High-Performance Data Transport for Grid Applications

T. Kelly, University of Cambridge, UK
S. Ravot, Caltech, USA
J.P. Martin-Flatin, CERN, Switzerland

TERENA Networking Conference, Zagreb, Croatia, 21 May 2003

Outline

- Overview of DataTAG project
- Problems with TCP in data-intensive Grids
  - Problem statement
  - Analysis and characterization
- Solutions:
  - Scalable TCP
  - GridDT
- Future Work

Overview of DataTAG Project

Member Organizations

DataTAG

CERN

INFN

ISTITUTO NAZIONALE DI FISICA NUCLEARE

INRIA

PPARC

UNIVERSITEIT VAN AMSTERDAM

http://www.datatag.org/

Project Objectives

- Build a testbed to experiment with massive file transfers (TBytes) across the Atlantic
- Provide high-performance protocols for gigabit networks underlying data-intensive Grids
- Guarantee interoperability between major HEP Grid projects in Europe and the USA

Testbed: Objectives

- Provisioning of 2.5 Gbit/s transatlantic circuit between CERN (Geneva) and StarLight (Chicago)
- Dedicated to research (no production traffic)
- Multi-vendor testbed with layer-2 and layer-3 capabilities:
  - Cisco, Juniper, Alcatel, Extreme Networks
- Get hands-on experience with the operation of gigabit networks:
  - Stability and reliability of hardware and software
  - Interoperability

Testbed: Description

- Operational since Aug 2002
- Provisioned by Deutsche Telekom
- High-end PC servers at CERN and StarLight:
  - 4x SuperMicro 2.4 GHz dual Xeon, 2 GB memory
  - 8x SuperMicro 2.2 GHz dual Xeon, 1 GB memory
  - 24x SysKonnect SK-9843 GigE cards (2 per PC)
  - total disk space: 1.7 TBytes
  - can saturate the circuit with TCP traffic

Network Research Activities

- Enhance performance of network protocols for massive file transfers (TBytes):
  - Data-transport layer: TCP, UDP, SCTP
- QoS:
  - LBE (Scavenger)
- Bandwidth reservation:
  - AAA-based bandwidth on demand
  - Lightpaths managed as Grid resources
- Monitoring

Rest of this talk

Problems with TCP in Data-Intensive Grids

Problem Statement

- *End-user's perspective:*
  - Using TCP as the data-transport protocol for Grids leads to a poor bandwidth utilization in fast WANs:
    - e.g., see demos at iGrid 2002
- *Network protocol designer's perspective:*
  - TCP is inefficient in high bandwidth*delay networks because:
    - TCP implementations have not yet been tuned for gigabit WANs
    - TCP was not designed with gigabit WANs in mind

TCP: Implementation Problems

- TCP's current implementation in Linux kernel 2.4.20 is not optimized for gigabit WANs:
  - e.g., SACK code needs to be rewritten
- SysKonnect device driver must be modified:
  - e.g., enable interrupt coalescence to cope with ACK bursts

TCP: Design Problems

- TCP's congestion control algorithm (AIMD) is not suited to gigabit networks
- Due to TCP's limited feedback mechanisms, line errors are interpreted as congestion:
  - Bandwidth utilization is reduced when it shouldn't
- RFC 2581 (which gives the formula for increasing *cwnd*) "forgot" delayed ACKs
- TCP requires that ACKs be sent at most every second segment → ACK bursts → difficult to handle by kernel and NIC

AIMD Algorithm (1/2)

- Van Jacobson, SIGCOMM 1988
- Congestion avoidance algorithm:
  - For each ACK in an RTT without loss, increase:

$$cwnd_{i+1} = cwnd_i + \frac{1}{cwnd_i}$$

  - For each window experiencing loss, decrease:

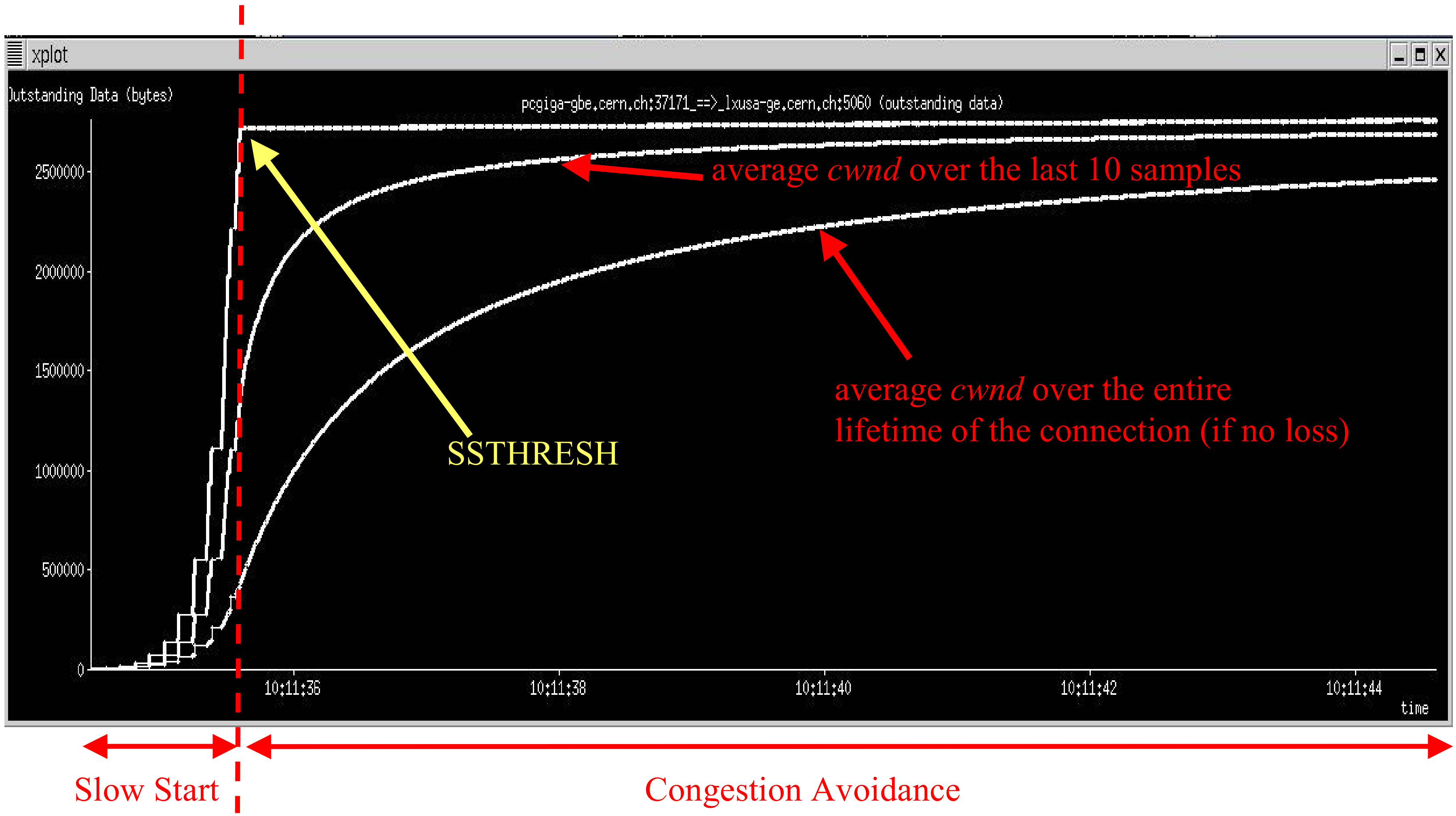
$$cwnd_{i+1} = \frac{1}{2} \times cwnd_i$$

- Slow-start algorithm:
  - Increase by one MSS per ACK until *ssthresh*

AIMD Algorithm (2/2)

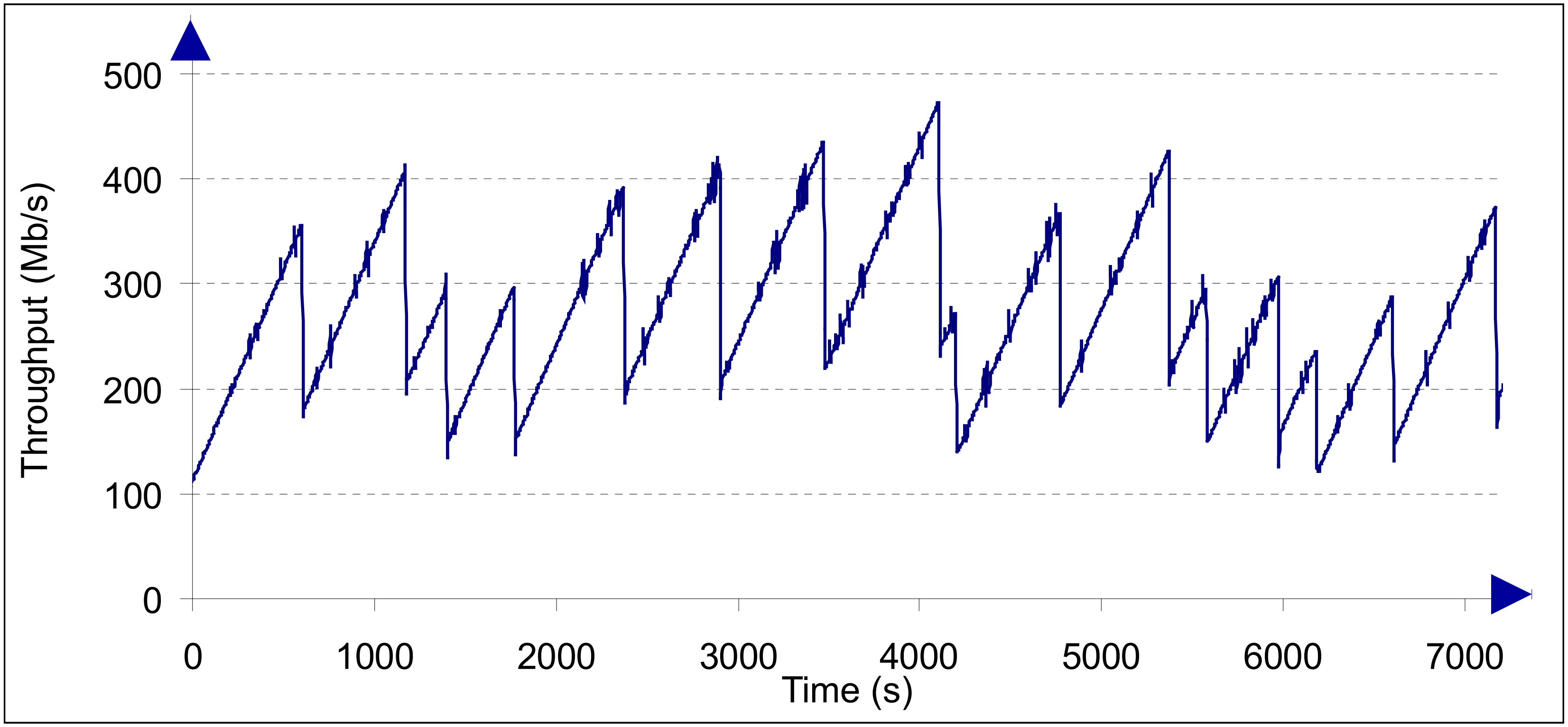
- Additive Increase:
  - A TCP connection increases slowly its bandwidth utilization in the absence of loss:
    - forever, unless we run out of send/receive buffers or detect a packet loss
    - TCP is greedy: no attempt to reach a stationary state
- Multiplicative Decrease:
  - A TCP connection reduces its bandwidth utilization drastically whenever a packet loss is detected:
    - assumption: packet loss means congestion (line errors are negligible)

Congestion Window (*cwnd*)



Disastrous Effect of Packet Loss on TCP in Fast WANs (1/2)

AIMD throughput as a function of time   C=1 Gbit/s   MSS=1,460 Bytes



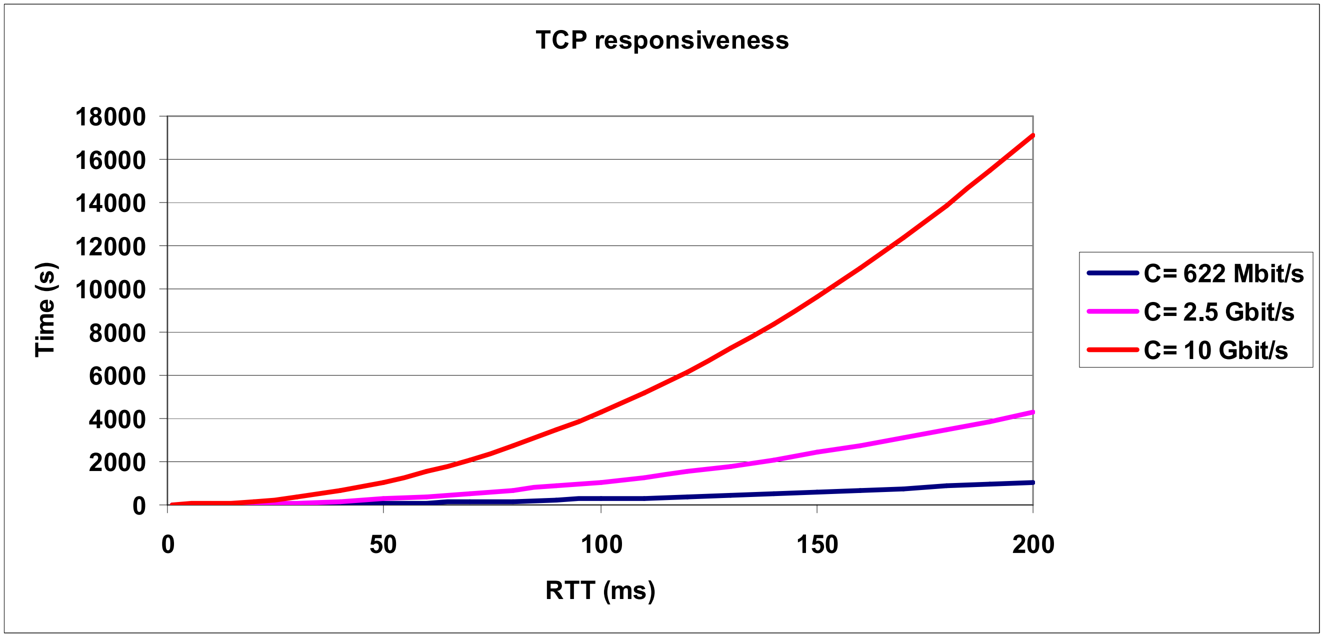
Disastrous Effect of Packet Loss on TCP in Fast WANs (2/2)

- Long time to recover from a single loss:
  - TCP should react to congestion rather than packet loss:
    - line errors and transient faults in equipment are no longer negligible in fast WANs
  - TCP should recover quicker from a loss
- TCP is more sensitive to packet loss in WANs than in LANs, particularly in fast WANs (where *cwnd* is large)

Characterization of the Problem (1/2)

The *responsiveness* ρ measures how quickly we go back to using the network link at full capacity after experiencing a loss (i.e., loss recovery time if loss occurs when bandwidth utilization = network link capacity)

$$\rho = \frac{C \cdot RTT^2}{2 \cdot inc}$$



Characterization of the Problem (2/2)

inc size = MSS = 1,460 Bytes
 # inc = window size in pkts

Capacity	RTT	# inc	Responsiveness
9.6 kbit/s (typ. WAN in 1988)	max: 40 ms	1	0.6 ms
10 Mbit/s (typ. LAN in 1988)	max: 20 ms	8	~150 ms
100 Mbit/s (typ. LAN in 2003)	max: 5 ms	20	~100 ms
622 Mbit/s	120 ms	~2,900	~6 min
2.5 Gbit/s	120 ms	~11,600	~23 min
10 Gbit/s	120 ms	~46,200	~1h 30min

Congestion vs. Line Errors

RTT=120 ms, MTU=1,500 Bytes, AIMD

Throughput	Required Bit Loss Rate	Required Packet Loss Rate
10 Mbit/s	**$2\ 10^{-8}$**	**$2\ 10^{-4}$**
100 Mbit/s	**$2\ 10^{-10}$**	**$2\ 10^{-6}$**
2.5 Gbit/s	**$3\ 10^{-13}$**	**$3\ 10^{-9}$**
10 Gbit/s	**$2\ 10^{-14}$**	**$2\ 10^{-10}$**

At gigabit speed, the loss rate required for packet loss to be ascribed only to congestion is unrealistic with AIMD

Solutions

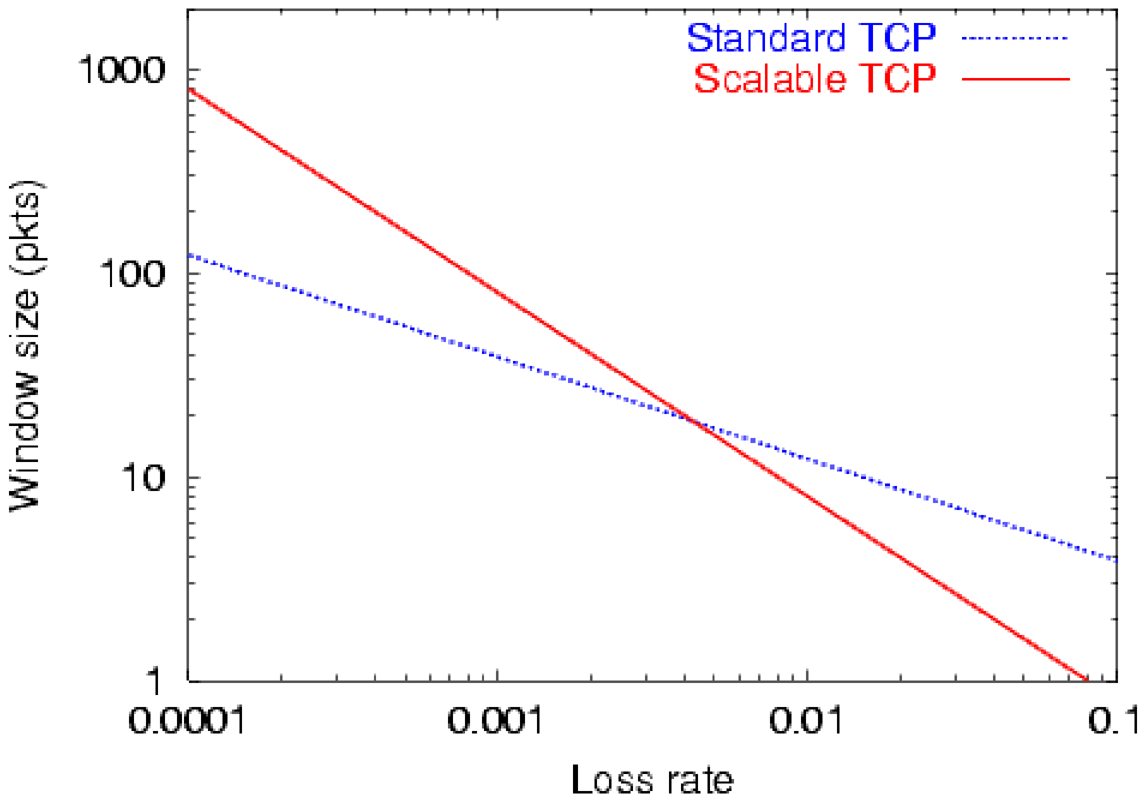
What Can We Do?

- To achieve higher throughputs over high bandwidth*delay networks, we can:
  - Change AIMD to recover faster in case of packet loss
  - Use larger MTU (Jumbo frames: 9,000 Bytes)
  - Set the initial *ssthresh* to a value better suited to the RTT and bandwidth of the TCP connection
  - Avoid losses in end hosts (implementation issue)
- Two proposals:
  - Kelly: Scalable TCP
  - Ravot: GridDT

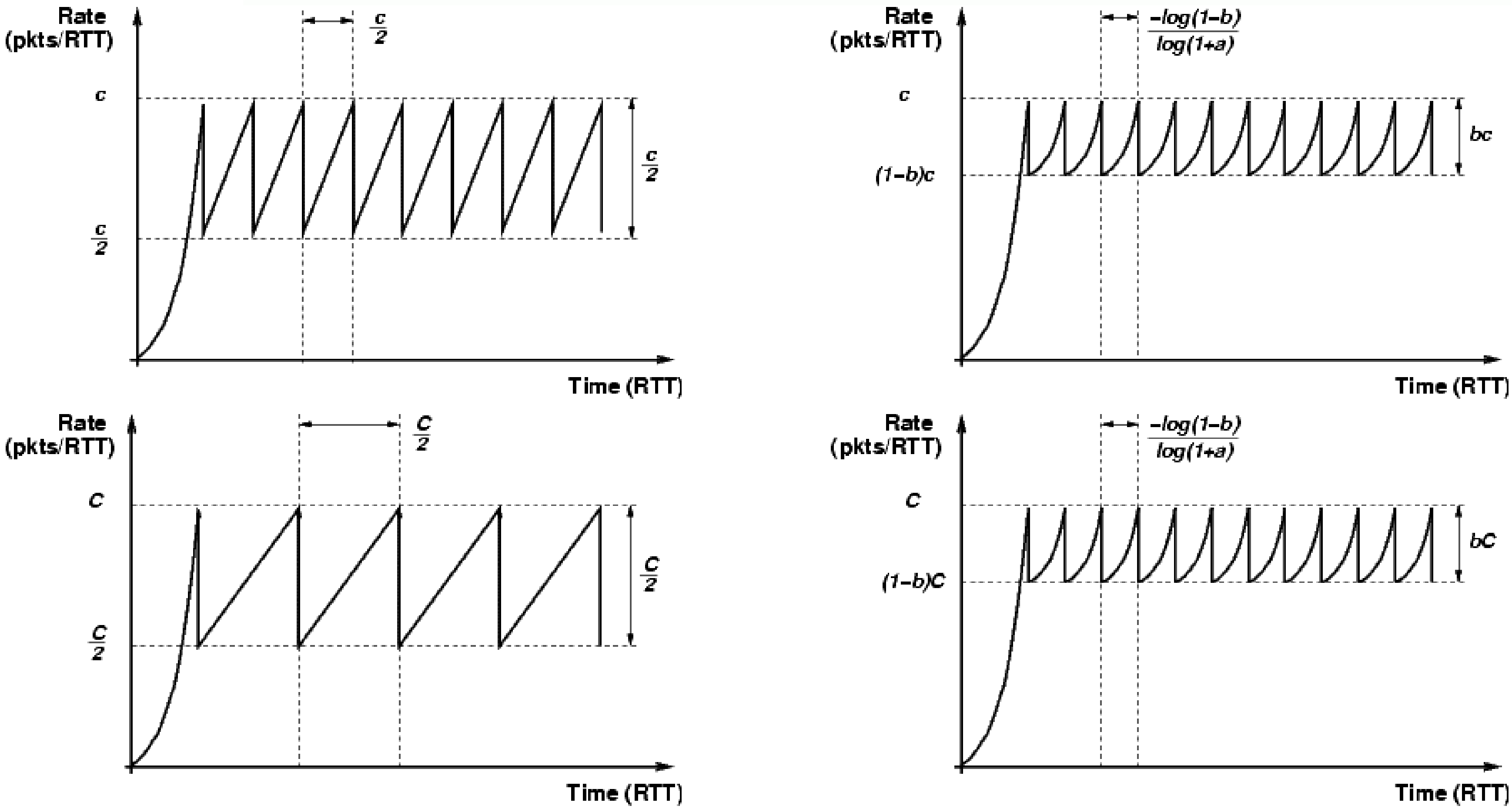
Scalable TCP: Algorithm

- For cwnd>lwnd, replace AIMD with new algorithm:
  - for each ACK in an RTT without loss:
    - $cwnd_{i+1} = cwnd_i + a$
  - for each window experiencing loss:
    - $cwnd_{i+1} = cwnd_i - (b \times cwnd_i)$
- Kelly's proposal during internship at CERN: (lwnd,a,b) = (16, 0.01, 0.125)
  - Trade-off between fairness, stability, variance and convergence
- Advantages:
  - Responsiveness improves dramatically for gigabit networks
  - Responsiveness is independent of capacity

Scalable TCP: *lwnd*



Scalable TCP: Responsiveness Independent of Capacity



Scalable TCP: Improved Responsiveness

- Responsiveness for RTT=200 ms and MSS=1,460 Bytes:
  - Scalable TCP: ~3 s
  - AIMD:
    - ~3 min at 100 Mbit/s
    - ~1h 10min at 2.5 Gbit/s
    - ~4h 45min at 10 Gbit/s
- Patch available for Linux kernel 2.4.19
- For more details, see paper and code at:
  - http://www-lce.eng.cam.ac.uk/˜ctk21/scalable/

Scalable TCP vs. AIMD: Benchmarking

Number of flows	2.4.19 TCP	2.4.19 TCP + new dev driver	Scalable TCP
1	7	16	44
2	14	39	93
4	27	60	135
8	47	86	140
16	66	106	142

Bulk throughput tests with C=2.5 Gbit/s. Flows transfer 2 GBytes and start again for 20 min.

GridDT: Algorithm

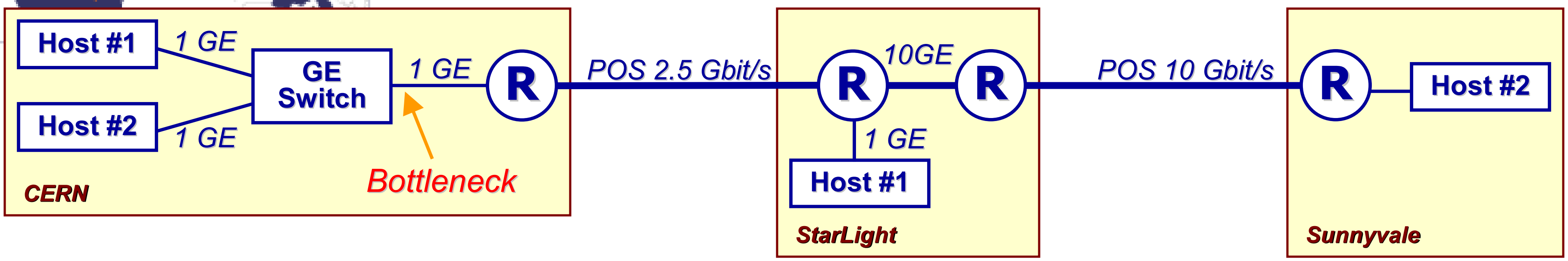
- Congestion avoidance algorithm:
  - For each ACK in an RTT without loss, increase:

$$cwnd_{i+1} = cwnd_i + \frac{A}{cwnd_i}$$

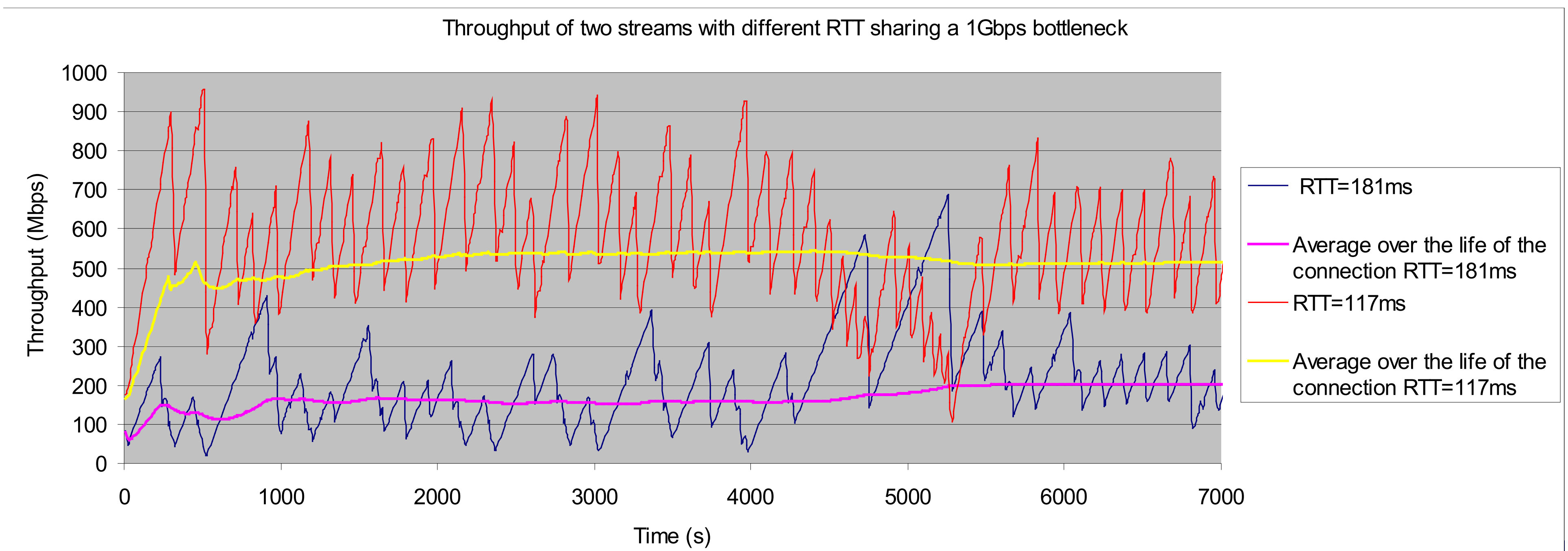
- By modifying A dynamically according to RTT, GridDT guarantees fairness among TCP connections:

$$\frac{A1}{A2} = \left(\frac{RTT_{A1}}{RTT_{A2}}\right)^2$$

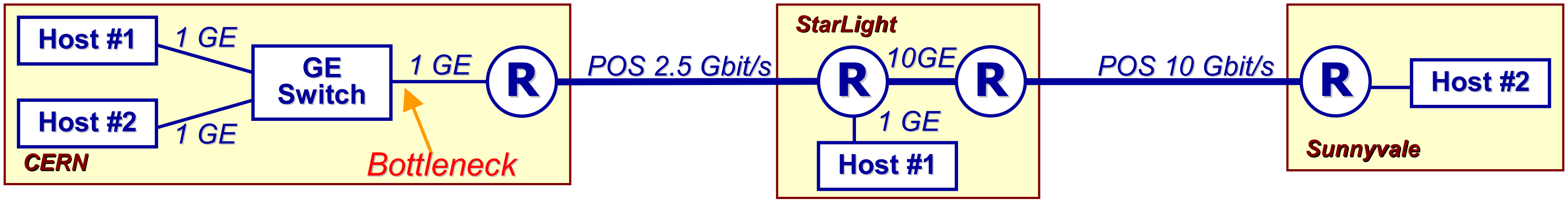
AIMD: RTT Bias



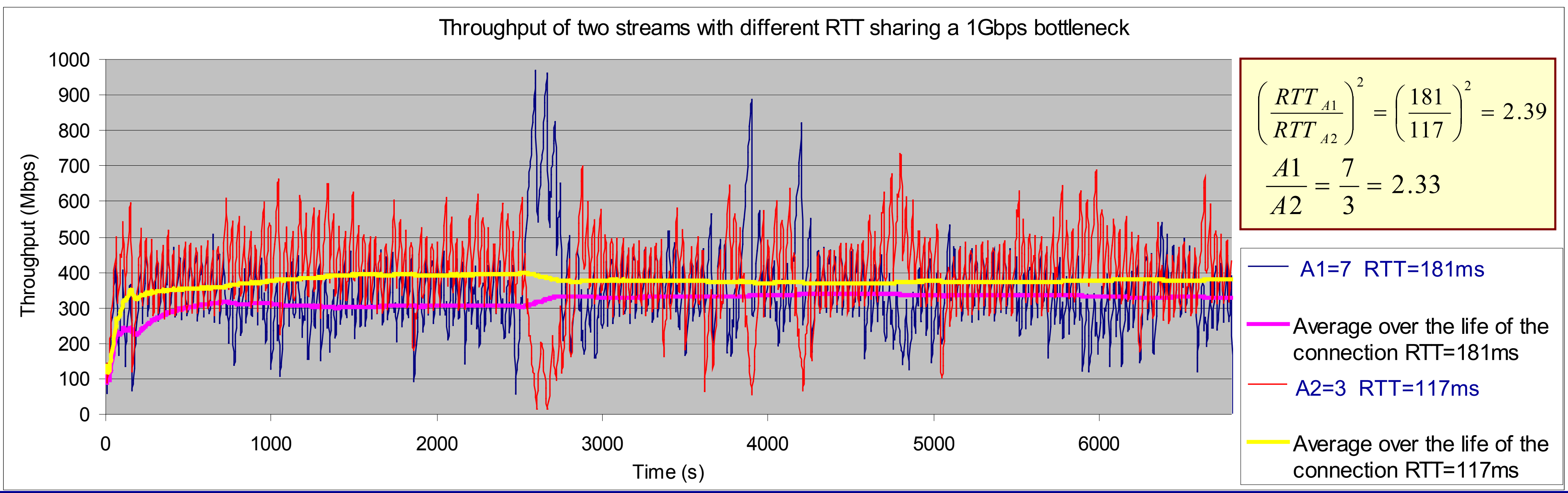
- Two TCP streams share a 1 Gbit/s bottleneck
- CERN-Sunnyvale: RTT=181ms. Avg. throughput over a period of 7,000s = 202 Mbit/s
- CERN-StarLight: RTT=117ms. Avg. throughput over a period of 7,000s = 514 Mbit/s
- MTU = 9,000 Bytes. Link utilization = 72%



GridDT Fairer than AIMD



- CERN-Sunnyvale: RTT = 181 ms. Additive inc. A1 = 7. Avg. throughput = 330 Mbit/s
- CERN-StarLight: RTT = 117 ms. Additive inc. A2 = 3. Avg. throughput = 388 Mbit/s
- MTU = 9,000 Bytes. Link utilization 72%



$$\left(\frac{RTT_{A1}}{RTT_{A2}}\right)^2 = \left(\frac{181}{117}\right)^2 = 2.39$$

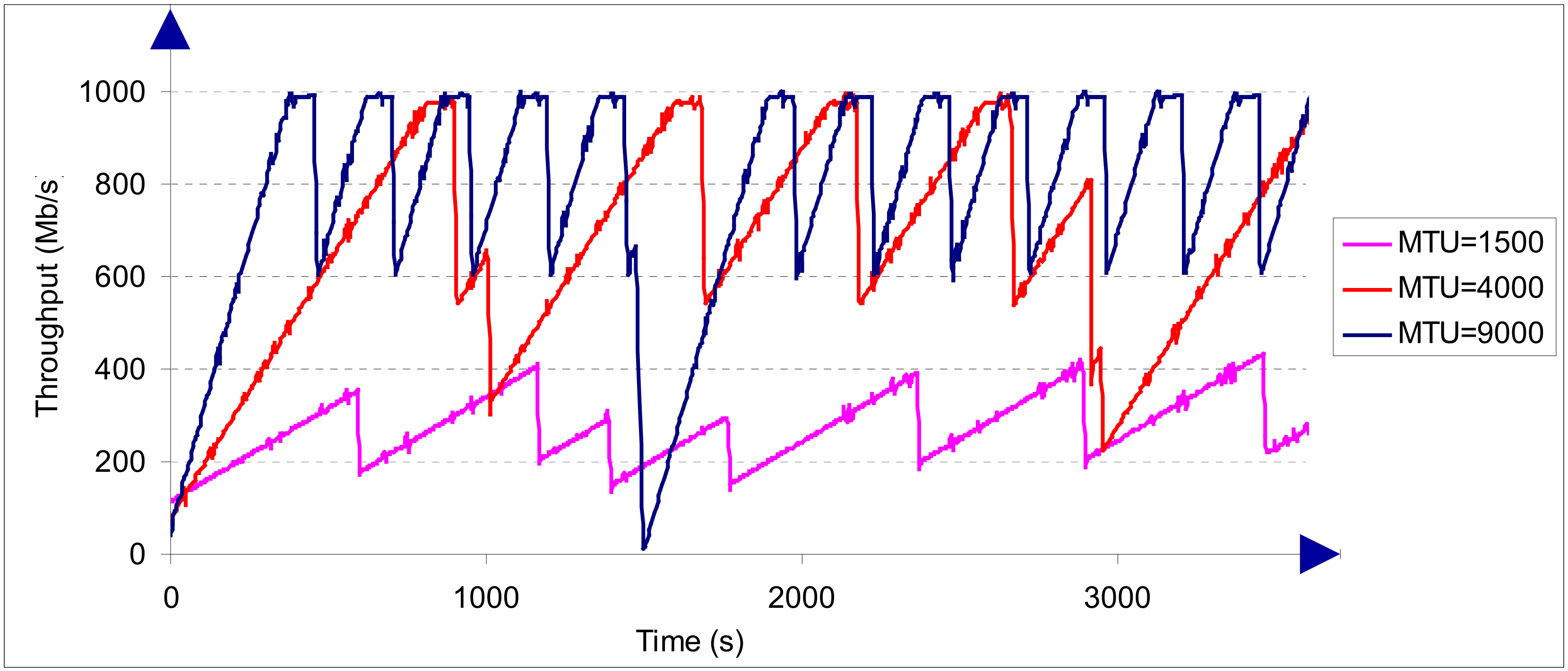
$$\frac{A1}{A2} = \frac{7}{3} = 2.33$$

Measurements with Different MTUs (1/2)

- Mathis advocates the use of large MTUs:
  - we tested standard Ethernet MTU and Jumbo frames
- Experimental environment:
  - Linux 2.4.19
  - Traffic generated by iperf
    - average throughout over the last 5 seconds
  - Single TCP stream
  - RTT = 119 ms
  - Duration of each test: 2 hours
  - Transfers from Chicago to Geneva
- MTUs:
  - POS MTU set to 9180
  - Max MTU on the NIC of a PC running Linux 2.4.19: 9000

Measurements with Different MTUs (2/2)

TCP max: 990 Mbit/s (MTU=9000)
UDP max: 957 Mbit/s (MTU=1500)



Measurement Tools

- We used several tools to investigate TCP performance issues:
  - Generation of TCP flows: *iperf* and *gensink*
  - Capture of packet flows: *tcpdump*
  - *tcpdump* → *tcptrace* → *xplot*
- Some tests performed with SmartBits 2000

Delayed ACKs

- RFC 2581 (spec. defining TCP congestion control AIMD algorithm) erred:

$$cwnd_{i+1} = cwnd_i + \frac{SMSS \times SMSS}{cwnd_i}$$

- Implicit assumption: one ACK per packet
- In reality: one ACK every second packet with delayed ACKs
- Responsiveness multiplied by two:
  - Makes a bad situation worse in fast WANs
- Problem fixed by RFC 3465 (Feb 2003)
  - Not implemented in Linux 2.4.20

Related Work

- Floyd: High-Speed TCP
- Low: Fast TCP
- Katabi: XCP
- Web100 and Net100 projects
- PFLDnet 2003 workshop:
  - http://www.datatag.org/pfldnet2003/

Research Directions

- Compare performance of TCP variants
- More stringent definition of congestion:
  - Lose more than 1 packet per RTT
- ACK more than two packets in one go:
  - Decrease ACK bursts
- SCTP vs. TCP