Proportional Time Emulation and Simulation of ATM Networks

Sean B House
shouse@kualumni.org

Thesis Defense for the Degree of
Master of Science in Computer Engineering
Department of Electrical Engineering & Computer Science
University of Kansas

December 8, 2000

Agenda

- Overview & Motivation
  - *What is the problem?*
- Related work
  - *What other solutions to the problem exist?*
- Proportional Time Emulation and Simulation
  - *How did we choose to solve the problem?*
- Evaluation of ProTEuS
  - *How well did we solve the problem?*
- Conclusions and future work
  - *What improvements could be made to our solution?*

Overview & Motivation

- The problem is how to enable efficient simulations of large networks for long periods of simulated time
- Performance evaluation criteria
  - Execution time
  - Verity
  - Scalability
  - Accessibility
- Relevance of each varies with application
- Efficient network simulations justify specialized system support to improve simulation performance
  - Fine-grained scheduling and embedded system support

Overview & Motivation

- Sequential discrete event simulations
  - Don't scale well with respect to size nor simulated time
  - Faster single processors are not sufficient to offset scaling
- Parallel discrete event simulations
  - Scale better in both dimensions
  - Generally suffer performance deterioration when the application is fine-grained
    - ATM network simulations are fine-grained, CPU bound computations
  - Optimistic synchronization increases concurrency, but
    - Permits temporal violation
    - Requires significant overhead to detect and recover

Overview & Motivation

- ProTEuS treats Linux workstations as embedded components of a distributed simulation
  - KU Real-Time (KURT) controls synchronized distributed simulations of ATM networks
    - Increases scheduling granularity
    - Increases CPU utilization
    - Simplifies synchronization of distributed entities
  - Physical ATM network provides an essentially error-free, low-latency communication medium between hosts
- The methods employed by ProTEuS ATM network simulations are applicable to other synchronous distributed applications as well

Sequential Discrete Event Simulation

- BONeS, OPNET, Extend, Ns/VINT…
- Scale poorly in network size and simulated time
  - Limited to performance improvement of single processors
- Requires re-implementation of network algorithms
  - Opportunity for error
  - Encourages simplifying abstractions
    - Simpler generally easier to create
    - Simpler generally faster
    - Simpler often less accurate
- Ns/VINT chooses abstraction over parallelization
  - Some recent work by GA Tech PADS to parallelize Ns

Parallel Discrete Event Simulation

- Fundamental rule of discrete event simulation
  - Events must be executed in order of non-decreasing virtual time
    - Sequential simulations utilize a single sorted event queue
    - Parallel simulations need synchronization to enforce temporal causality amongst processors
  - Conservative synchronization
    - Events executed in virtual timestamp order without exception
    - May unnecessarily impede the progress of a simulation
  - Optimistic synchronization
    - Events executed at time of arrival, perhaps resulting in violation
    - Stragglers events must be detected and corrected through rollback

Parallel Discrete Event Simulation

- GTW, WARPED, ParaSol, ParSEC…
  - Based on Jefferson's Time Warp principle
  - Utilize optimistic concurrency control
    - State-saving and rollback overhead dominate performance
  - GTW arguably the most popular rendition of Time Warp
    - Optimized for fine-grained applications
  - ParaSol includes Ariadne user-level threading
    - Increases simulation control over execution
  - WARPED includes many Time Warp optimizations
    - Cancellation (anti-messages and rollback)
    - Check-pointing (state-saving)
    - Message aggregation (communication overhead)

Parallel Discrete Event Simulation

- Scalable Simulation Framework (SSF)
  - Utilizes conservative concurrency control
    - Boundary synchronization similar to ProTEuS
  - Includes its own user-level pseudo-thread implementation
    - Makes the scheduler *simulation aware*
- Requires re-implementation of network algorithms
  - Encourages simplifying abstractions
- Improving performance can entail significant tuning
  - Synchronization, state-saving, GVT update, etc.
- Favors shared memory multiprocessors
  - NOWs are viable, but performance suffers

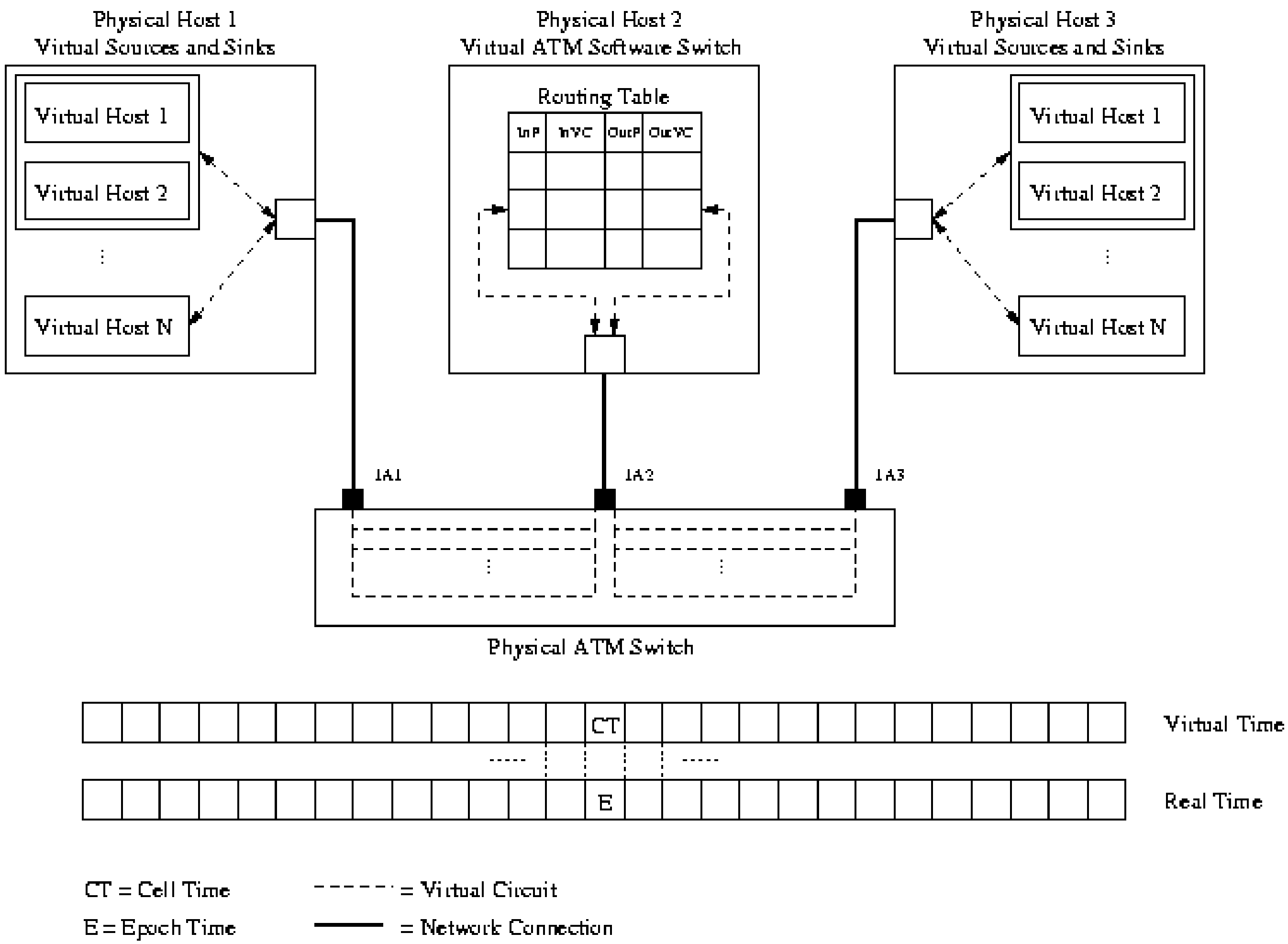
ProTEuS

- Essentially configures a rack of Linux workstations as an embedded system
  - Provides the ability to use real system code
    - Virtual network devices localize virtual time line control
    - E.g., requires modifying only the timer handling in TCP layer
- Real-time control (KURT)
  - Fine-grained system-level scheduling control (UTIME)
  - Real-Time scheduling synchronizes at primary level
    - Application determines the sufficiency
    - Simple secondary synchronization ensures correctness
  - Global synchronization avoids state-saving and rollback

ProTEuS

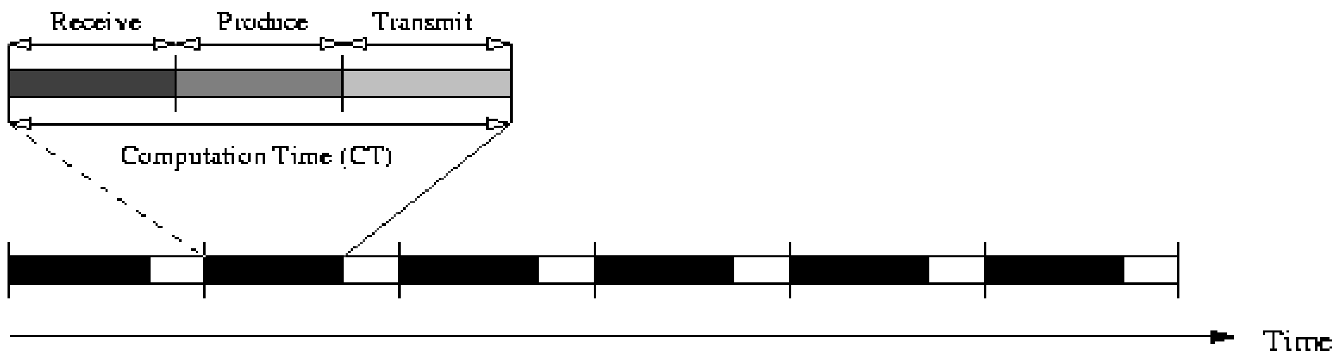
- Virtual Network Device layer
  - Maps the simulated network onto the physical network
  - Provides simulated network device functionality
    - ATM Traffic shaping
    - Segmentation and Re-assembly (SAR)
- Virtual ATM Software Switch
  - Provides cell- and packet-level ATM switching
  - Provides Available Bit Rate (ABR) congestion feedback
- NetSpec controls simulation execution
  - Distributes jobs to simulation hosts and collects results

ProTEuS



Physical Host 1
Virtual Sources and Sinks
Virtual Host 1
Virtual Host 2
Virtual Host N
Physical Host 2
Virtual ATM Software Switch
Routing Table
InP
InVC
OutP
OutVC
Physical Host 3
Virtual Sources and Sinks
Virtual Host 1
Virtual Host 2
Virtual Host N
1A1
1A2
1A3
Physical ATM Switch
CT
Virtual Time
E
Real Time
CT = Cell Time
E = Epoch Time
= Virtual Circuit
= Network Connection

ProTEuS



- Epoch is the fundamental unit of virtual time
  - Represented proportionally by real-time period
  - Computation: Receive, Produce, and Transmit messages
  - Slack: Allows remote messages to arrive
- KURT coarsely synchronizes epoch execution
  - Application semantics determine sufficiency
  - Secondary synchronization protocol ensures correctness

Challenges

- Scheduling jitter
  - Linux bottom halves
  - Interrupt service
    - Especially for ProTEuS messages (multiplexing)
  - Impact depends on epoch length
- Clock synchronization
  - Epoch boundaries not synchronized in absolute time
  - Rate of increment (frequency) not synchronized
- Application semantics may help offset
  - Messages produced in epoch *N* are used in epoch *N*+*?*
  - Application determines *?*

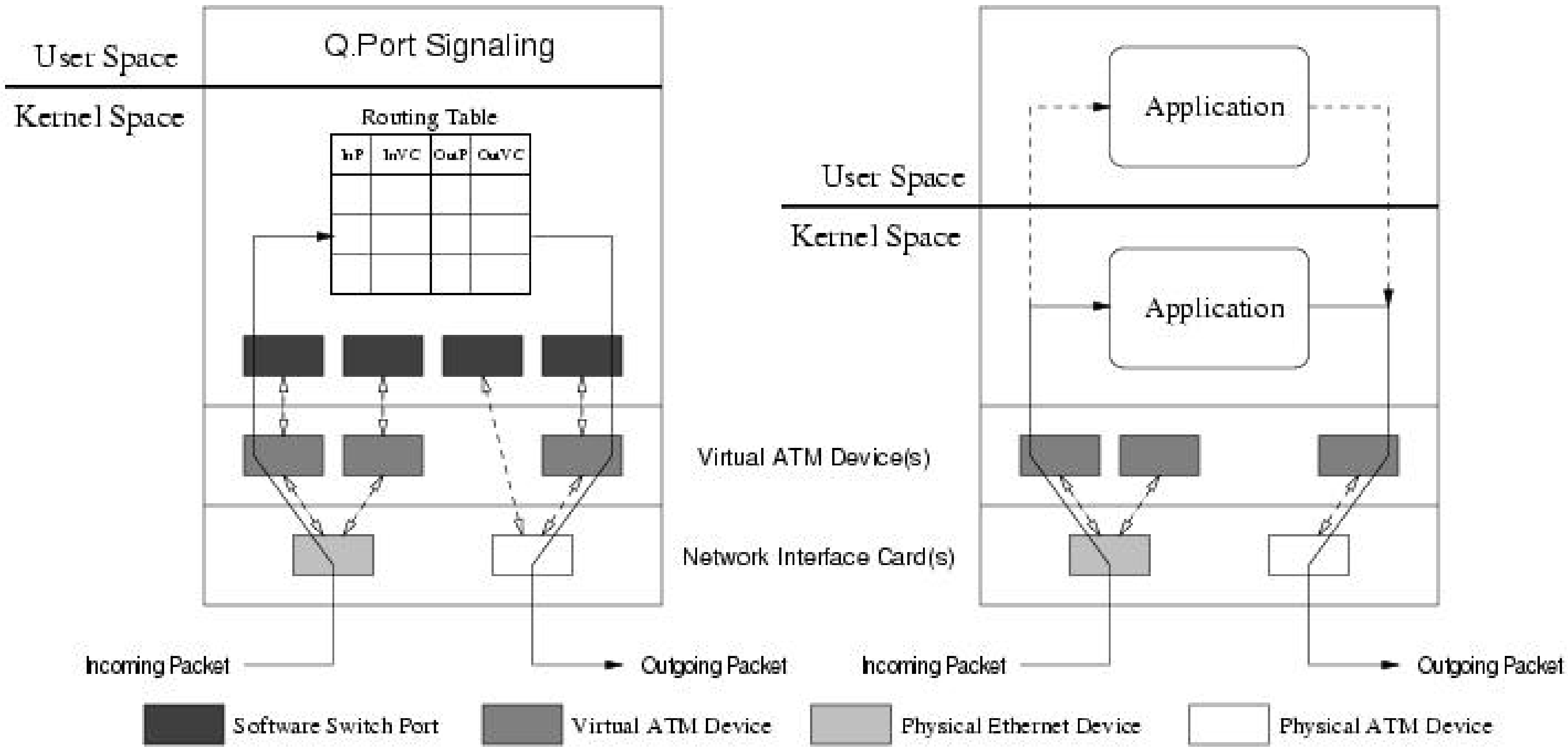
Virtual Network Devices

- Linux pseudo device driver (module)
- Kernel-level abstraction
  - Creates a virtual device associated with a physical device
  - Comprised of configurable layers
    - SAR, PT, QoS, etc.
  - To the protocol stack, it looks like a physical device
  - To the physical device, it looks like a protocol
- Virtual device need not implement the same network medium as the physical device
  - Virtual ATM over Ethernet
  - Virtual ATM over ATM

Virtual ATM Software Switch

- Linux pseudo device driver (module)
- Maps an incoming (Port, VC) to outgoing (Port, VC)
- Implements three output port queueing disciplines
  - Per-class
  - Per-VC
  - Shared backlog
- ABR feedback provided by EPRCA or ERICA
- Operates in one of two modes
  - Best-effort implemented as a Linux bottom half
  - Periodic real-time service under KURT control

Virtual Network Devices



NetSpec

- Distributed network performance evaluation tool
- Block-oriented script describes an experiment
  - Blocks describe jobs that are distributed to hosts
  - Daemon processes on hosts perform jobs and report results
- Virtual Network Device Daemon (*nsvdevd*)
  - Creates/configures virtual devices
    - Including control of Proportional Time
- Virtual ATM Software Switch Daemon (*nsvswitchd*)
  - Creates/configures switch ports &ATM VC routing entries
- Test daemon (*nstestd*)
  - Sources/sinks ATM, UDP or TCP traffic

ATM Network Simulation

- Virtual device layer maps a virtual ATM network onto a physical ATM network
  - Mapping is extremely flexible – almost arbitrary
    - IP loop back and (monolithic) software switch only restrictions
- Real-time scheduling imposes a correspondence between virtual time and real-time
  - ATM cell time is the basic virtual time unit
  - Real-time period depends on the load on the busiest host
- Multiplexing simulated network connections across physical network connections minimizes interrupts
  - Number of interrupts strongly influences epoch time

Evaluation

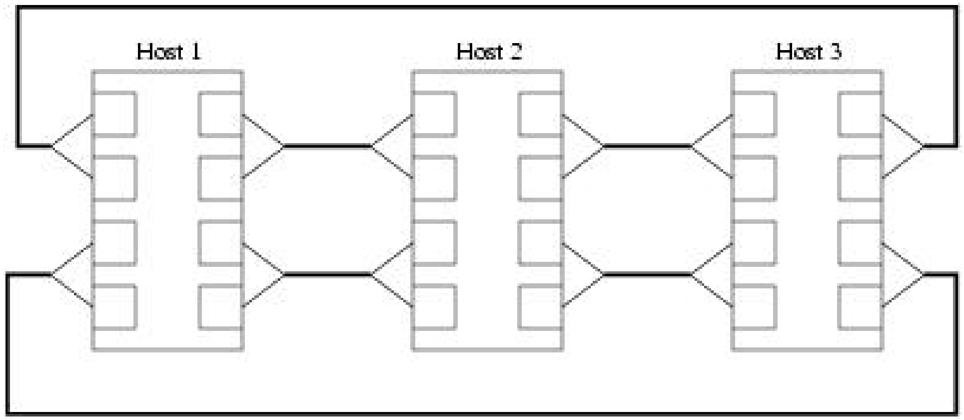
- Experimental setup
  - BONeS – 300 MHz Sun UltraSPARC-II, 512 MB RAM
  - GTW – 168 MHz 8-proc Sun UltraEnterprise, 1GB RAM
  - ProTEuS – 200 MHz Intel x86, 128 MB RAM
- Properties of synchronous distributed applications
  - Load balancing, slack time, delta values (synchronization tolerance), waiting for missing data, and bottom halves
- Faithfulness of ProTEuS for ATM simulation
- ProTEuS vs. GTW
  - Scaling vs. network size and simulated time
  - Scaling vs. round-trip time

Distributed Synchronous Computation

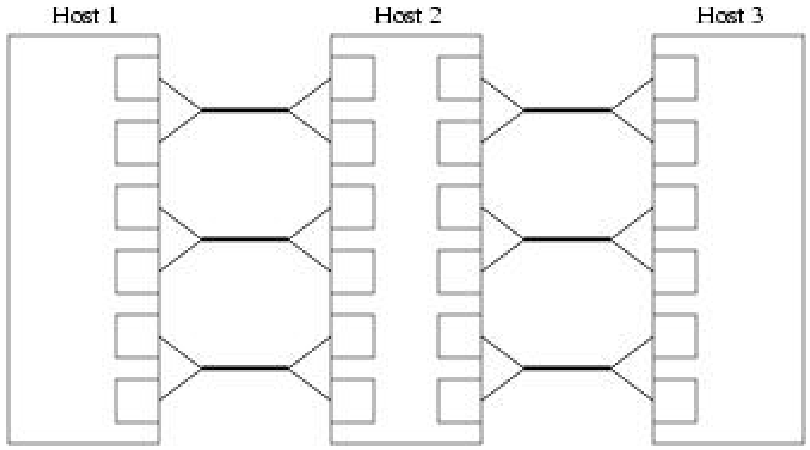
- Generic distributed topology with a fixed computation component per element, per epoch
  - Two distributions with a fixed number of total elements
    - Balanced – roughly equal work per host, per epoch
    - Unbalanced – most heavily loaded host has 2x work per epoch
  - Two granularities of computation
    - 25?s per element per epoch – aggressive fine-grained needs
    - 200?s per element per epoch – coarse granularity, high utilization
  - Epoch computation interval is ? of components
- Investigate the effects of load balancing, slack time choice, synchronization tolerance (?), waiting for missing data, and Linux bottom halves

Distributed Synchronous Computation

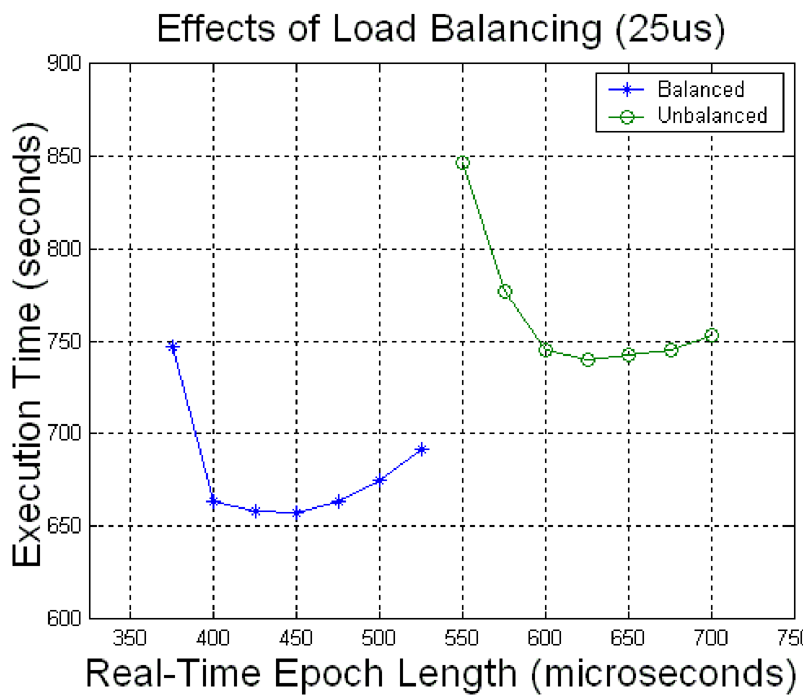
- Balanced topology

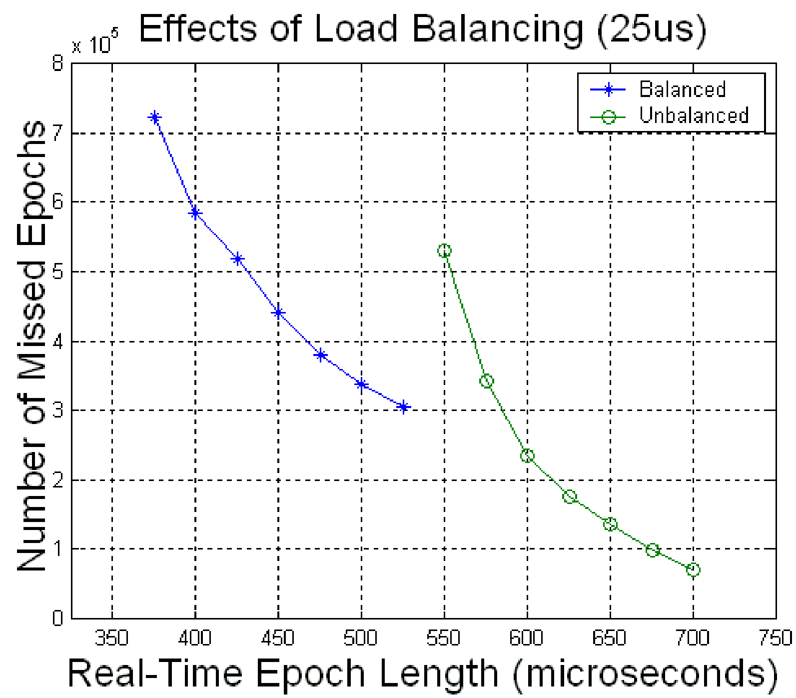


- Unbalanced topology



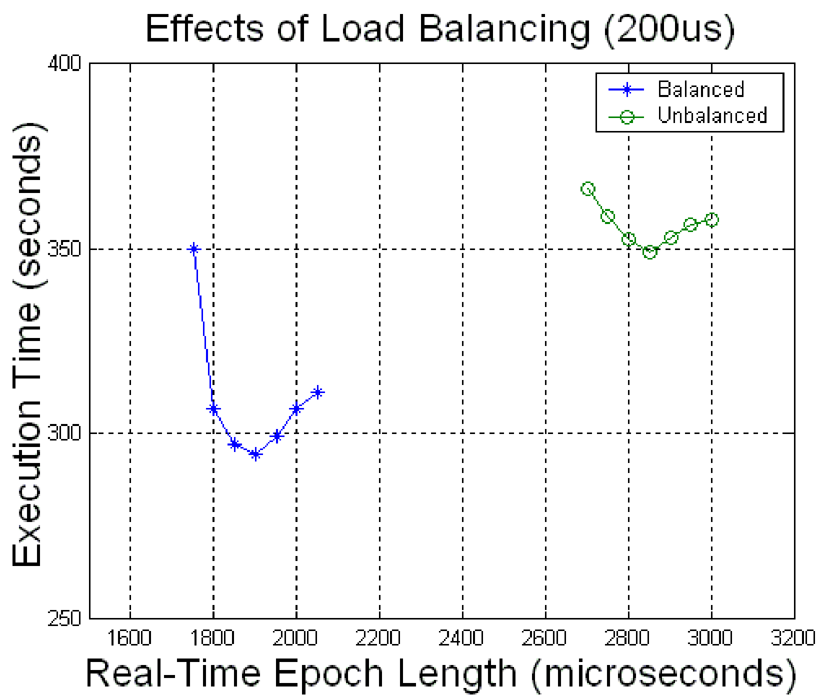
Load Balancing

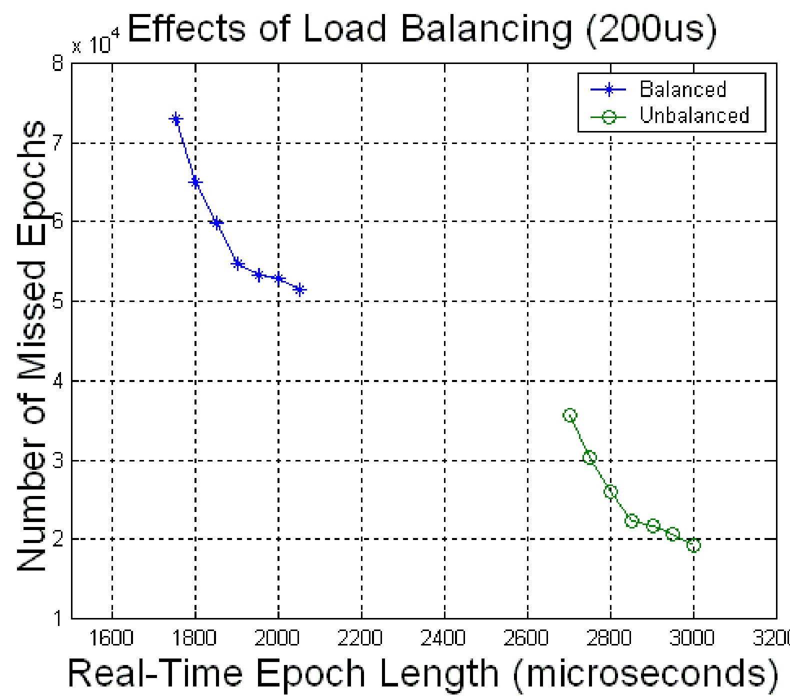




- Balancing load achieves a smaller minimum execution time
  - Occurs at a different number of missed epochs in each topology
- Missing a greater number of smaller epochs can reduce total execution time

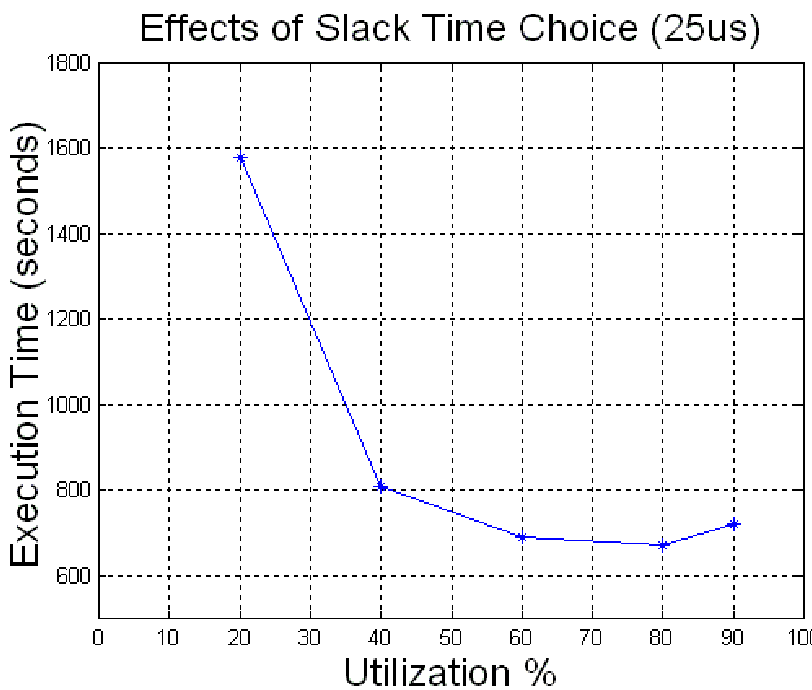
Load Balancing

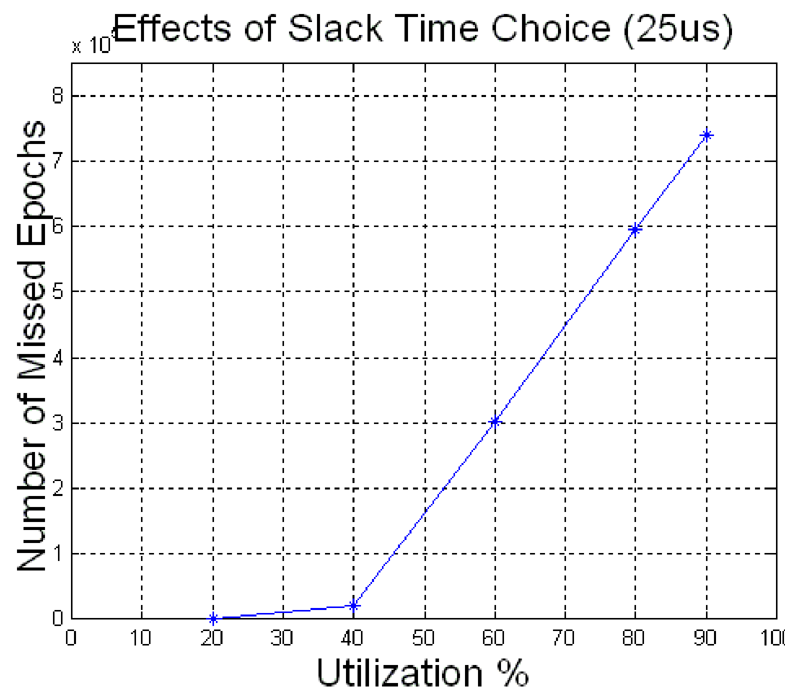




- if ((? ET * N) > (? ME * ET)) ? Improvement
  - ? ET: Change in epoch time, N: Total number of epochs, ? ME: Change in missed epochs, ET: New epoch time
- Network dependent minimum slack time required

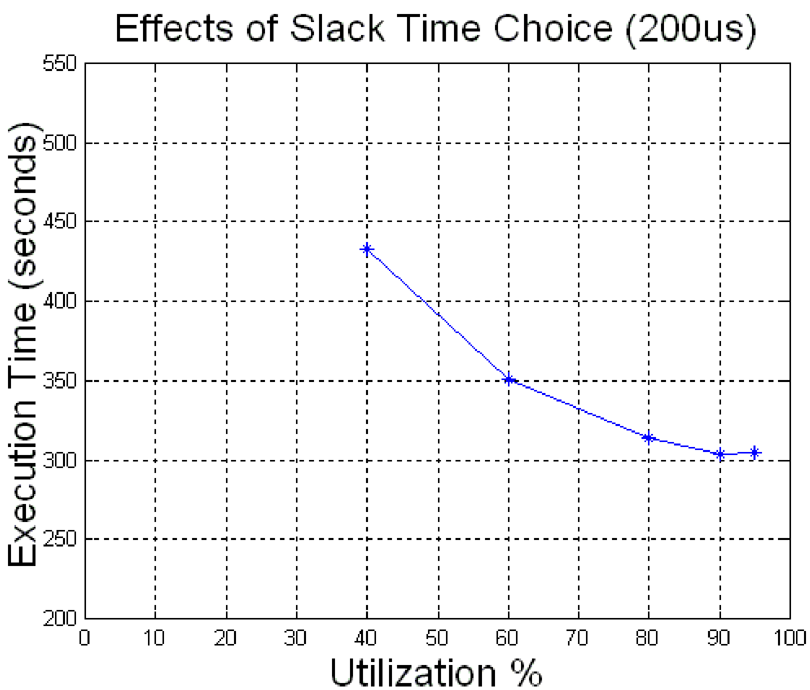
Slack Time

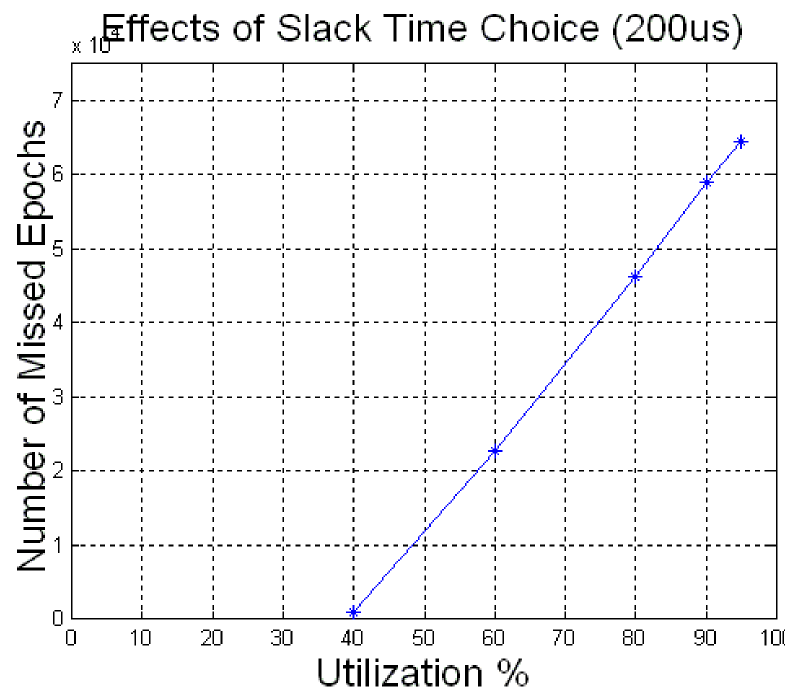




- Roughly 2% of epochs (21,084 out of 1,000,000) missed at a utilization of 40% and a granularity of 800? s
  - Roughly 8% the granularity of standard Linux (10ms)
  - Within the constraints of *many* synchronous distributed applications

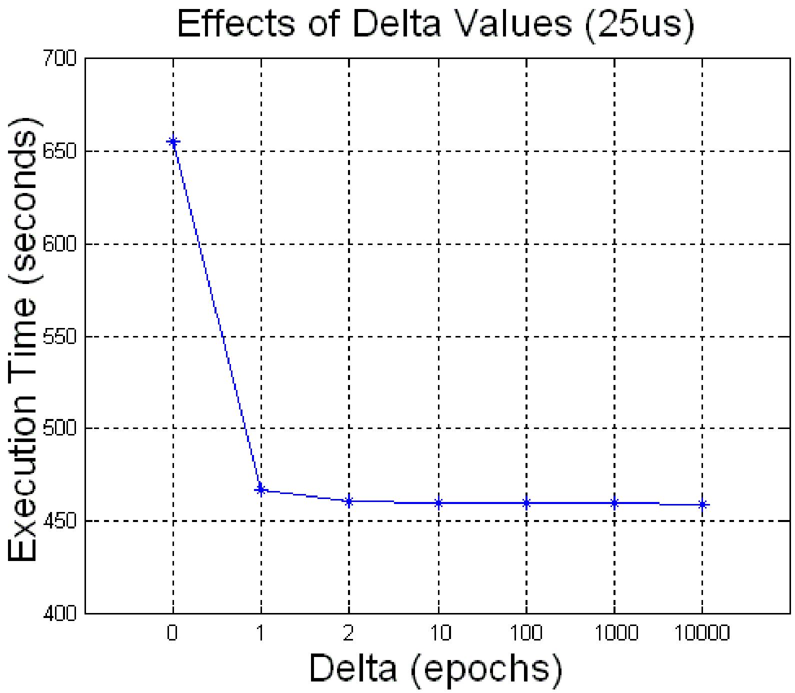
Slack Time

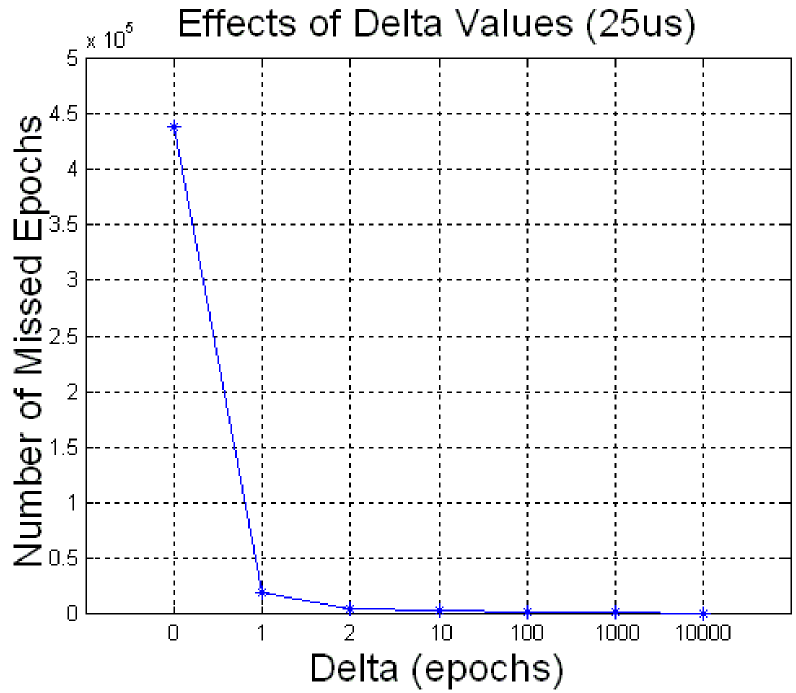




- Minimum slack time in the neighborhood of 150? s required
  - Dependent on network latency and its associated variation
  - Dependent on the ISRs (frequency and associated length)
  - Dependent on clock synchronization in absolute time

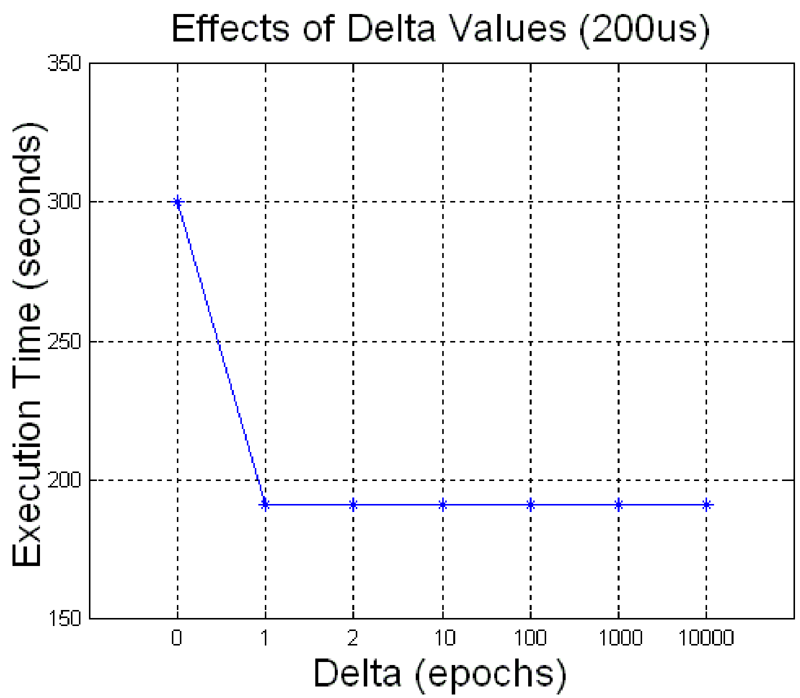
Delta Values

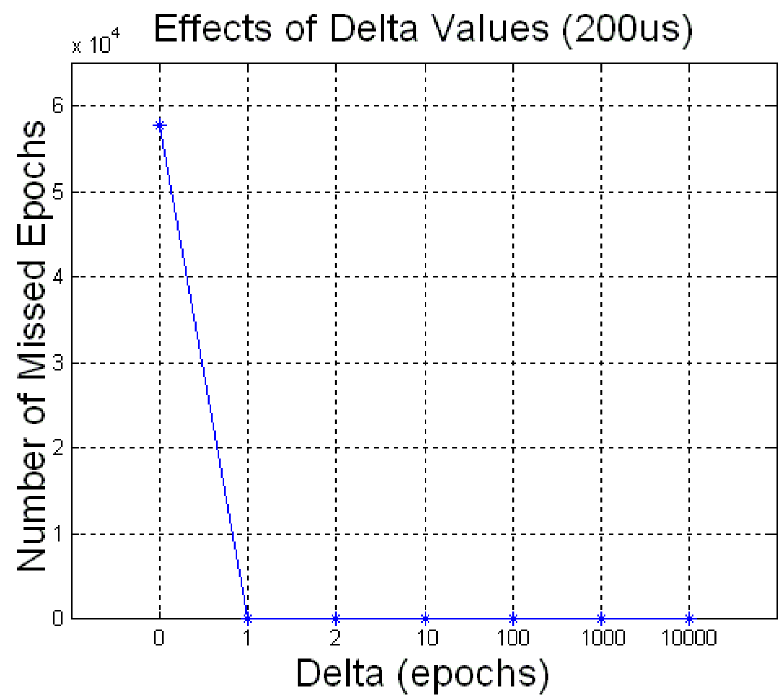




- Allowing hosts to get out of synch by even *a single epoch* has profound effects on performance
  - ?=1 reduces the number of missed epochs from 437,763 to 19,360 (1/20) and decreases total execution time by nearly 30%

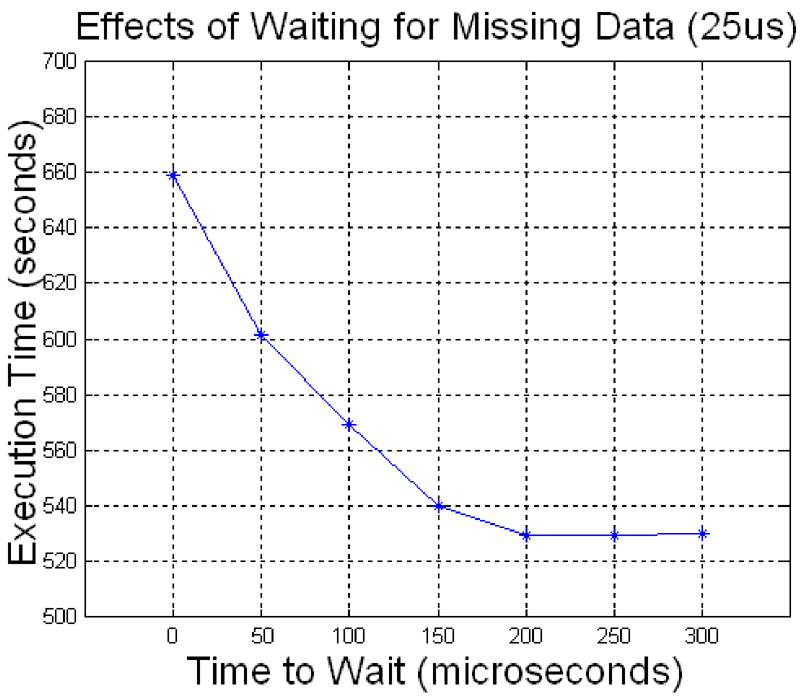
Delta Values

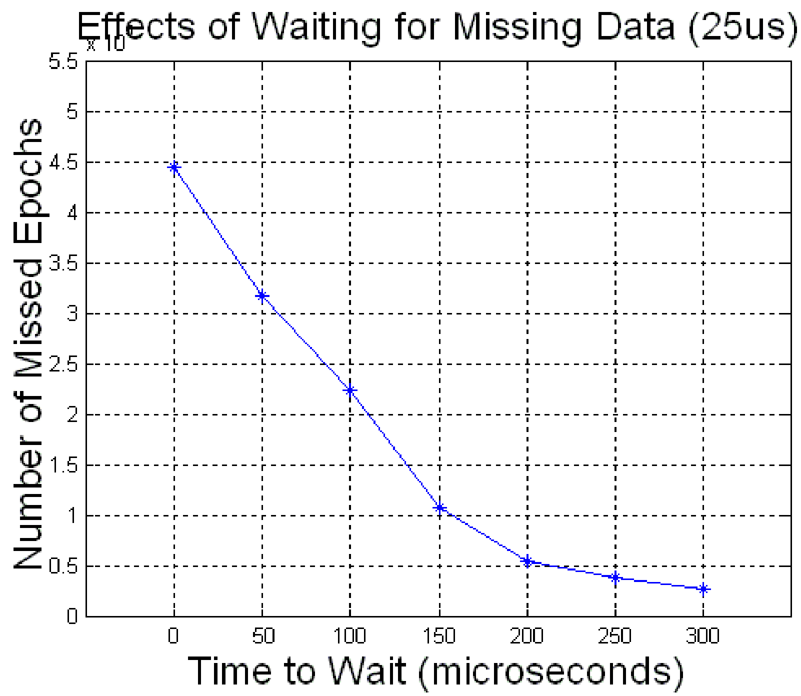




- Masks some clock synchronization issues
- In network simulations, ProTEuS can utilize virtual network delay to raise ? values without affecting simulation results
  - Data produced in epoch *N* is consumed in epoch *N+delay*

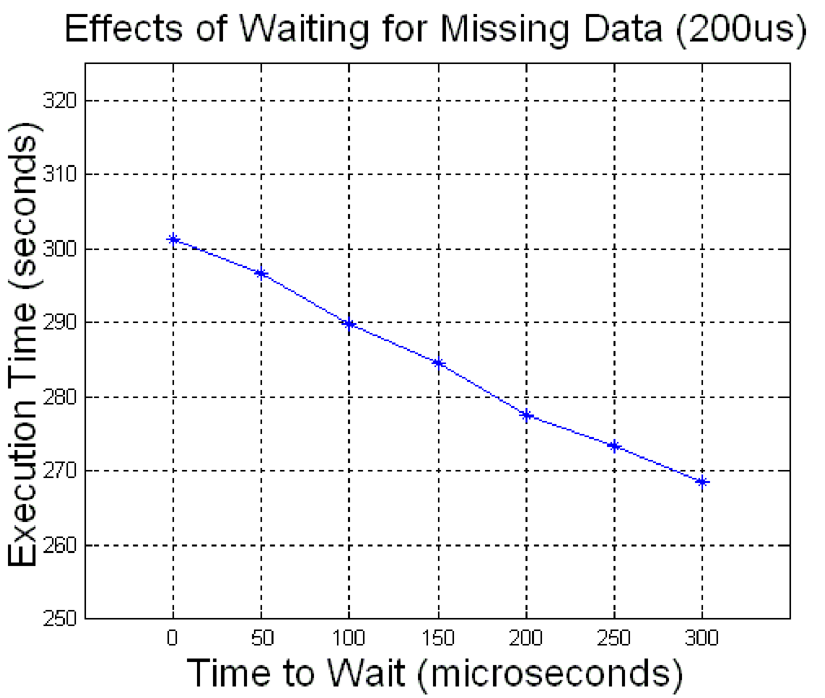
Wait Time

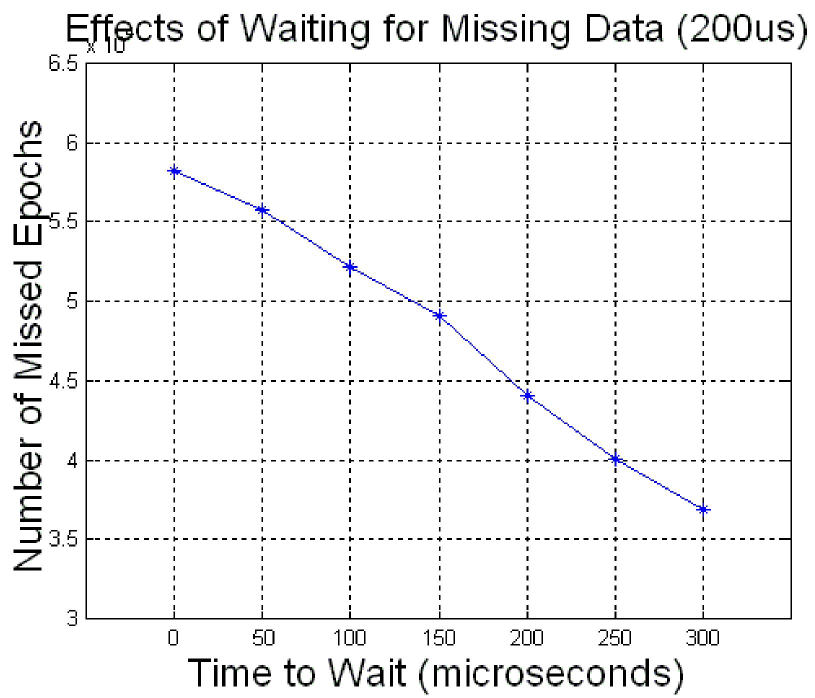




- Give data a "second chance" to arrive
  - Helps offset network delay variation and clock synchronization
- Waiting too long can cause execution to overlap into the next scheduled epoch and actually degrade performance

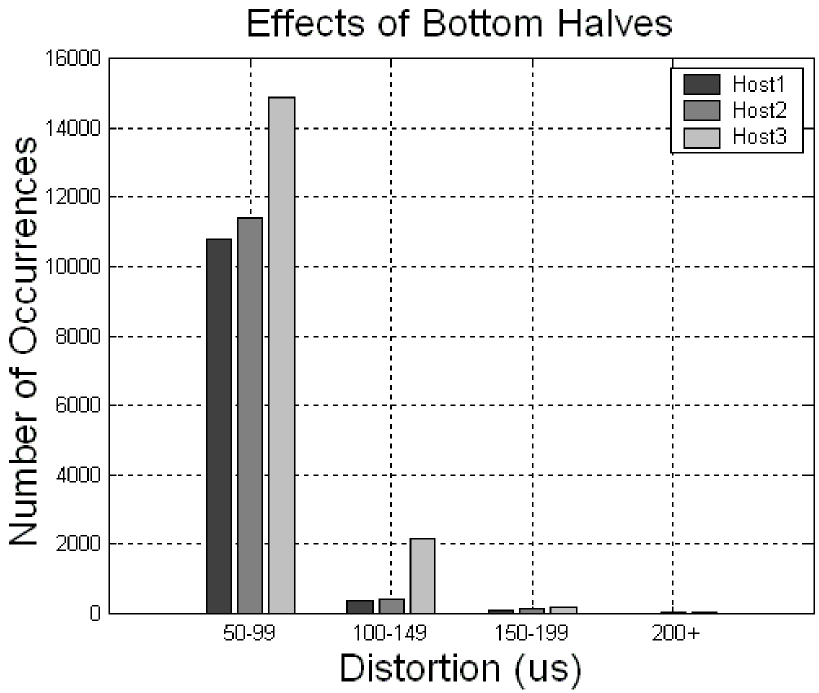
Wait Time





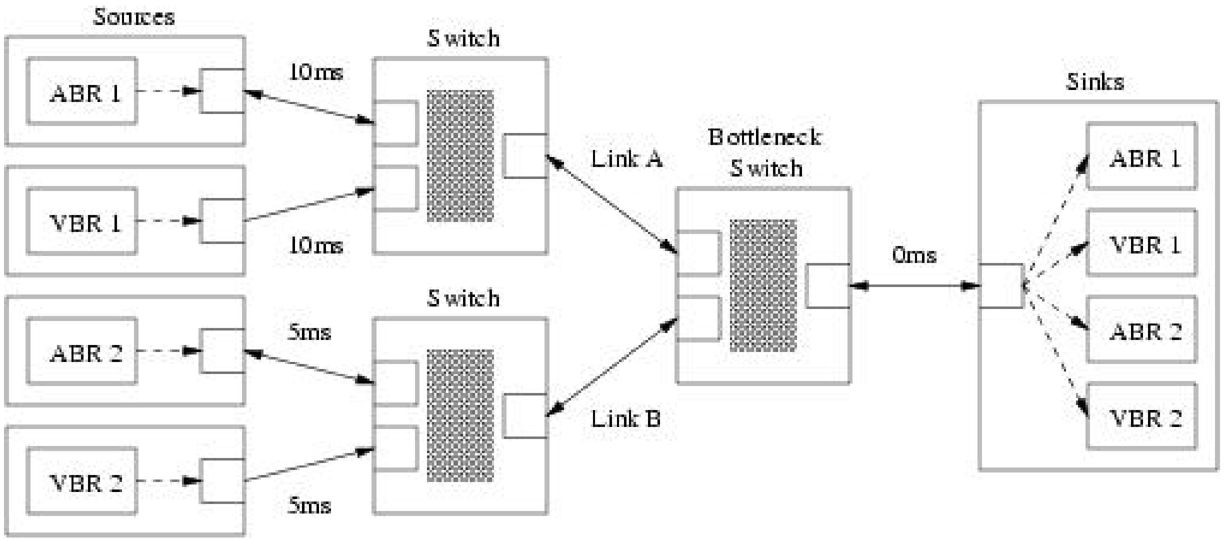
- Generally beneficial to simulation performance
  - Diminishing returns as wait time approaches the epoch length
- Although waiting appears to have beneficial effects on performance, no ProTEuS results herein utilize it

Bottom Halves



- Latency between KURT timer interrupt and the actual start of the epoch over a 60 second wall clock period
- Several hundred times per second latency approaches 100? s
  - Basically offsets the slack time of the epoch, causing epoch misses
- Not all results tabulated here – most occurrences < 50? s
- Two biggest culprits we essentially inactive
  - SCSI bottom half
  - Network bottom half (ATM does not use a bottom half)

Faithfulness



- Compare verity of BONeS, GTW and ProTEuS
  - Link utilization on links A and B
  - Mean queueing delay on each ABR stream
  - ABR queue length at the output port on the bottleneck
  - ABR source rate for ABR source 1

Link Utilization

		Link A			Link B	
Experiment	*BONeS*	*GTW*	*ProTEuS*	*BONeS*	*GTW*	*ProTEuS*
A: 5ms B: 20ms	*0.495*	*0.502*	*0.503*	*0.505*	*0.498*	*0.499*
A: 15ms B: 15ms	*0.494*	*0.498*	*0.499*	*0.506*	*0.502*	*0.501*
A: 20ms B: 5ms	*0.493*	*0.498*	*0.499*	*0.507*	*0.502*	*0.501*

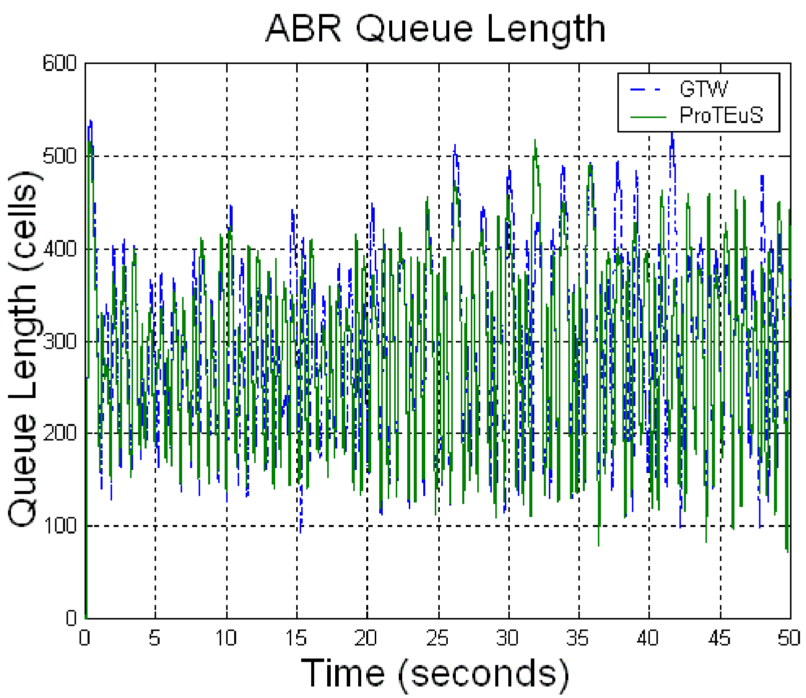
- All three simulation platforms sufficiently close
- In general, lower RTT delay on RM cell feedback should produce higher link utilizations
  - Utilization should *decrease* down the columns on Link A
  - Utilization should *increase* down the columns on Link B
  - Experiment 1 – Link A should have higher utilization
  - Experiment 2 & 3 – Link B should have higher utilization
    - Link B always exhibits higher utilization in BONeS

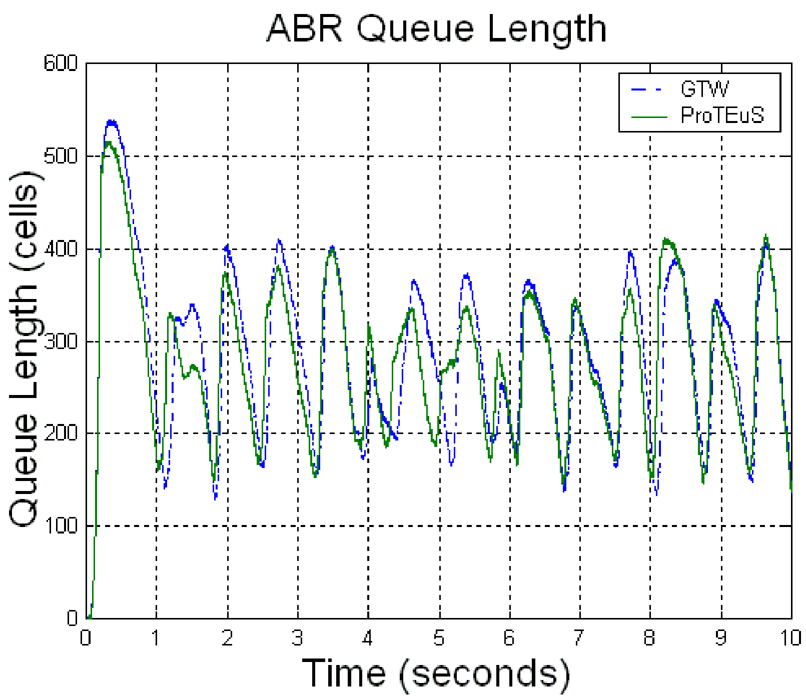
Mean Queueing Delay

	ABR 1 (seconds)			ABR 2 (seconds)		
Experiment	*BONeS*	*GTW*	*ProTEuS*	*BONeS*	*GTW*	*ProTEuS*
A: 5ms B: 20ms	0.143	0.159	0.156	0.147	0.164	0.163
A: 15ms B: 15ms	0.149	0.165	0.163	0.148	0.161	0.160
A: 20ms B: 5ms	0.154	0.167	0.165	0.147	0.159	0.157

- All three simulation platforms sufficiently close
- In general, lower RTT delay on RM cell feedback should produce lower mean queueing delays
  - Delay should *increase* down the columns on ABR 1
  - Delay should *decrease* down the columns on ABR 2
  - Experiment 1 – ABR 1 should have lower delay
  - Experiment 2 & 3 – ABR 2 should have lower delay
    - Delay gap isn't growing in both directions in BONeS experiments

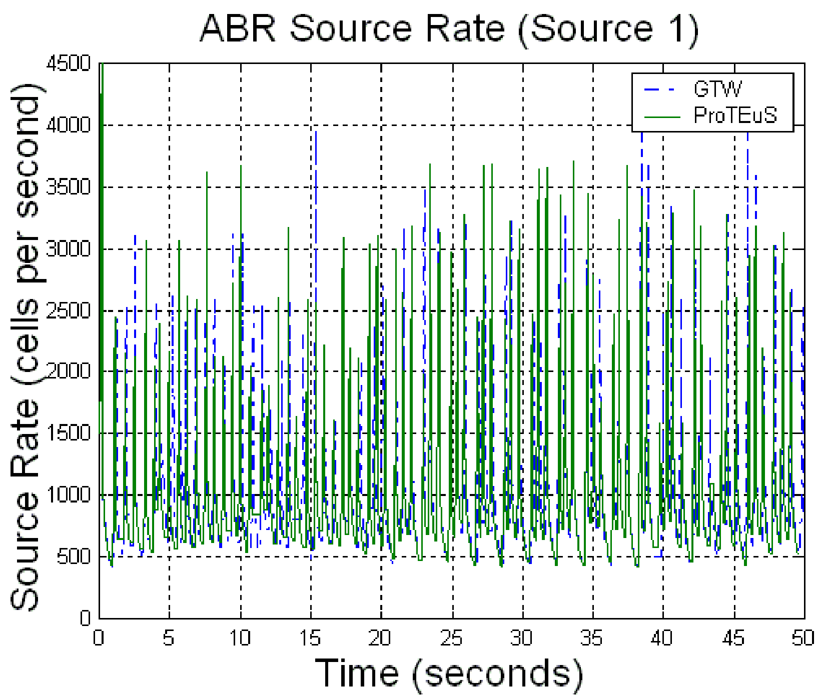
ABR Queue Length

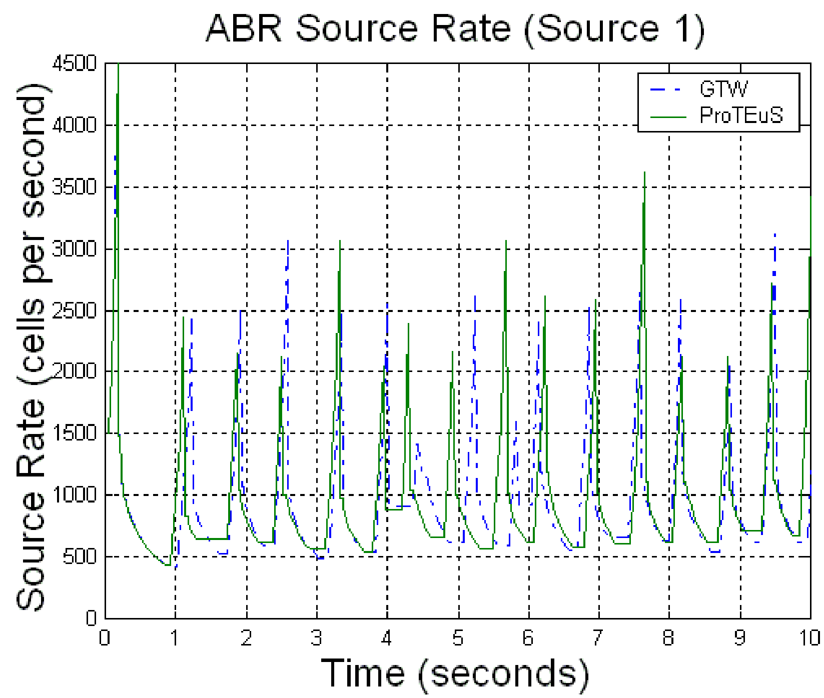




- Both GTW and ProTEuS produce *very* similar results
- Definite differences, but they are essentially insignificant
  - Small differences around 1.5s and 5s, but
  - Virtually indistinguishable during ramp-up and between 2s and 4s

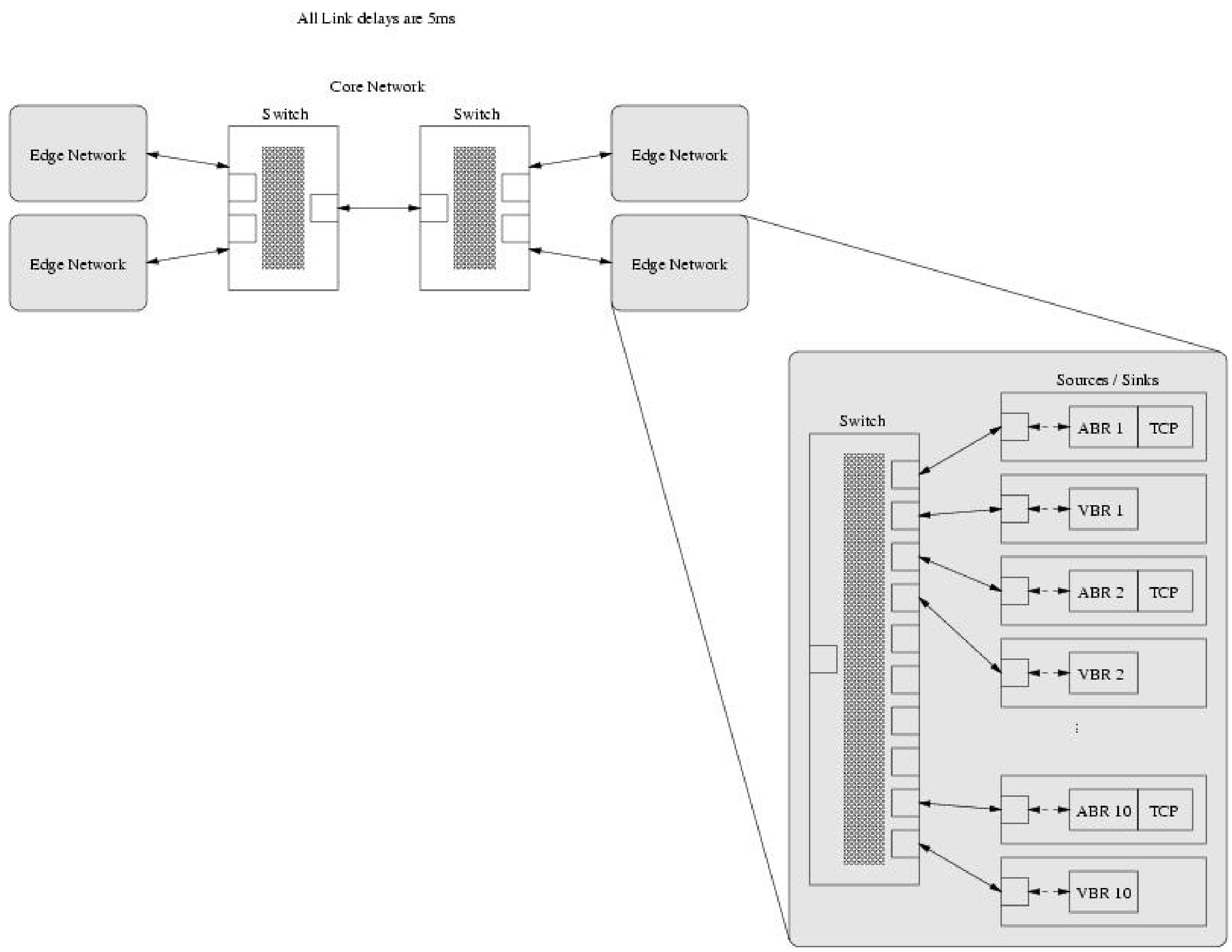
ABR Source Rate





- Very similar, largely indistinguishable, results
- Ramp-up period is particularly close
- Discrepancy between 4.5s and 6s certainly deserves due attention, but is not significant enough to conclude anything

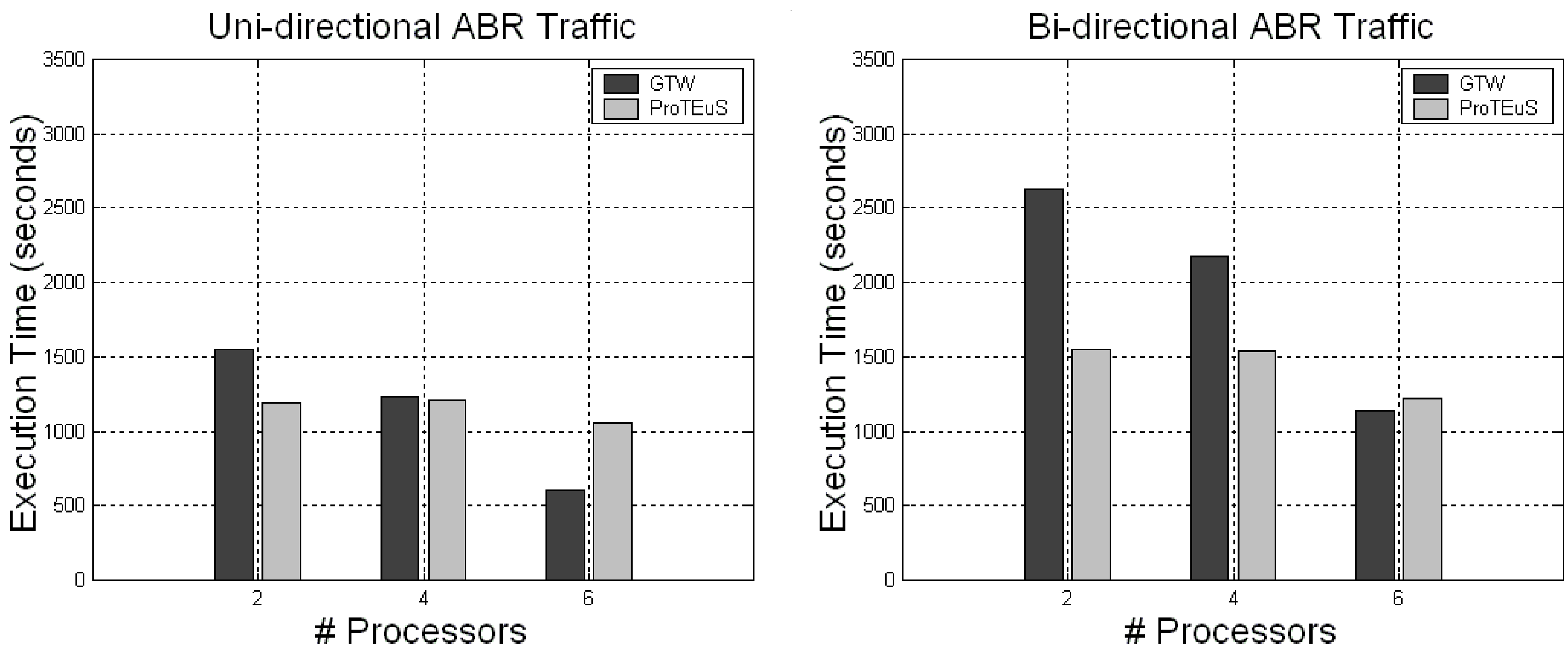
Scenario A


All Link delays are 5ms
Core Network
Switch
Switch
Edge Network
Edge Network
Edge Network
Edge Network
Sources / Sinks
Switch
ABR 1
TCP
VBR 1
ABR 2
TCP
VBR 2
ABR 10
TCP
VBR 10

Scenario A

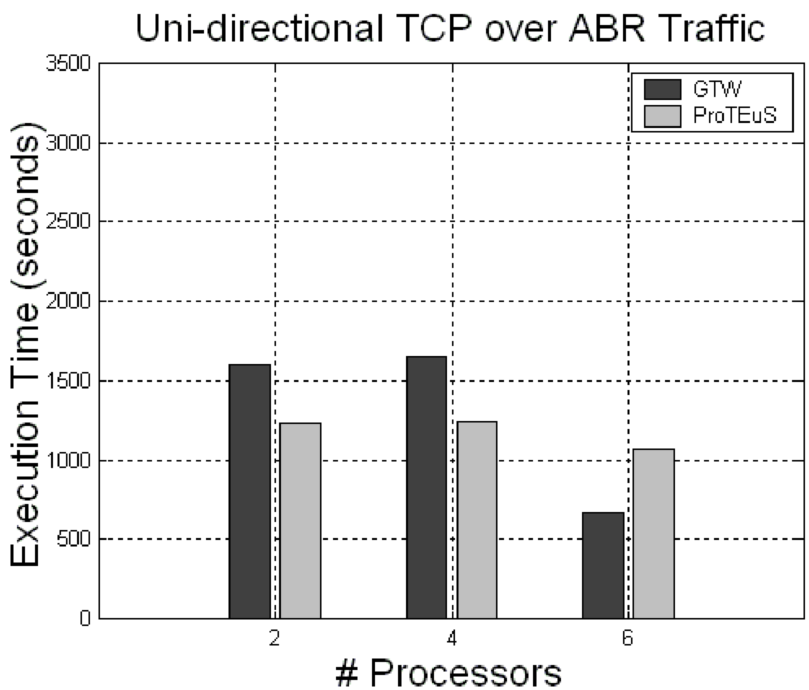
- 6 switch, 40 host edge-core ATM network
- VBR and ABR traffic
  - Uni-directional ABR
  - Bi-directional ABR
  - Uni-directional TCP over ABR
  - Bi-directional TCP over ABR
- Fill the pipe at the bottleneck without causing congestion
  - Serves to keep the event load high
- Use three virtual to physical mappings – 2, 4 & 6 processors
- VBR sources are 50% duty-cycle square waves
- ABR/TCP sources are greedy
- Run for 10 simulated seconds

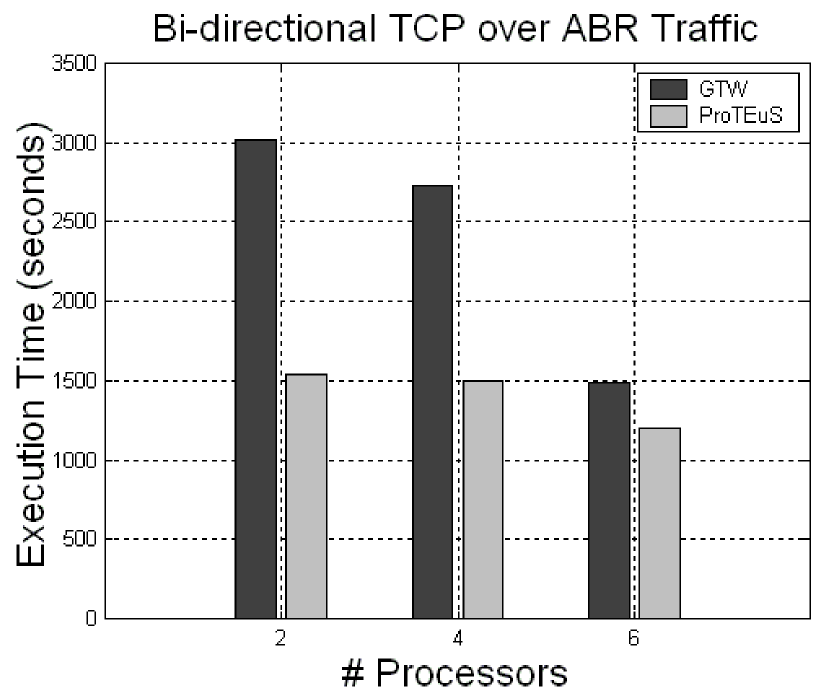
Scenario A



- ProTEuS is achieving essentially *no* speedup
  - Depends on the virtual to physical mappings
  - "Floor function" prevents the real-time period from getting any smaller, even though the computation interval of the epoch decreases

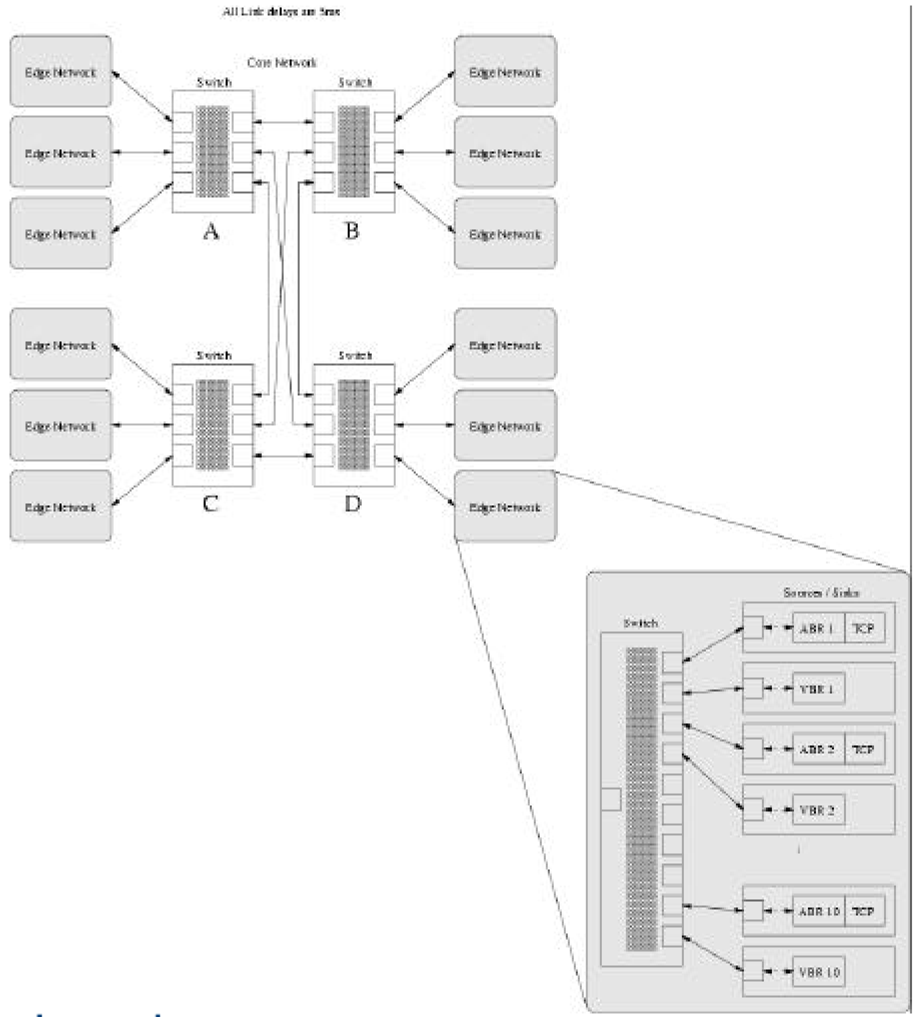
Scenario A





- Doubling the load roughly doubles the GTW execution time
  - ProTEuS takes only a marginal performance penalty
- TCP has a non-trivial impact on GTW, but not ProTEuS
  - In GTW, TCP increases the state-saving overhead

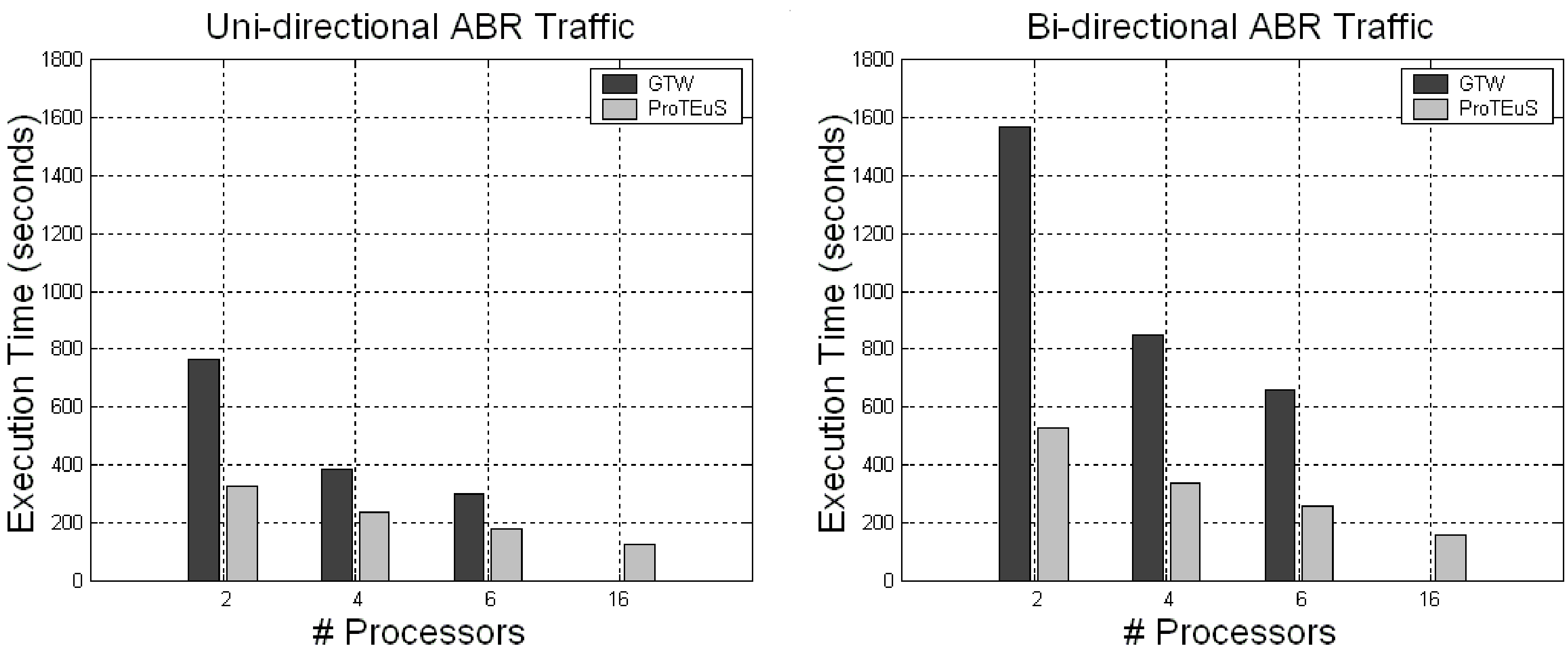
Scenario B


Core Network
Switch
Edge Network
A
B
C
D
Sources / Sinks
ABR 1
TCP
VBR 1
ABR 2
VBR 2
ABR 10
VBR 10

Scenario B

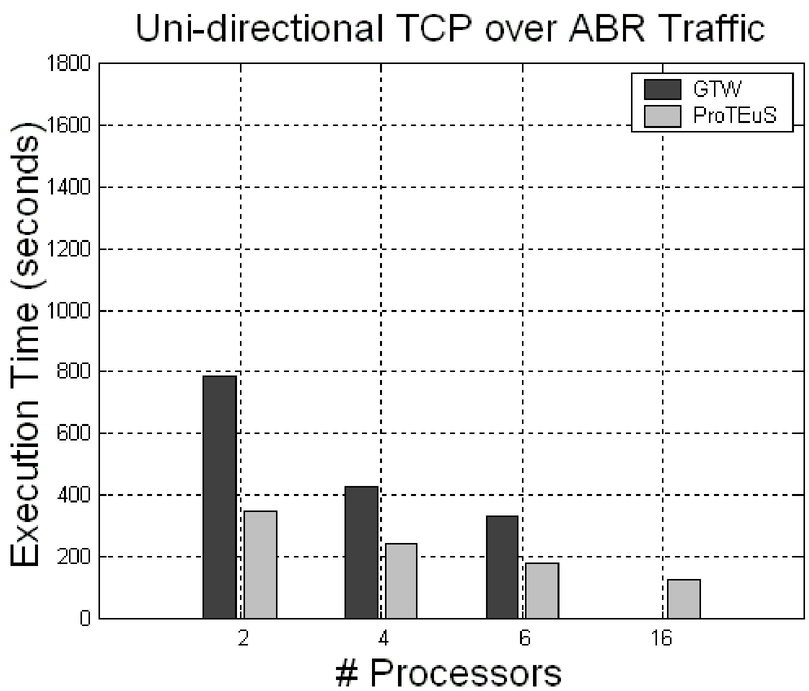
- 16 switch, 120 host edge-core ATM network
- VBR and ABR/TCP traffic
  - Same four traffic pattern scenarios
  - Routing changes
    - Edge networks are peered together
    - Exchange traffic exclusively with their peer
- Traffic parameters (PCR, ICR, TCP window, etc.) changed to reflect the new topology
  - Preserves high utilizations on bottleneck links
- Use three virtual to physical mappings –2, 4 & 6 processors
  - ProTEuS utilizes a fourth mapping across 16 processors
- Run for 1 simulated second
  - GTW unable to simulate all scenarios for a full ten seconds

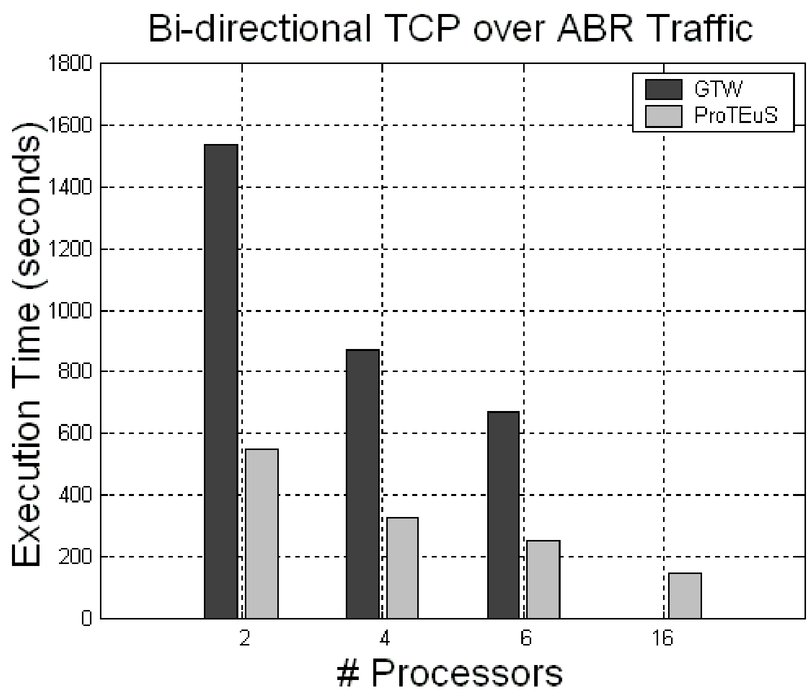
Scenario B



- ProTEuS now achieves scaling, even across 16 processors
  - ProTEuS performing better the more heavily loaded it becomes
- GTW doubles its execution time when load doubles
  - GTW performance tightly coupled to the event rate

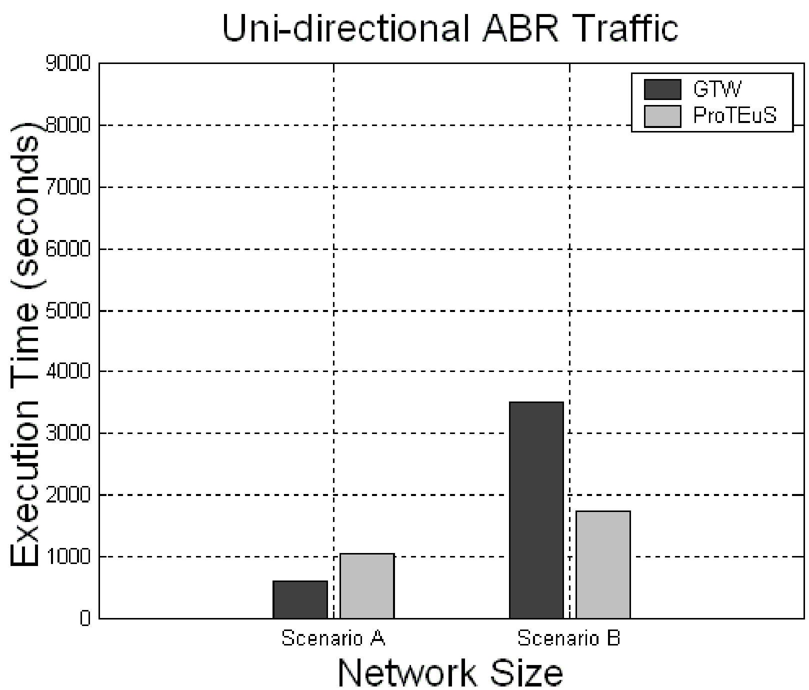
Scenario B

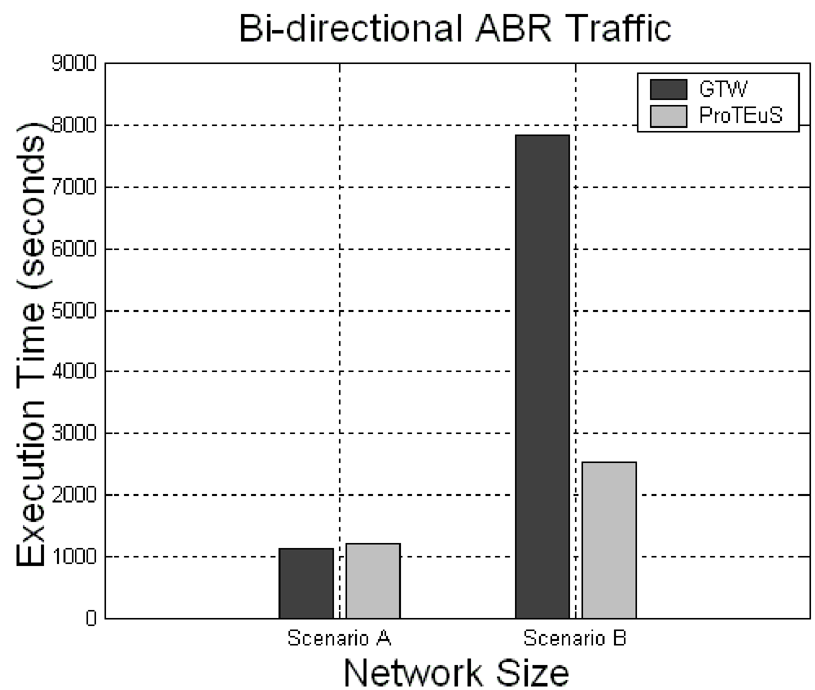




- Difficult to make heads-on comparisons in simulation time because of hardware differences
- In bi-directional cases, however, ProTEuS execution time is less than half of GTW across *all* mappings

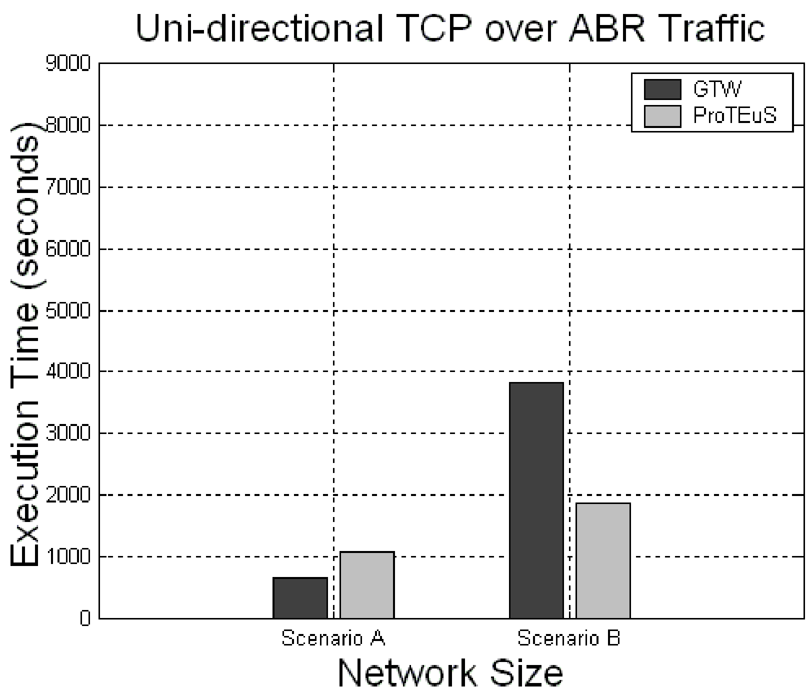
Scaling with Network Size

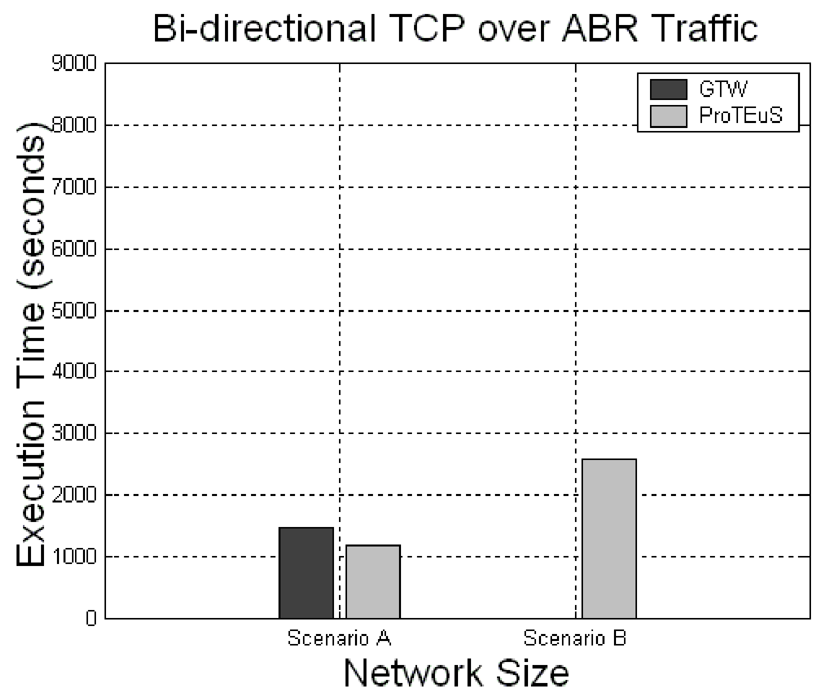




- Scenario A vs. Scenario B
  - Scenario B is roughly 3 times larger than Scenario A
- 6 processor virtual to physical mapping only
- 10 simulated seconds (Scenario B originally 1 second)

Scaling with Network Size



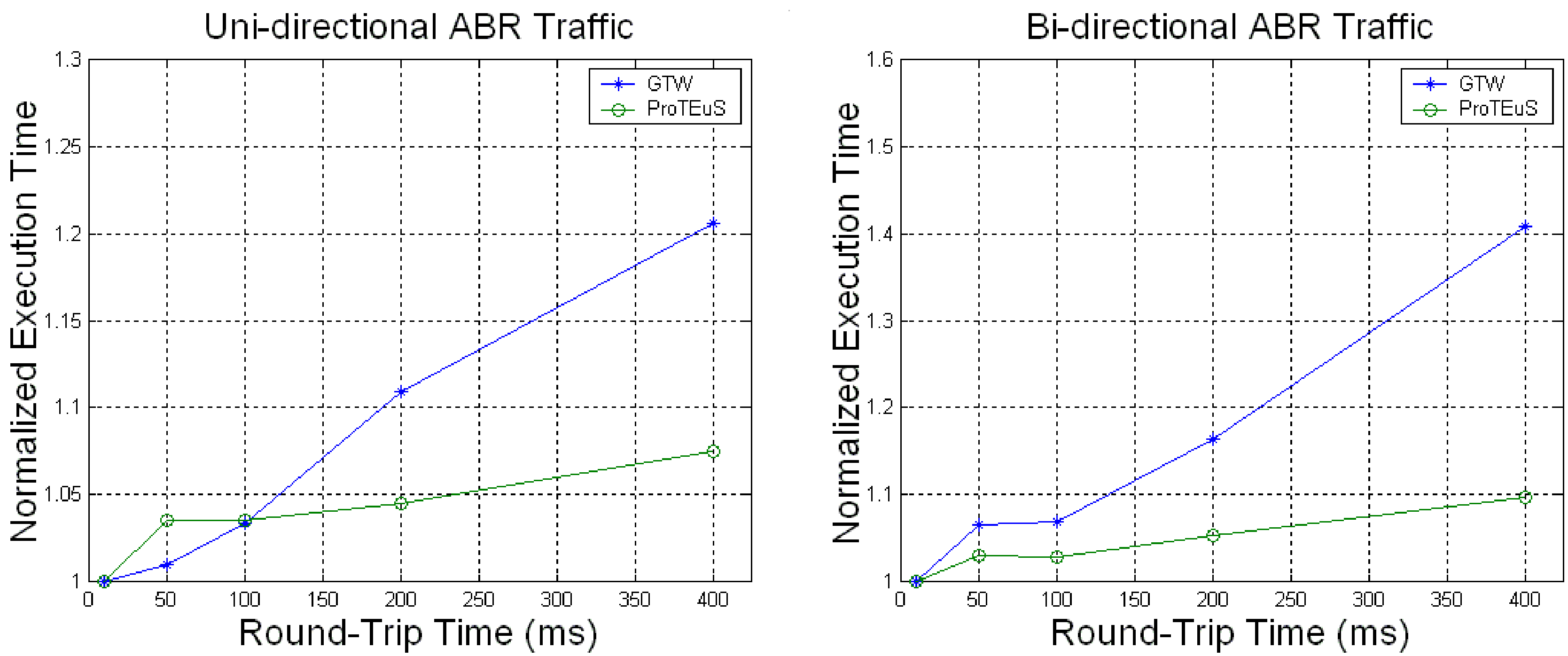


- Very clear that ProTEuS exhibits superior scaling tendencies *for these models*
- GTW unable to complete the 10 second bi-directional TCP experiment due to insufficient memory (1GB)

Scaling with Simulated Time

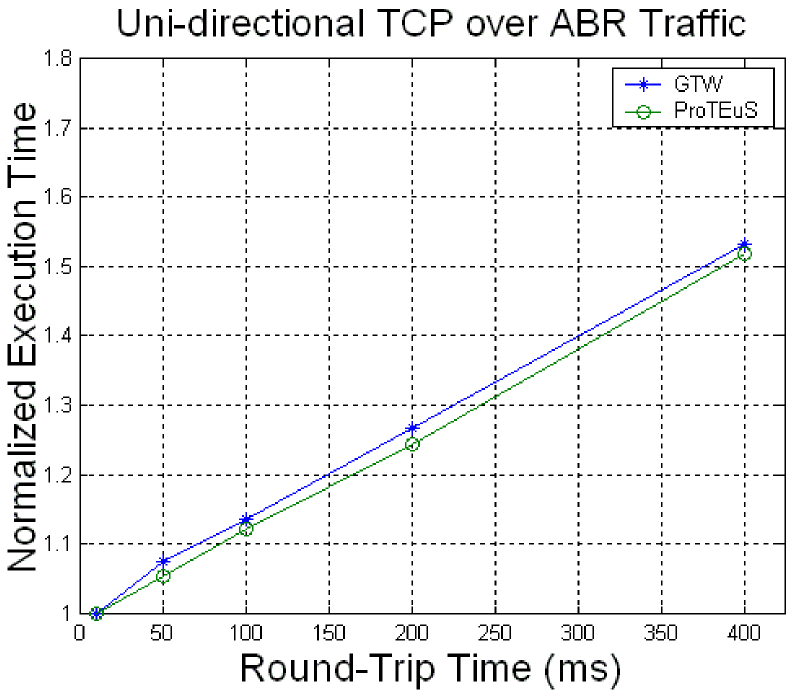
- E.g., uni-directional ABR experiment
  - 6 processor virtual to physical mapping
- From Scenario B results
  - GTW: 1 second simulation = 298.47 seconds
  - ProTEuS: 1 second simulation = 178.87 seconds
- From Network Size results
  - GTW: 10 second simulation = 3520.28 seconds
  - ProTEuS: 10 second simulation = 1754.40 seconds
- GTW scales super-linearly, ProTEuS scales linearly
  - GTW execution time increases by a factor of nearly 12
  - ProTEuS increases by slightly less than a factor of 10

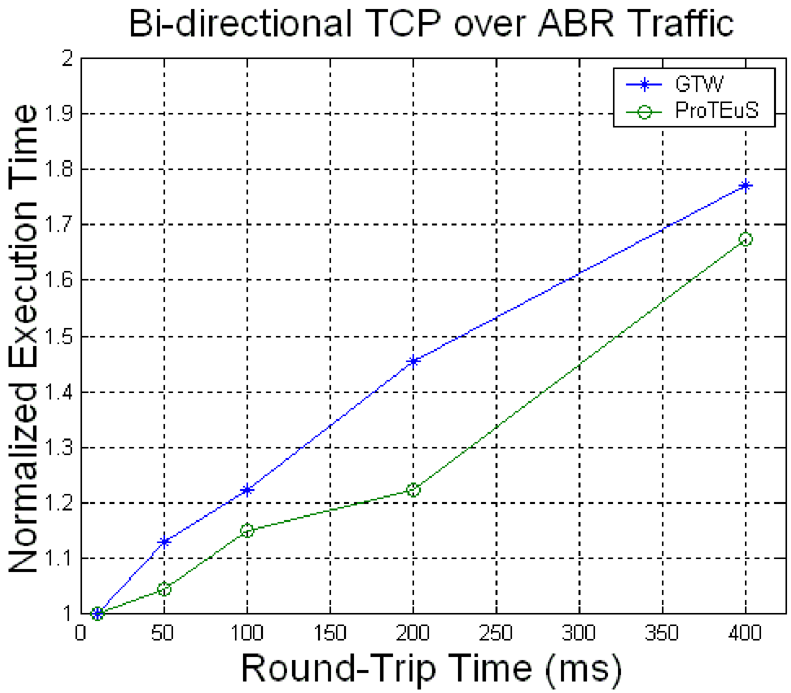
Scaling with Round-Trip Time



- Scenario A, 6 processor virtual to physical mapping
- Vary the round-trip time on the network from 10ms to 400ms
- Simulation stops when a specified number of cells are sent by each ABR/TCP source

Scaling with Round-Trip Time





- ABR scenarios show ProTEuS has little dependency on RTT, while GTW is clearly dependent
  - State-saving overhead increases
- It is the simulated system that depends on RTT, not ProTEuS

Conclusions

- Efficient network simulations are an important application area
  - Need to scale in network size and simulated time
  - Faster single processors are not sufficient to offset scaling
  - Current parallelization techniques are not entirely satisfactory, and suffer performance degradation in some applications, such as ATM network simulation
  - Justify specialized system support to improve simulation performance when possible
- ProTEuS uses real-time and embedded system techniques to support parallel simulations executing in proportional time

Conclusions

- NOW-based ProTEuS shows certain potential
  - Able to produce results largely analogous (arguably identical) to those of a conventional discrete event simulation platform
  - Compares favorably to SMMP based GTW and shows superior scaling tendencies in both network size and simulated time – at least for the applications herein
  - Does so without the loss of generality or verity and without the prohibitive hardware costs of other systems
- Other application areas may benefit from the same Proportional Time approach to synchronous distributed computation

Future Work

- Uncouple the ATM-specific implementation
  - QoS layer for ATM traffic shaping and QoS
  - PT layer for generic proportional time
- Improved control structure
  - Dynamic adjustment of epoch lengths
  - Dynamic simulation control (start, pause, stop, etc.)
- Low level master-slave clock synchronization
  - Absolute time and frequency
- Creation of advanced tools for specifying and configuring large simulation models
- Code optimizations

Future Work

- Take control of scheduling jitter
  - KURT scheduling of Linux bottom halves
  - Minimize interrupt service routine execution
- Application to other distributed applications
  - Proportional Time simulation of IP networks
  - Distributed virtual environments
- Virtualization of more of the system timeline
  - Process run frequency and copy to/from user space
- ATM specific enhancements
  - Supporting mismatched line rates
  - ATM Software switch traffic policing