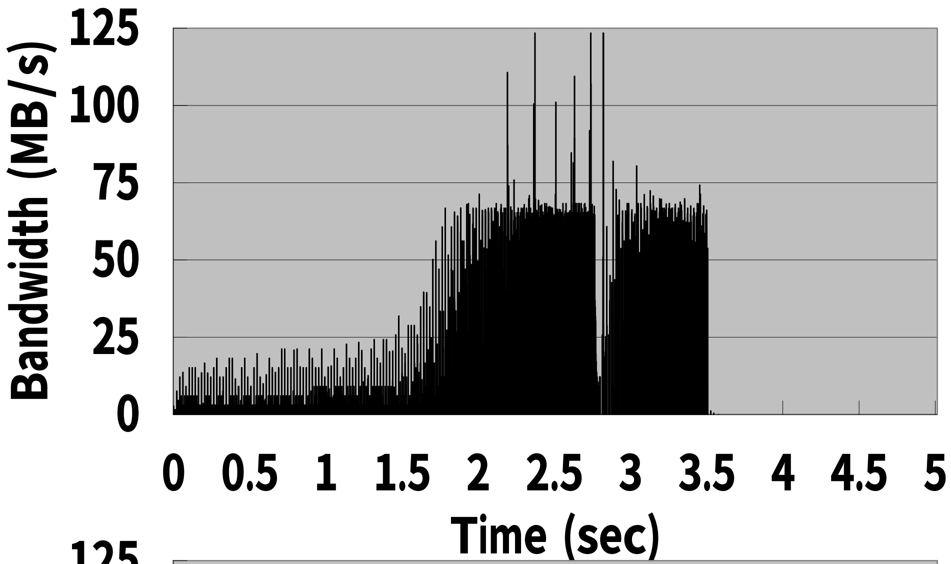
Evaluation Setting



- CPU: Pentium4/2.4GHz, Memory: DDR400 512MB
- NIC: Intel PRO/1000 (82547EI)
- OS: Linux-2.6.9-1.6 (Fedora Core 2)
- Socket Buffer Size: 20MB

Effect of TCP Parameter Switching

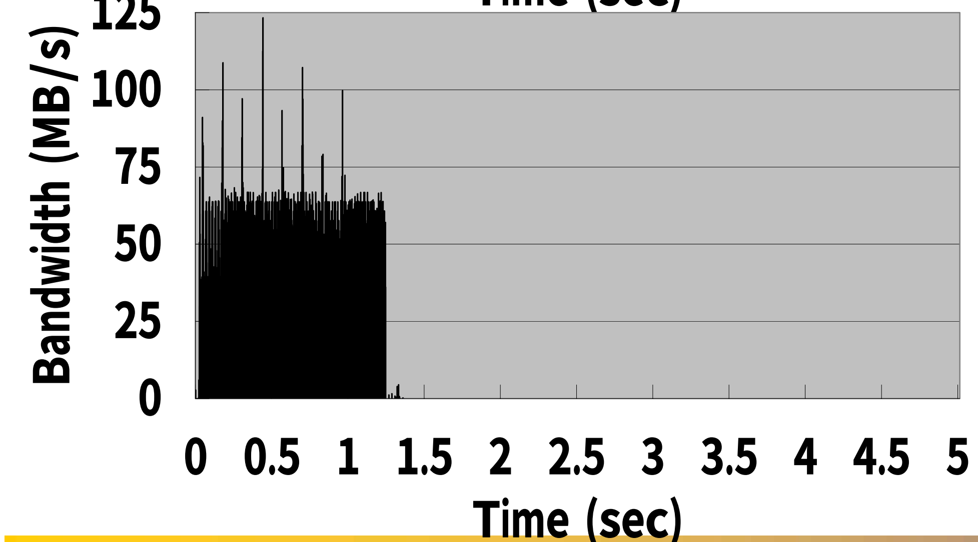
- Start one-to-one communication at time 0 after all-to-all.



- Without TCP Parameter Switching

 (traffic starts with cwnd=49)

  - Graphs show sending 50MB data.
  - cwnd is reduced after all-to-all communication.

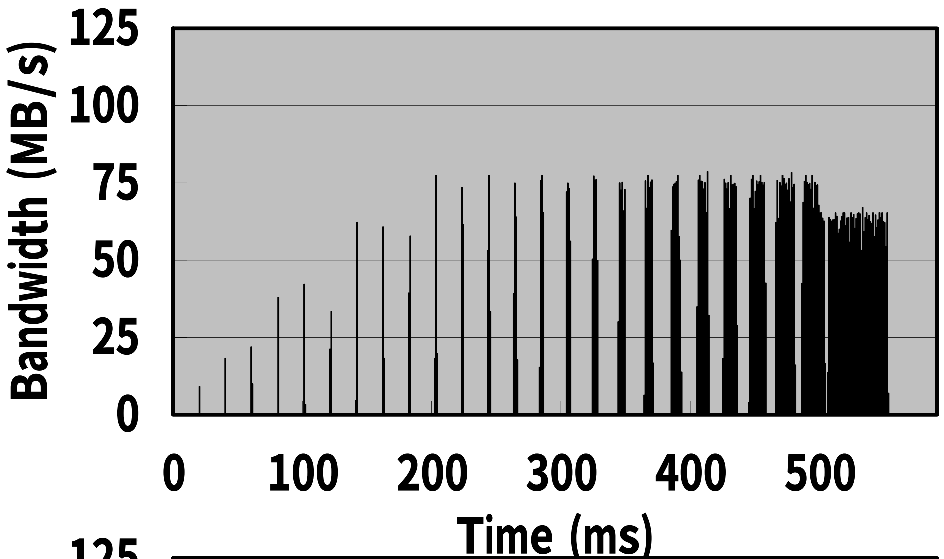


- With Switching

 (traffic starts with cwnd=582)

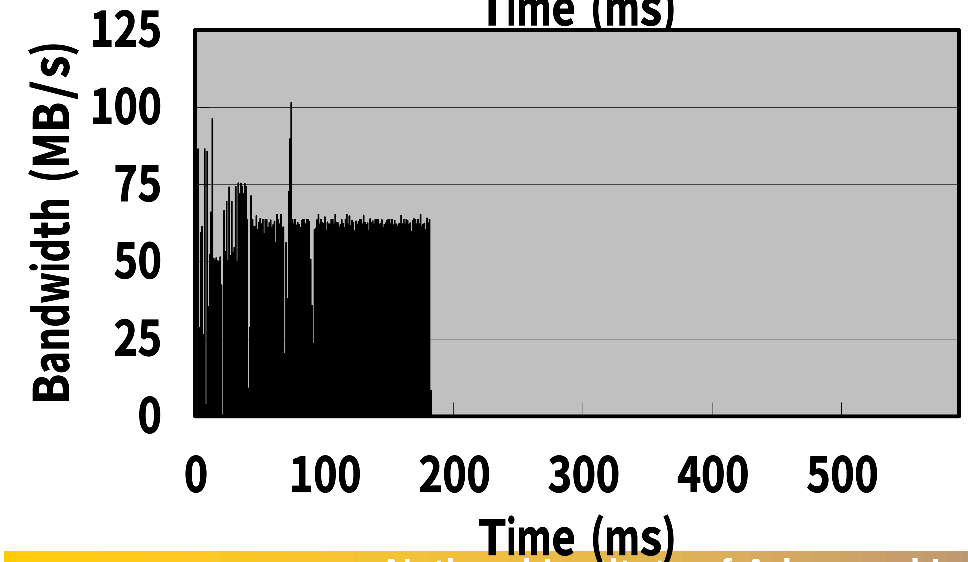
Effect of Pacing at Start-up

- Repeat sending 10MB data with 2second intervals.



- Without Pacing at Start-up

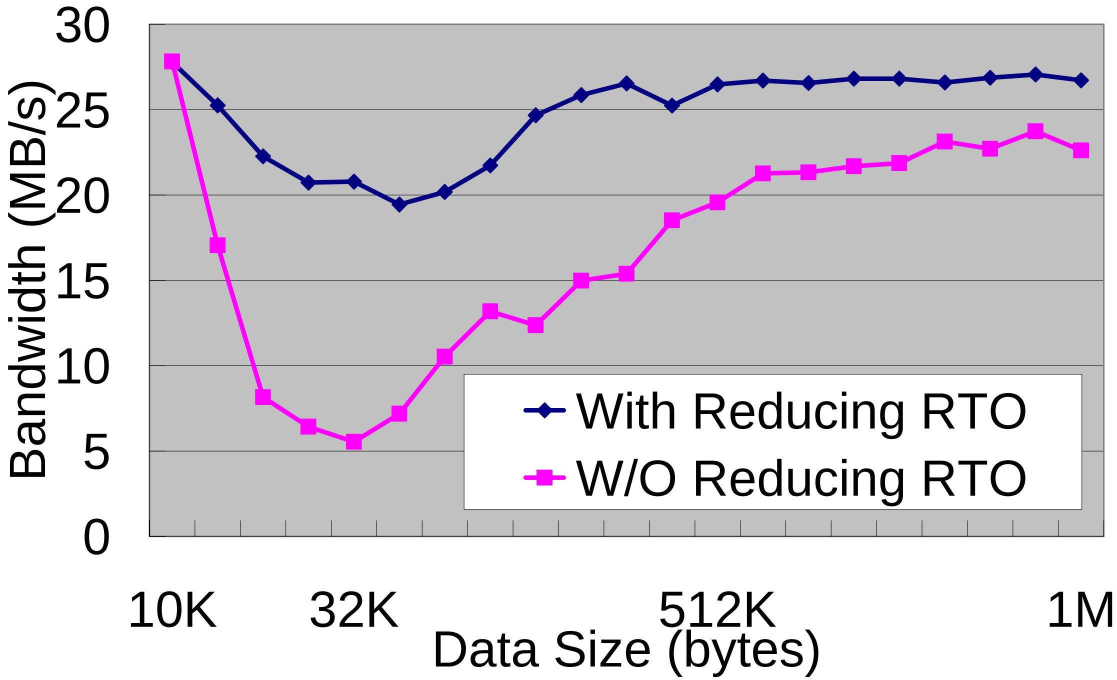
- Graphs show sending one chunk of 10MB data



- With Pacing at Start-up

Effect of Reducing RTO Time (Bandwidth)

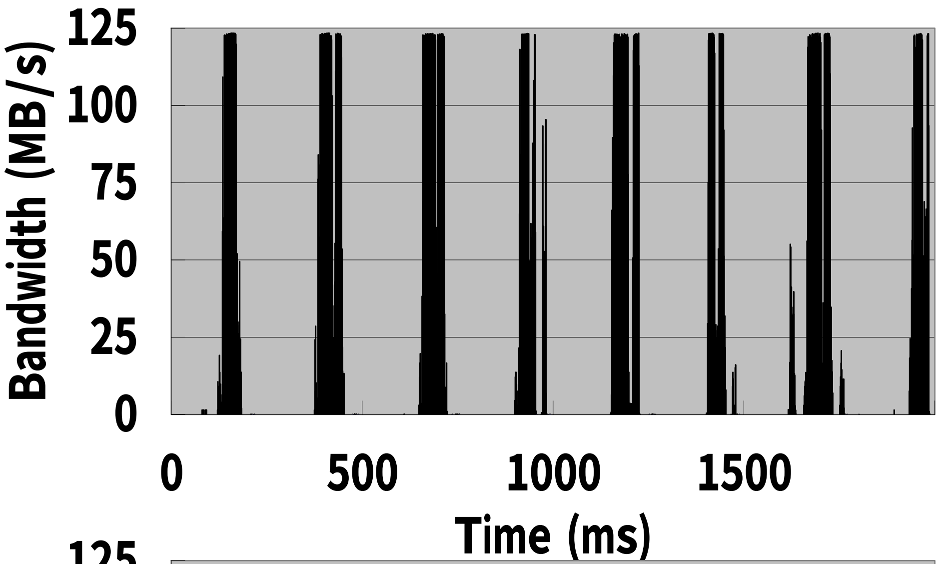
- Repeat MPI_Alltoall with varying data size.



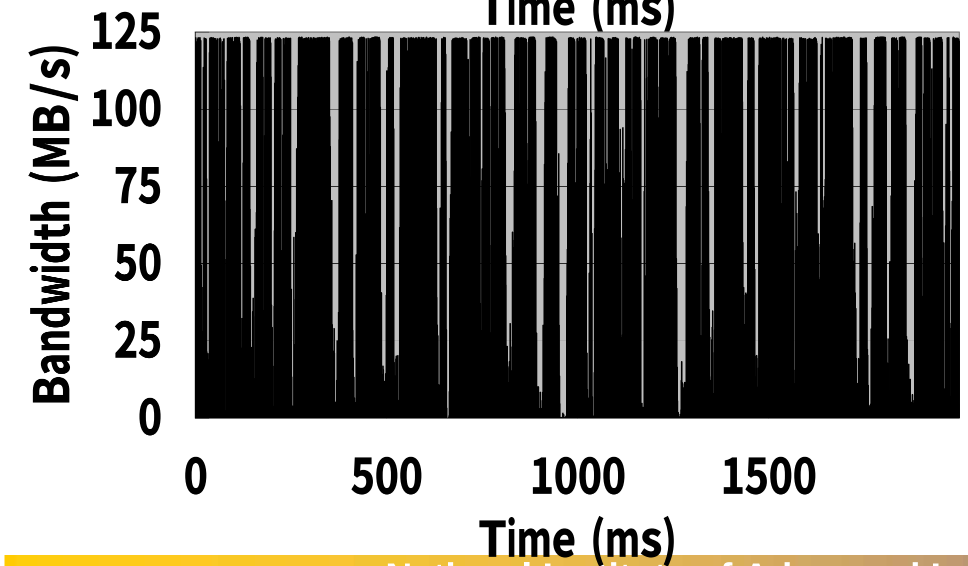
- Setting is a cluster environment, no delay, no bandwidth restriction.
- Packets are dropped at the switch between clusters, which causes Retransmit-Timeout.

Effect of Reducing RTO Time (Traffic)

- Repeat all-to-all communication with data size 32KB.

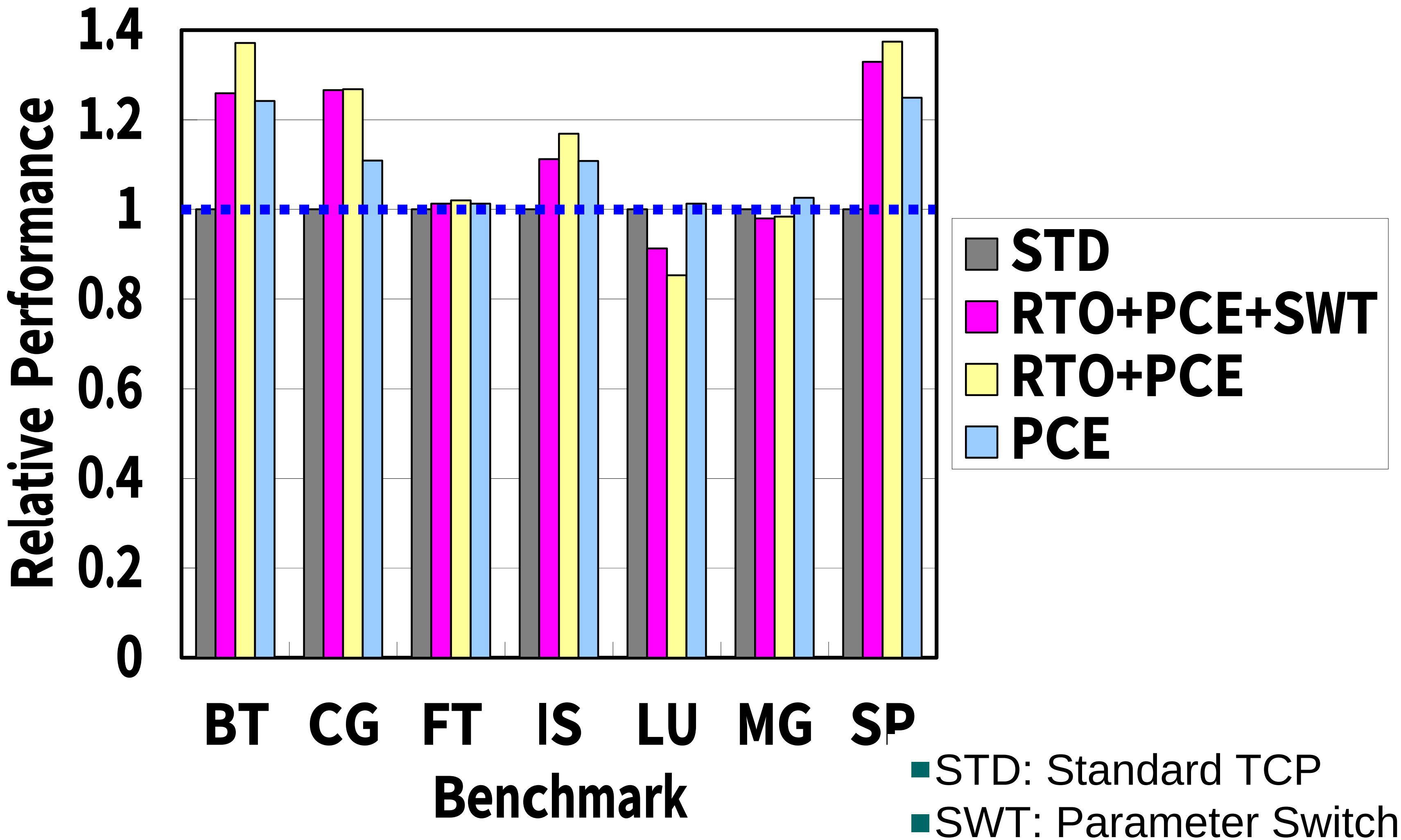


- Without Reducing RTO Time
  - Upper graph has long pauses despite of repeated calls of MPI_Alltoall



- With Reducing RTO Time

NPB Benchmarks



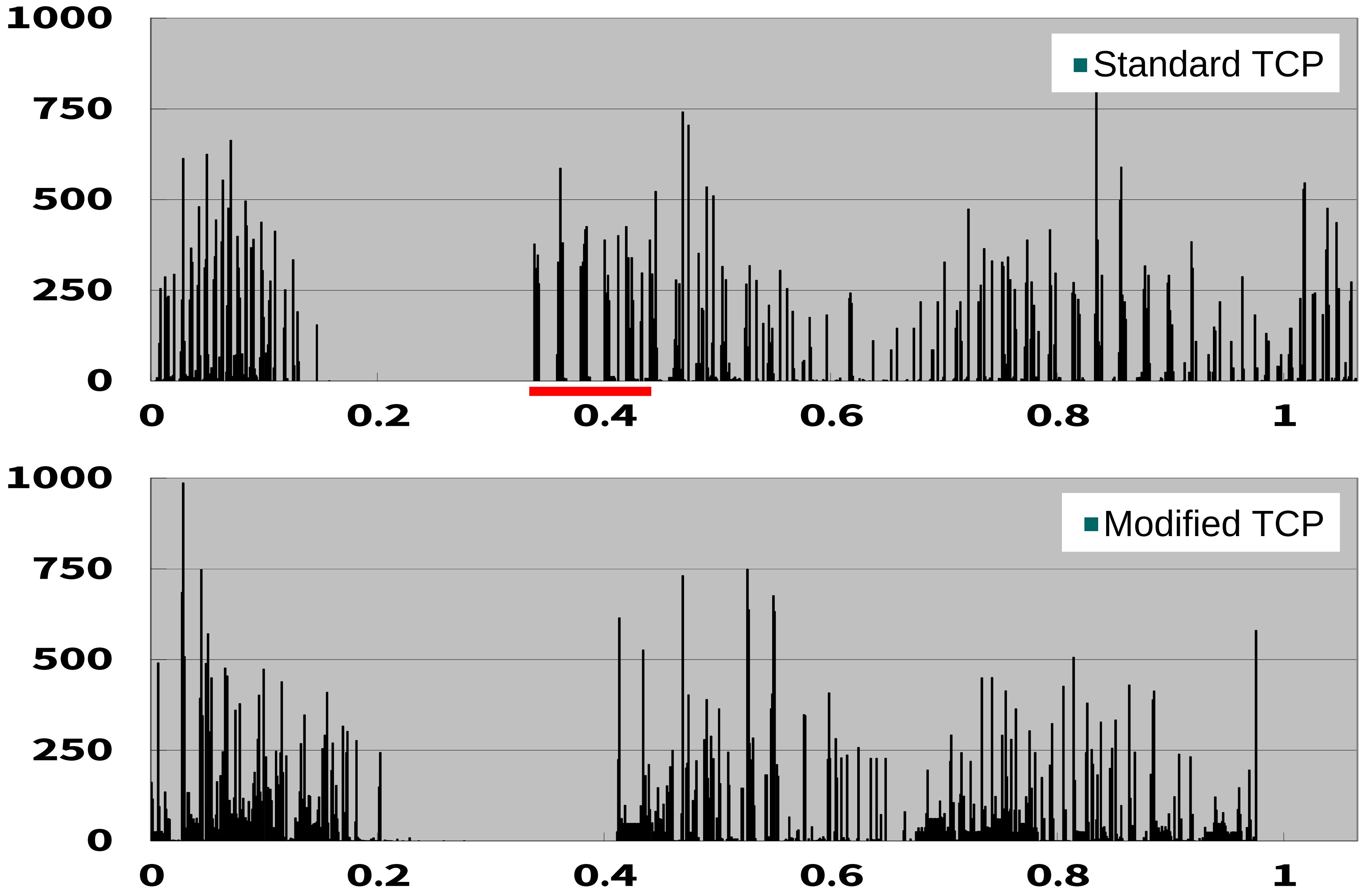
- STD: Standard TCP
- SWT: Parameter Switching
- RTO: Reducing RTO
- PCE: Pacing at Start-up

- NAS Parallel Benchmarks2.3
- Relative performance to the standard TCP

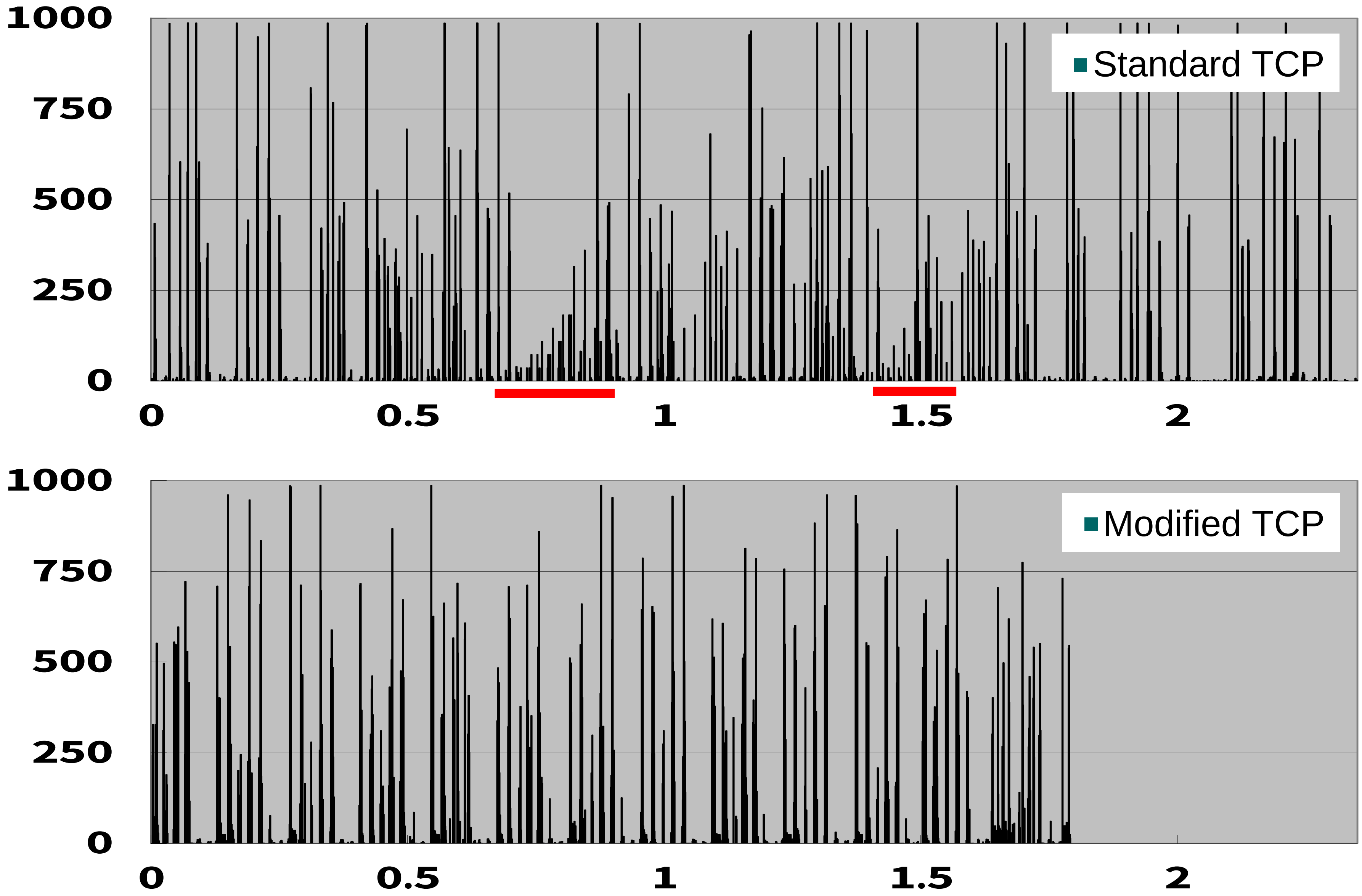
NPB Traffic Samples

- Traffic samples taken from NPB
- Sampling point
  - Traffic between two clusters
  - Sum of the traffic from 8 nodes
- Extraction of one benchmark loop
  - Some deviation on time 0 point, due to trigger

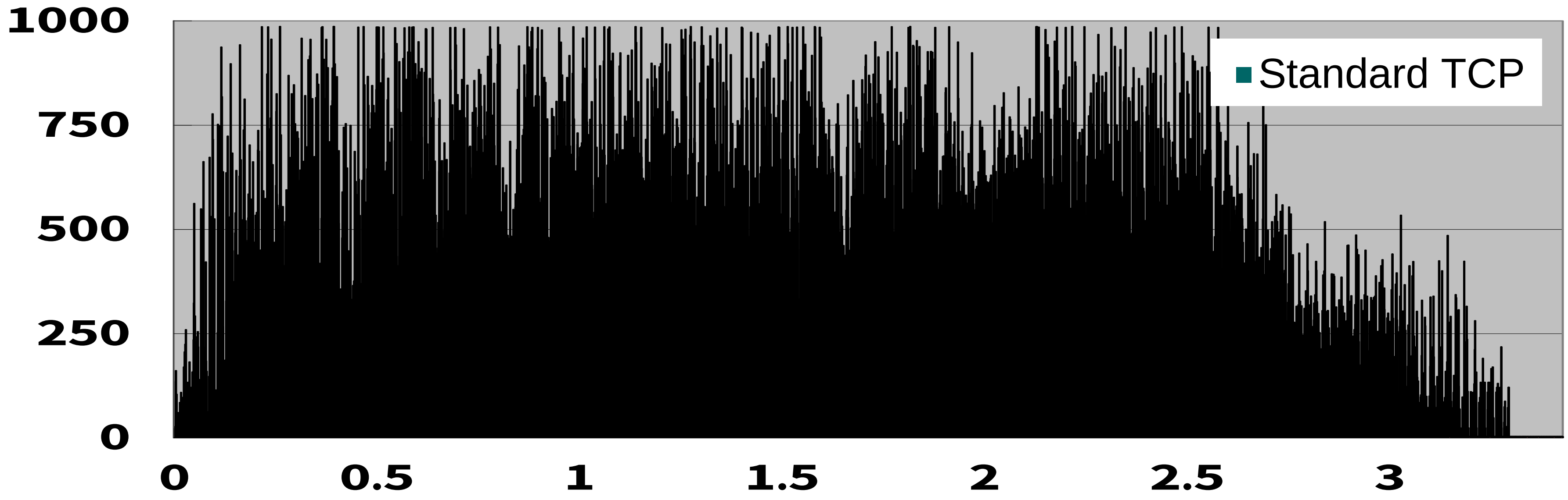
Traffic of BT Benchmark

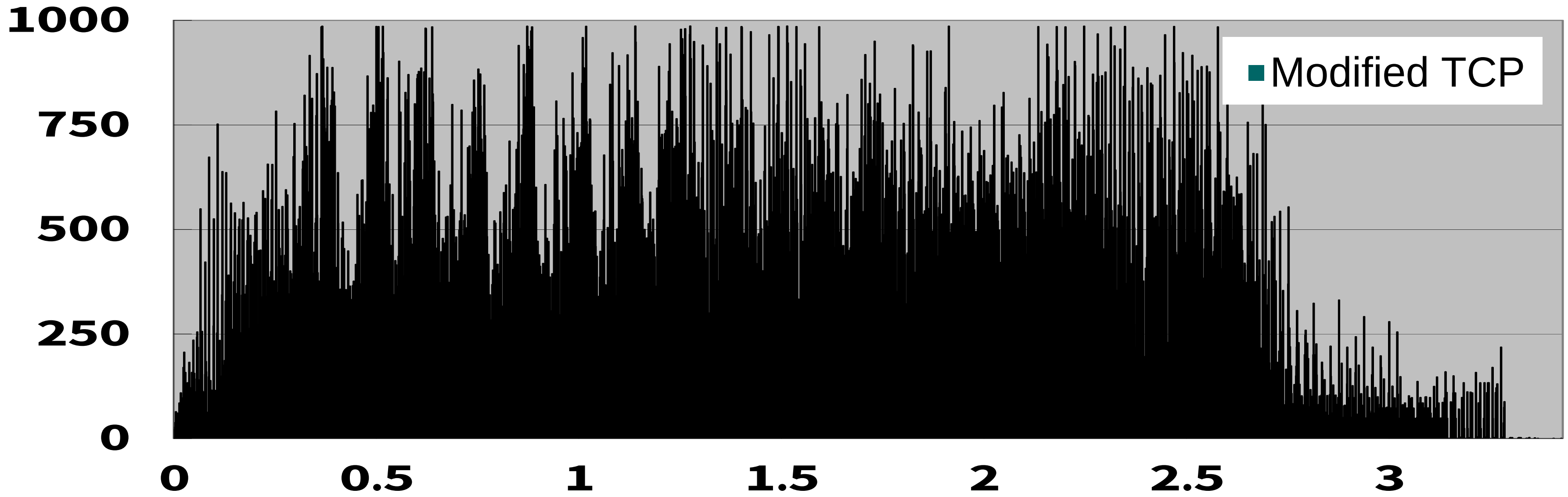


Traffic of CG Benchmark

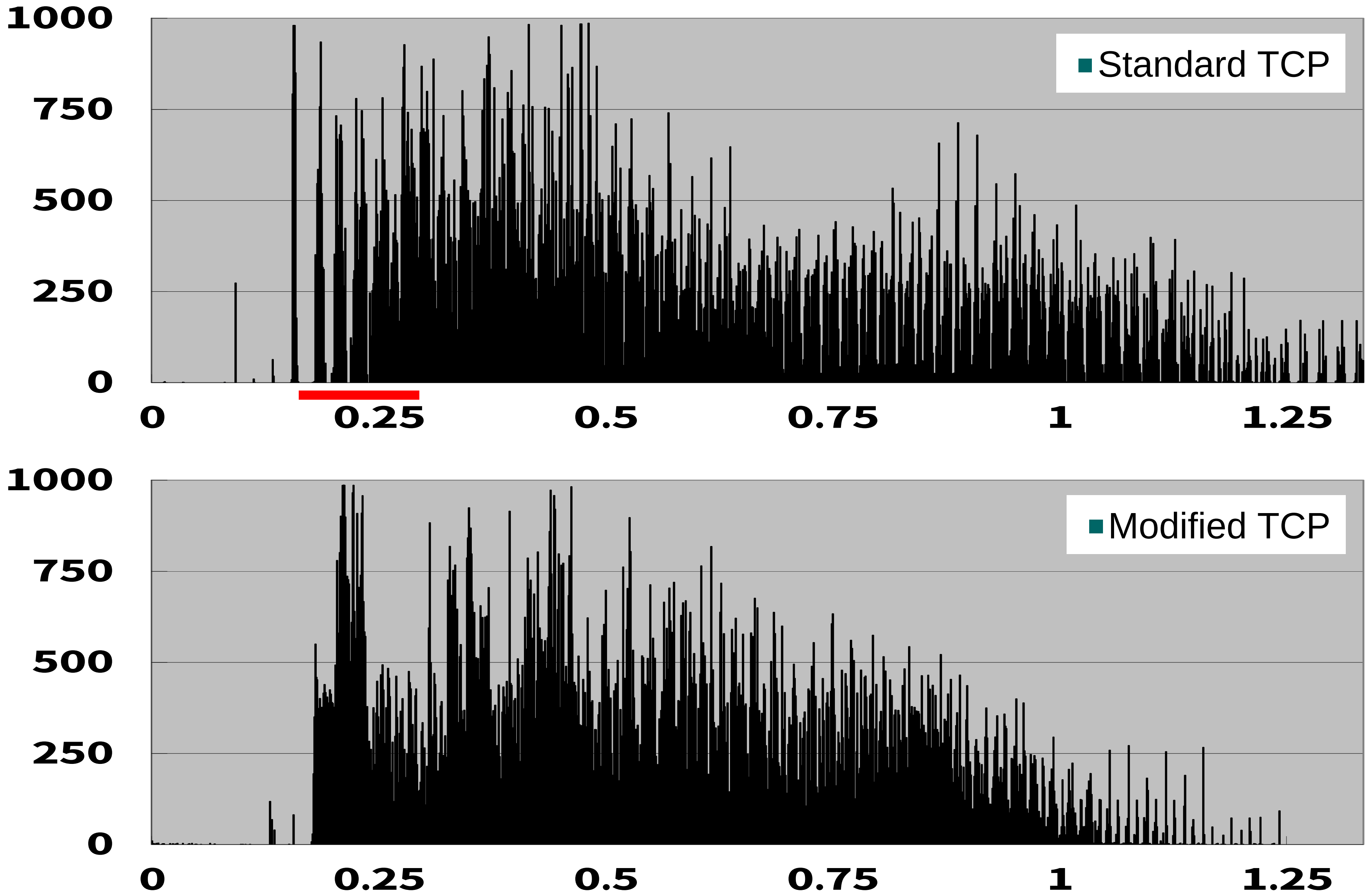


Traffic of FT Benchmark

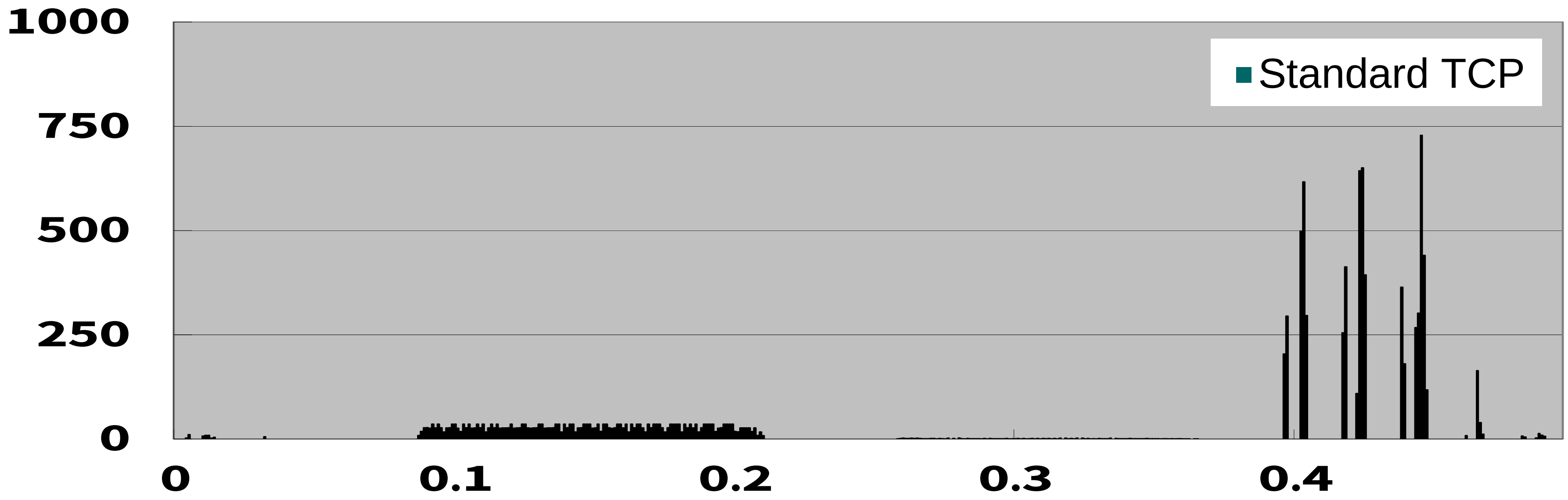


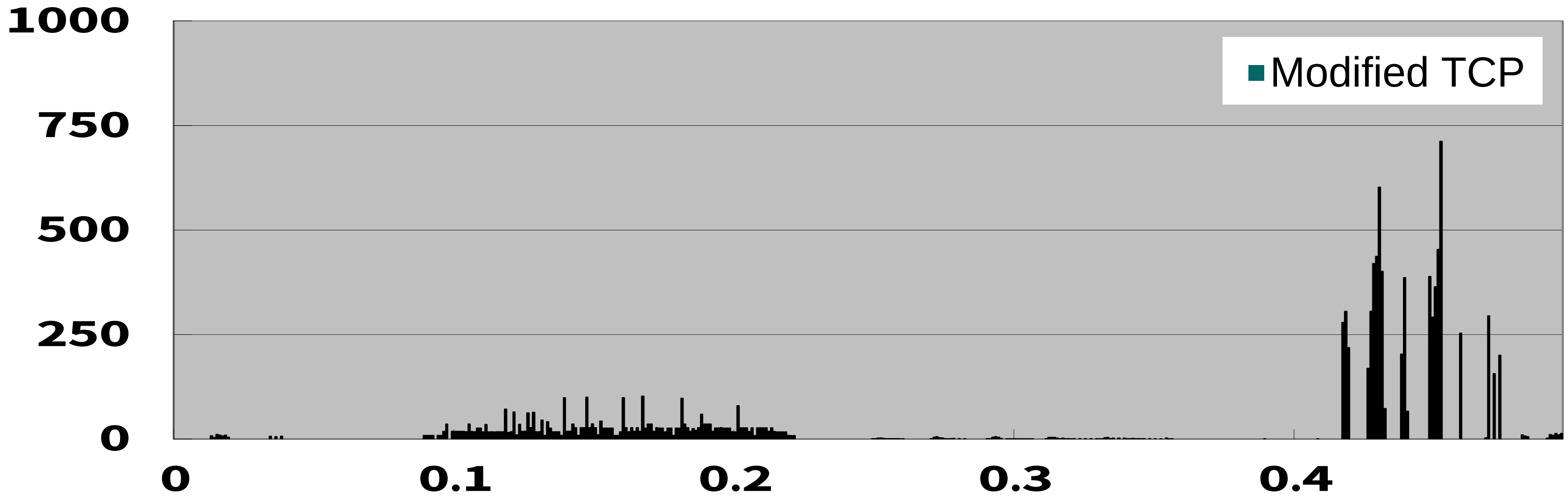


Traffic of IS Benchmark

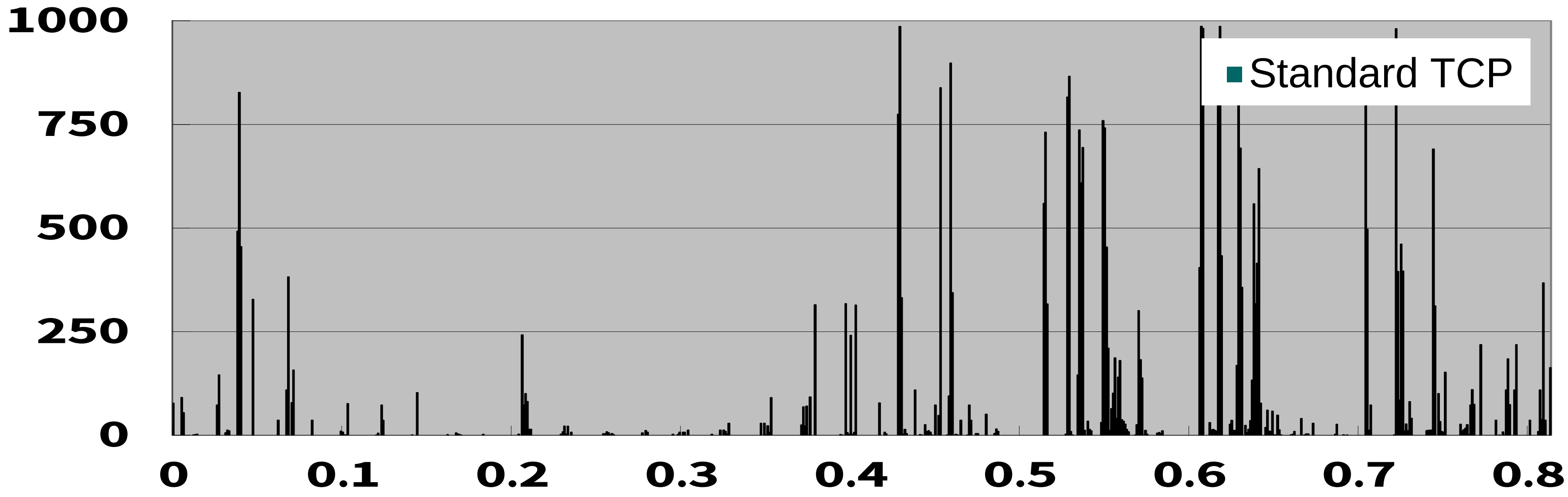


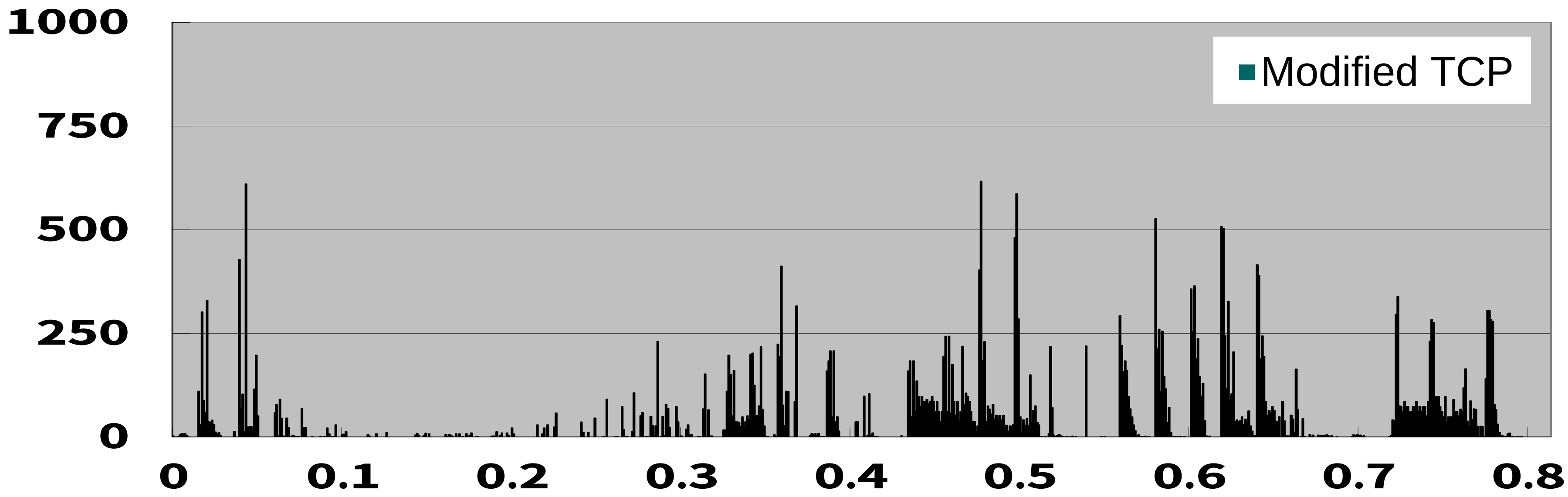
Traffic of LU Benchmark



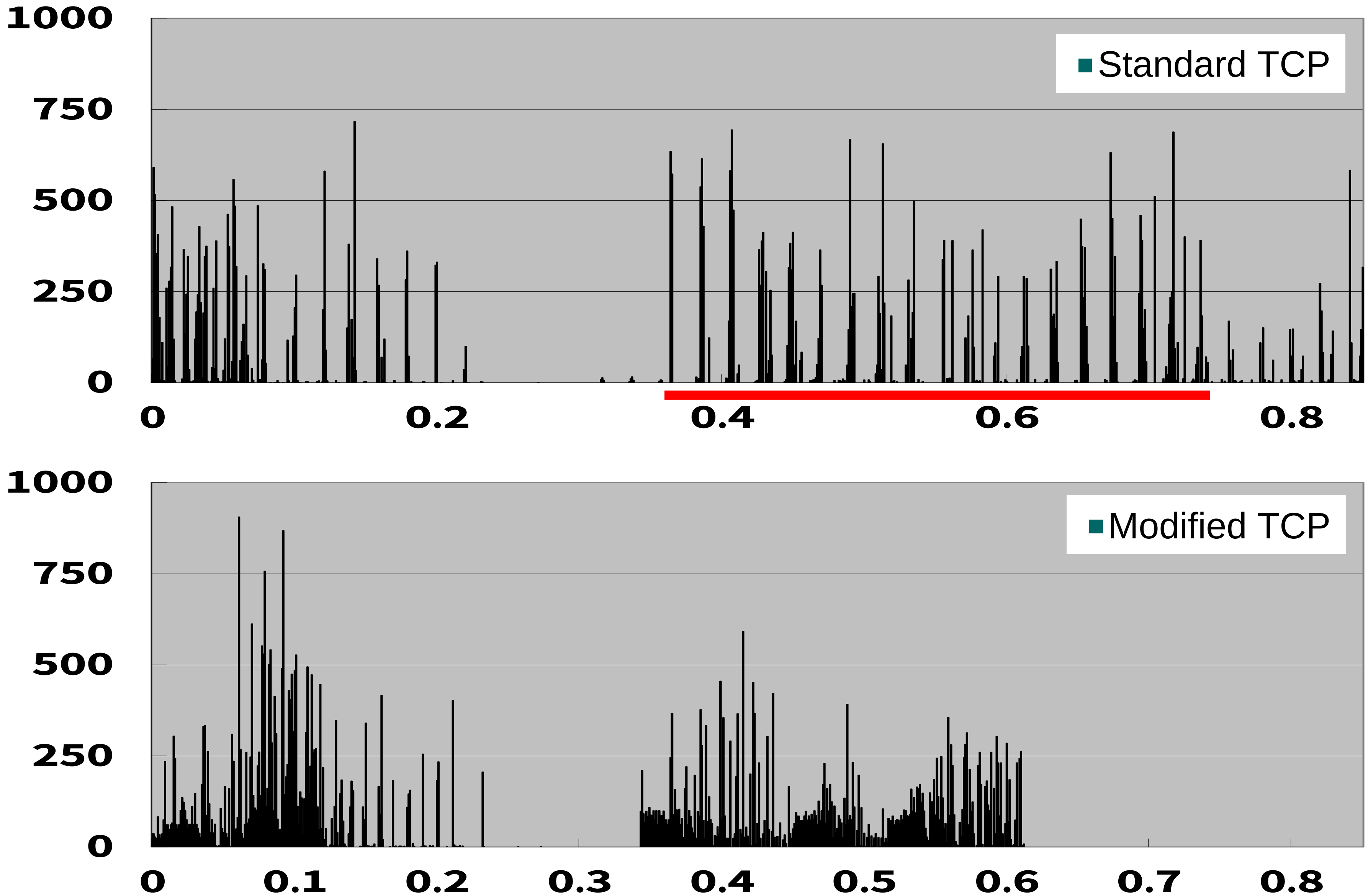


Traffic of MG Benchmark





Traffic of SP Benchmark



Related Work

- Issues are old:
  - Old networks and OS [Aron,Durschel]
    - Evaluation with 10Kbps-100Kbps network
    - Coarse timers, improvement from 500ms to 10ms
  - Focus on HTTP traffic [Hughes,Touch,Heidemann]
    - MPI is much sensitive to the issues
  - Most evaluations are simulation
- Our approach:
  - Cooperation with MPI library
  - Implemented in the Linux TCP stack
  - Fast timer for 1Gbps to10Gbps environments

Conclusion

- Modifications in TCP behavior at start-ups
  - Pacing at Start-up $\Rightarrow$ avoid frequent Slow-Start
  - Reducing RTO Time $\Rightarrow$ avoid long pause at packet losses
- Cooperation with MPI library
  - TCP Parameter Switching $\Rightarrow$ avoid cwnd mismatch
- NO CHANGES in Congestion Avoidance phase
  - Start-up behavior is mostly ignored, but it affects the performance of some MPI applications largely.
  - Fairness will be retained, because no modification in Congestion Avoidance phase.

GridMPI™ Project Page

http://www.gridmpi.org/

END