Efficient Network Protocols for Data-Intensive Worldwide Grids

Seminar at JAIST, Japan
3 March 2003

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

Outline

- DataTAG project
- Problems with TCP in data-intensive Grids
- Analysis and characterization
- Scalable TCP
- GridDT
- Research directions

The DataTAG Project

http://www.datatag.org/

Facts About DataTAG

- Budget: EUR ~4M
- Manpower:
  - 24 people funded
  - 30 people externally funded
- Start date: 1 January 2002
- Duration: 2 years

Three Objectives

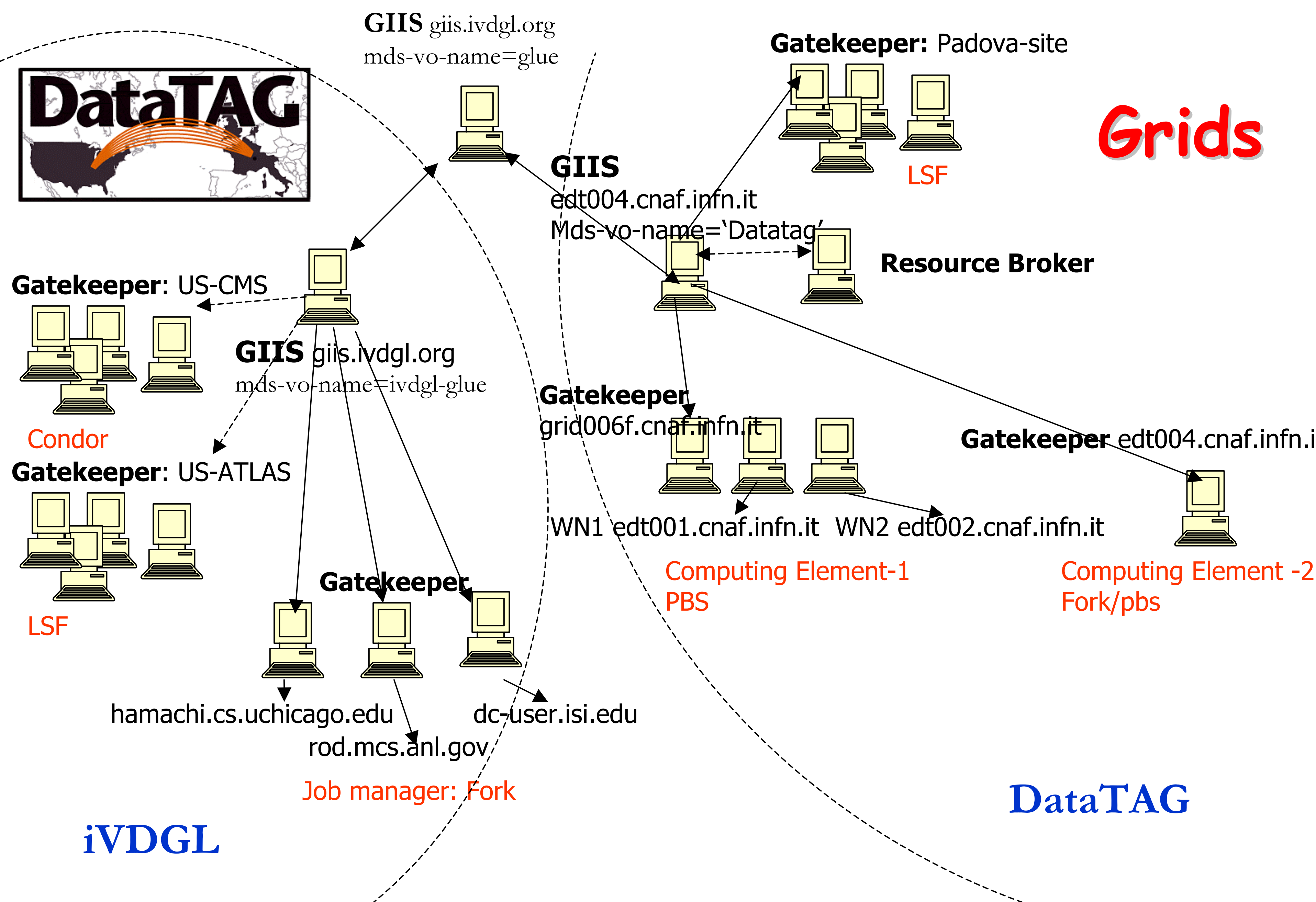
- Build a testbed to experiment with massive file transfers across the Atlantic
- Provide high-performance protocols for gigabit networks underlying data-intensive Grids
- Guarantee interoperability between several major Grid projects in Europe and USA

Collaborations

- *Testbed:* Caltech, Northwestern University, UIC, UMich, StarLight
- *Network Research:*
  - *Europe:* GEANT + Dante, University of Cambridge, Forschungszentrum Karlsruhe, VTHD, MB-NG, SURFnet
  - *USA:* Internet2 + Abilene, SLAC, ANL, FNAL, LBNL, ESnet
  - Canarie
- *Grids:* DataGrid, GridStart, CrossGrid, iVDGL, PPDG, GriPhyN, GGF

DataTAG

Grids


DataTAG
GIIS giis.ivdgl.org
mds-vo-name=glue
Gatekeeper: Padova-site
LSF
Grids
GIIS
edt004.cnaf.infn.it
Mds-vo-name='Datatag'
Resource Broker
Gatekeeper: US-CMS
GIIS giis.ivdgl.org
mds-vo-name=ivdgl-glue
Condor
Gatekeeper: US-ATLAS
LSF
Gatekeeper
grid006f.cnaf.infn.it
Gatekeeper edt004.cnaf.infn.it
WN1 edt001.cnaf.infn.it
WN2 edt002.cnaf.infn.it
Computing Element-1
PBS
Computing Element -2
Fork/pbs
Gatekeeper
hamachi.cs.uchicago.edu
dc-user.isi.edu
rod.mcs.anl.gov
Job manager: Fork
DataTAG
iVDGL

Grids in DataTAG

- Interoperability between European and U.S. Grids:
  - High Energy Physics (main focus)
  - Bioinformatics
  - Earth Observation
- Grid middleware:
  - DataGrid
  - iVDGL VDT (shared by PPDG and GriPhyN)
- Information modeling (GLUE initiative)
- Software development

Testbed

Objectives

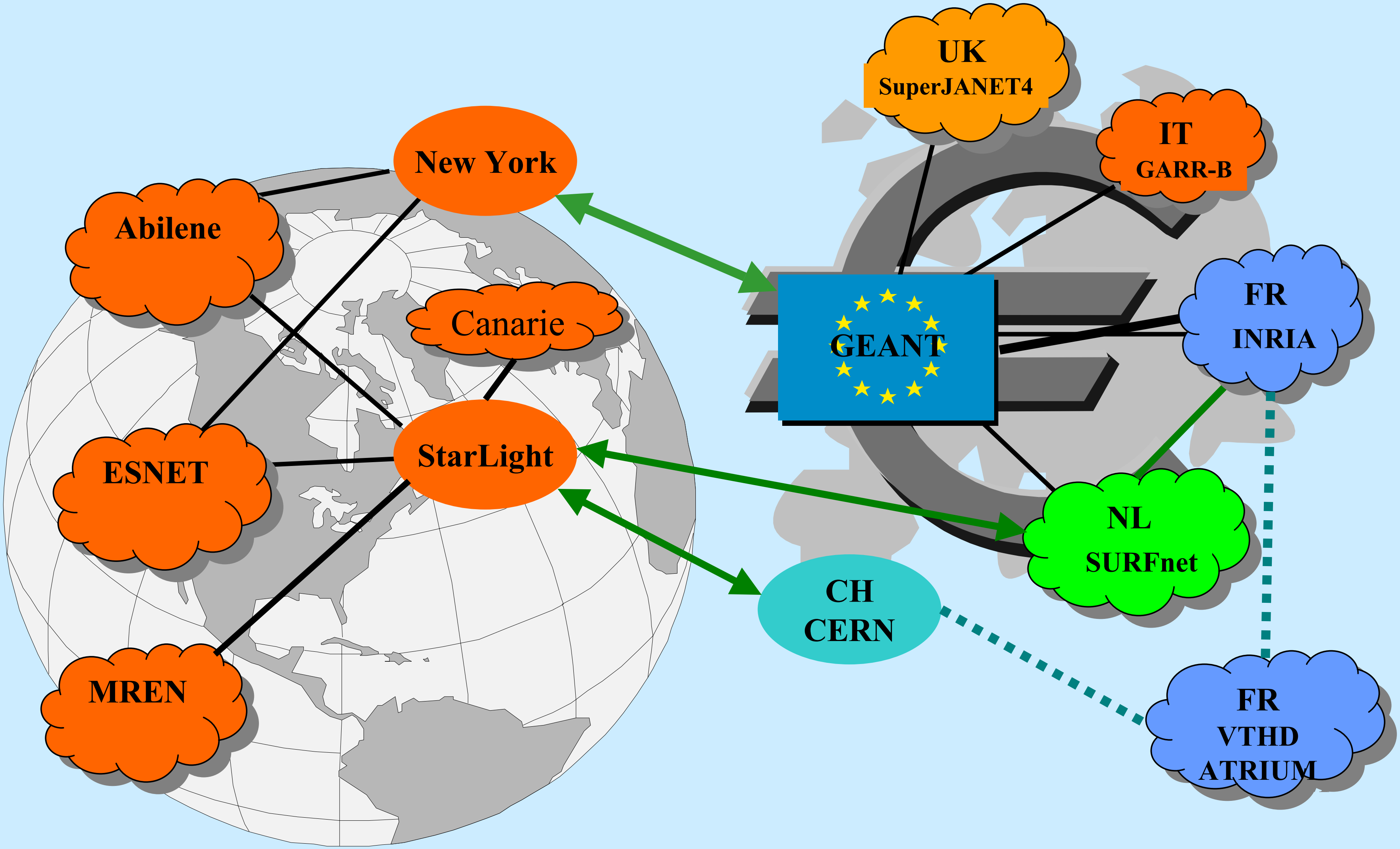
- Provisioning of 2.5 Gbit/s transatlantic circuit between CERN (Geneva) and StarLight (Chicago)
- Dedicated to research (no production traffic)
- Multi-vendor testbed with layer-2 and layer-3 capabilities:
  - Cisco, Juniper, Alcatel, Extreme Networks
- Get hands-on experience with the operation of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

DataTAG

2.5 Gbit/s Transatlantic Circuit

- Operational since 20 August 2002
- Provisioned by Deutsche Telekom
- Circuit initially connected to Cisco 76xx routers (layer 3)
- 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 GbE cards (2 per PC)
  - total disk space: 1680 GB
  - can saturate the circuit with TCP traffic
- Deployment of layer-2 equipment underway
- Upgrade to 10 Gbit/s expected in 2003

R&D Connectivity Between Europe & USA



Network Research

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

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 currently inefficient in high bandwidth*delay networks for 2 reasons:
  - 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
- Device drivers 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:

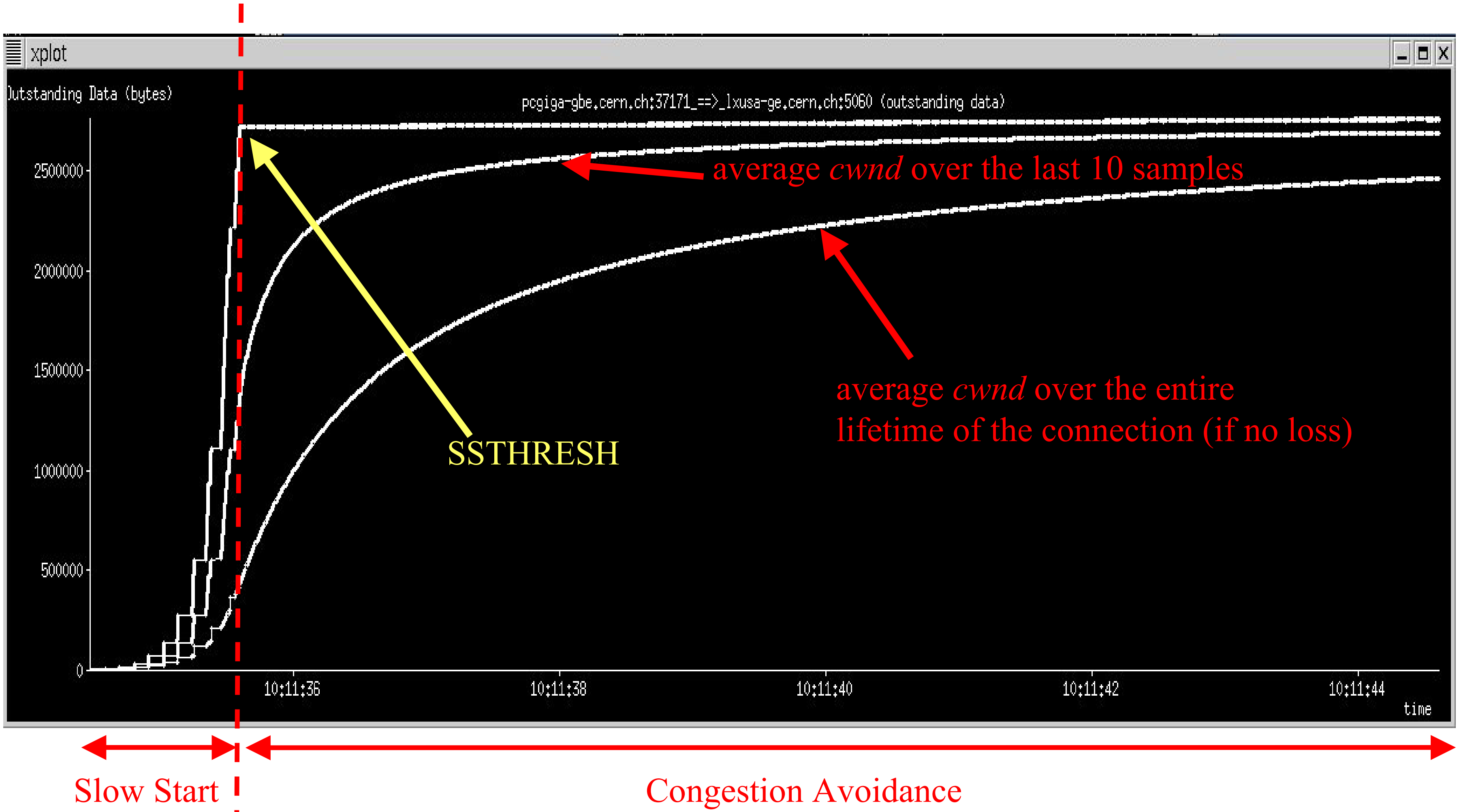
$$cwnd_{i+1} = cwnd_i - \frac{1}{2} \times cwnd_i$$

- Slow-start algorithm:
  - Increase by 1 MSS per ACK until *ssthresh*

AIMD Algorithm (2/2)

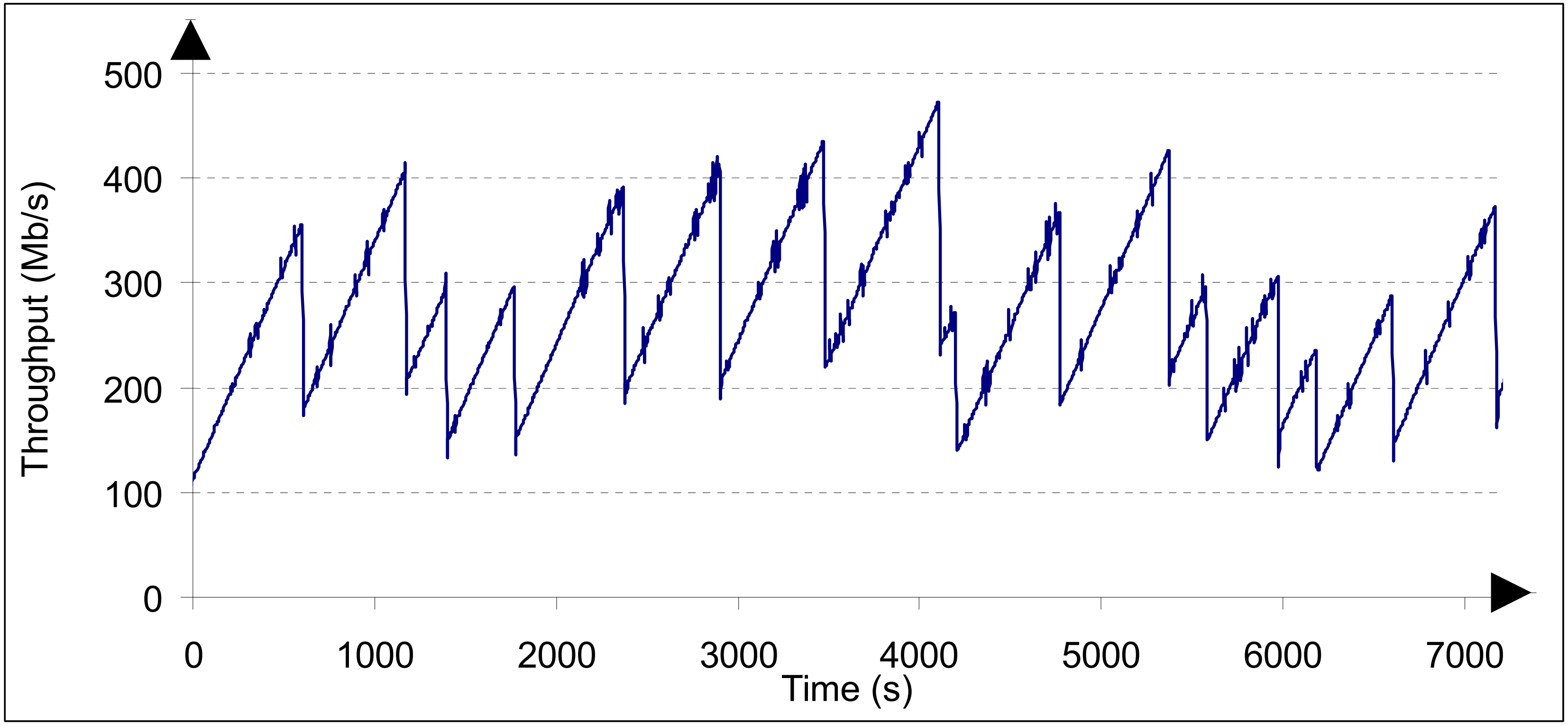
- **Additive Increase:**
  - A TCP connection increases slowly its bandwidth utilization in the absence of loss:
    - forever, unless we run out of send/receive buffers or detect a packet loss
    - TCP is greedy: no attempt to reach a stationary state
- **Multiplicative Decrease:**
  - A TCP connection 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)

TCP NewReno throughput as a function of time **C=1Gbit/s, MSS=1460bytes**



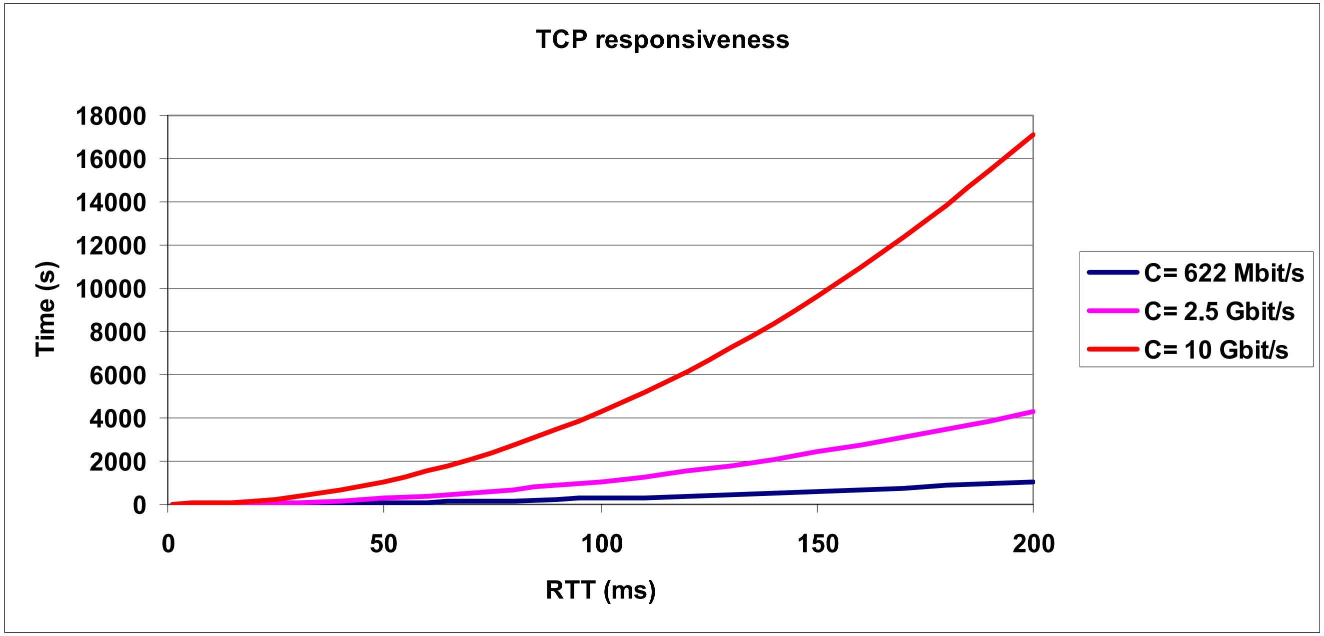
Disastrous Effect of Packet Loss on TCP in Fast WANs (2/2)

- 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)
  - 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)

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



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=1500 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

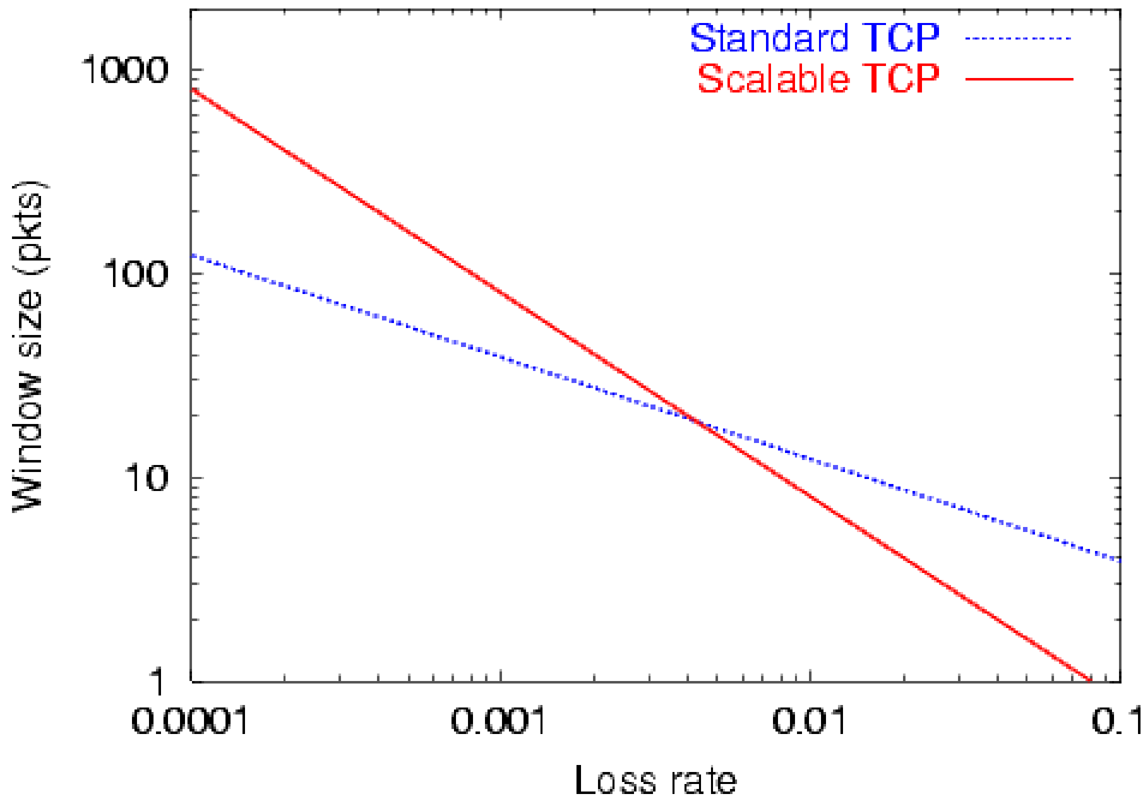
What Can We Do?

- To achieve higher throughputs over high bandwidth*delay networks, we can:
  - Change AIMD to recover faster in case of packet loss:
    - larger *cwnd* increment
    - less aggressive decrease algorithm
    - larger MTU (Jumbo frames)
  - Set the initial slow-start threshold (*ssthresh*) to a value better suited to the delay and bandwidth of the TCP connection
  - Avoid losses in end hosts:
    - implementation issue
- Two proposals: Scalable TCP (Kelly) and GridDT (Ravot)

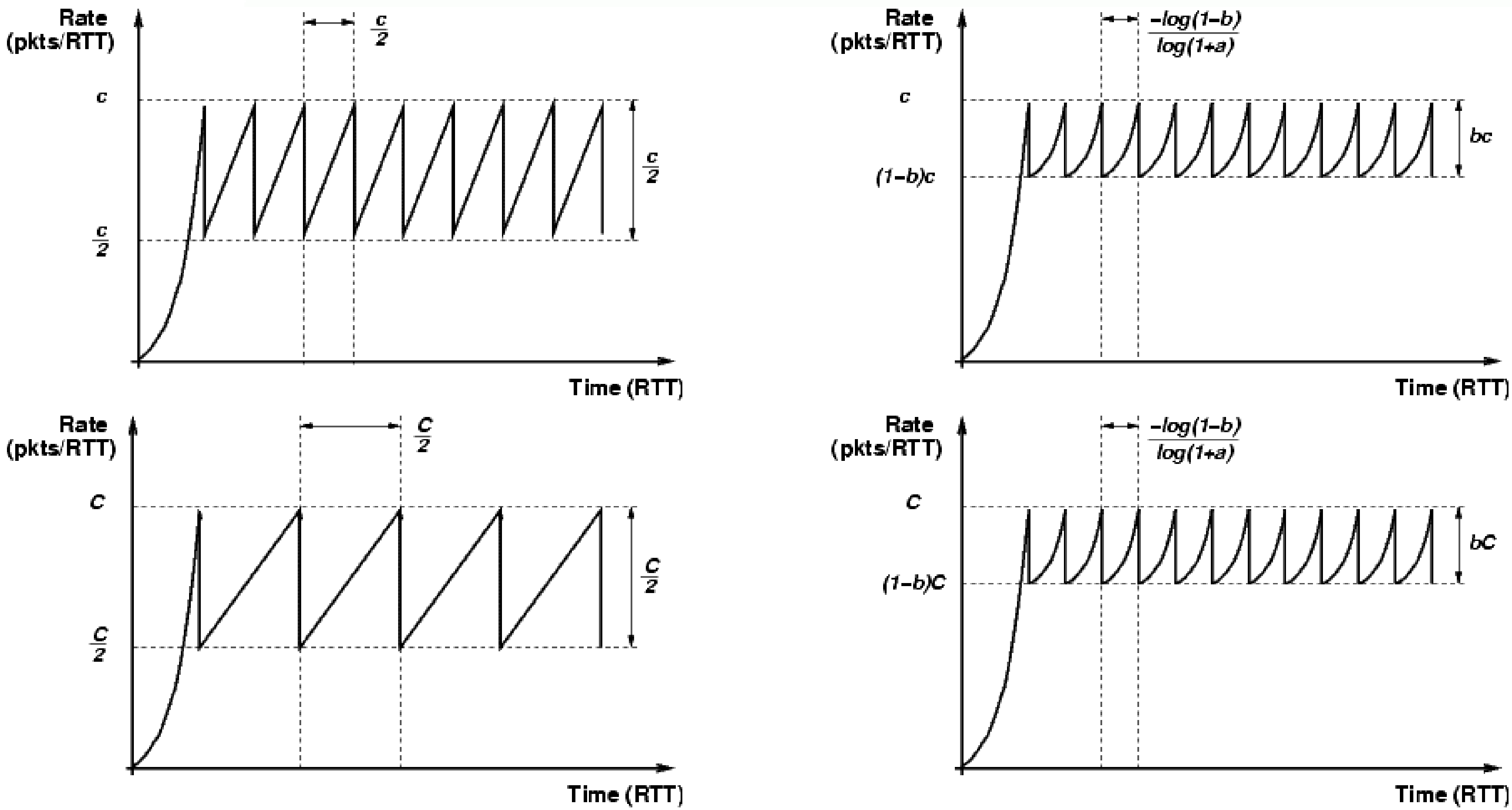
Scalable TCP: Algorithm

- 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=1460 bytes:
  - Scalable TCP: 2.7 s
  - TCP NewReno (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 details, see paper and code at:
  - http://www-lce.eng.cam.ac.uk/~ctk21/scalable/

Scalable TCP vs. TCP NewReno: 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 1200s.

GridDT: Algorithm

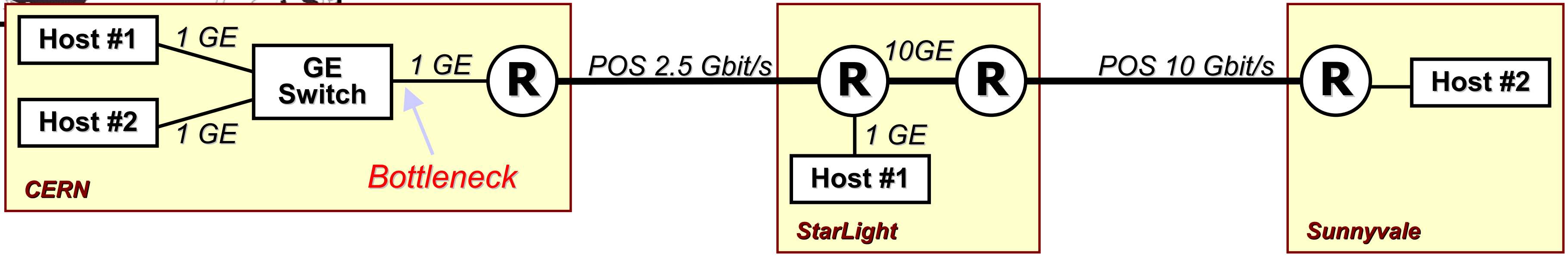
- Congestion avoidance algorithm:
  - For each ACK in an RTT without loss, increase:

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

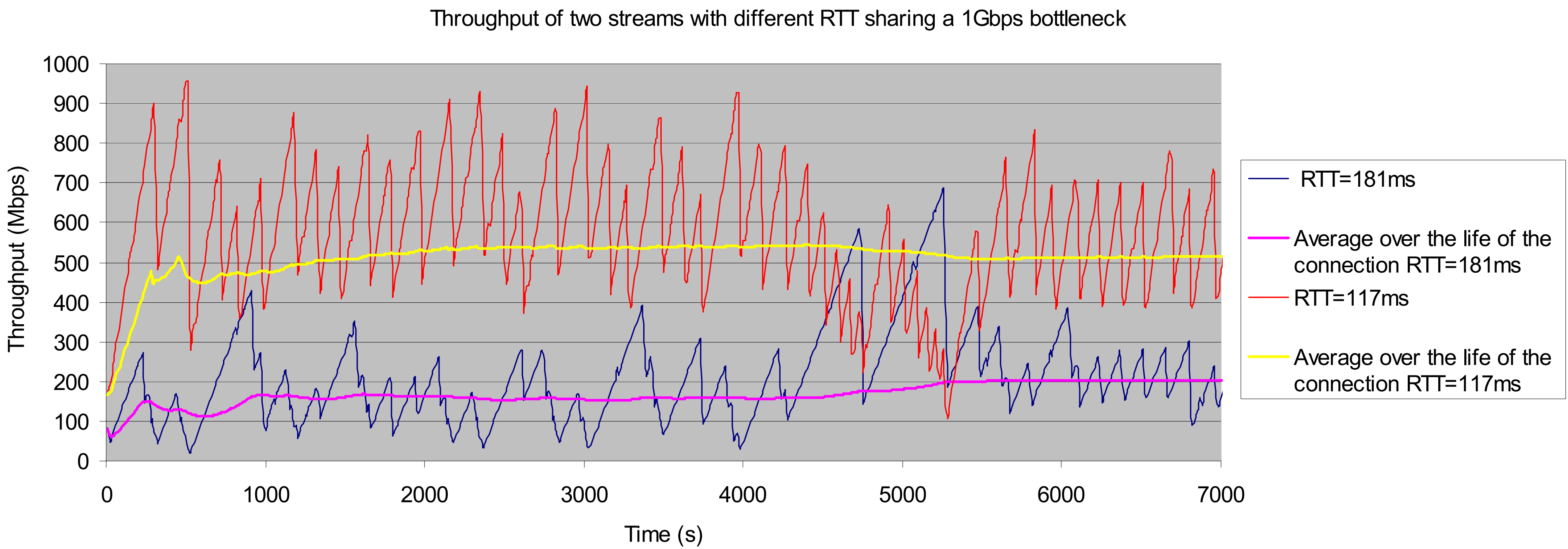
- By modifying A dynamically according to RTT, guarantee fairness among TCP connections:

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

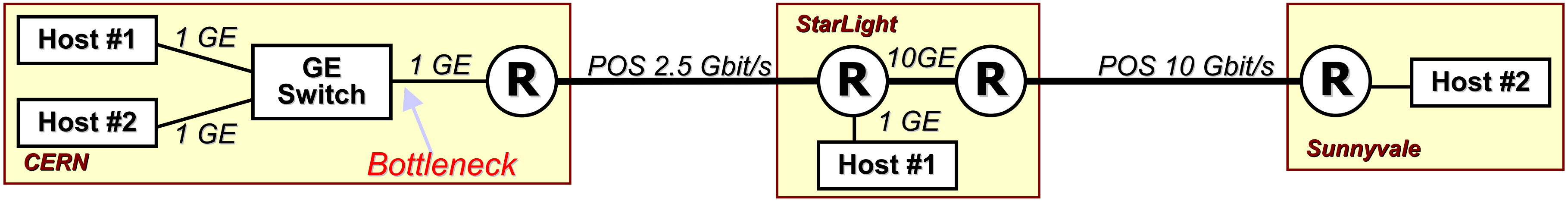
TCP NewReno: RTT Bias



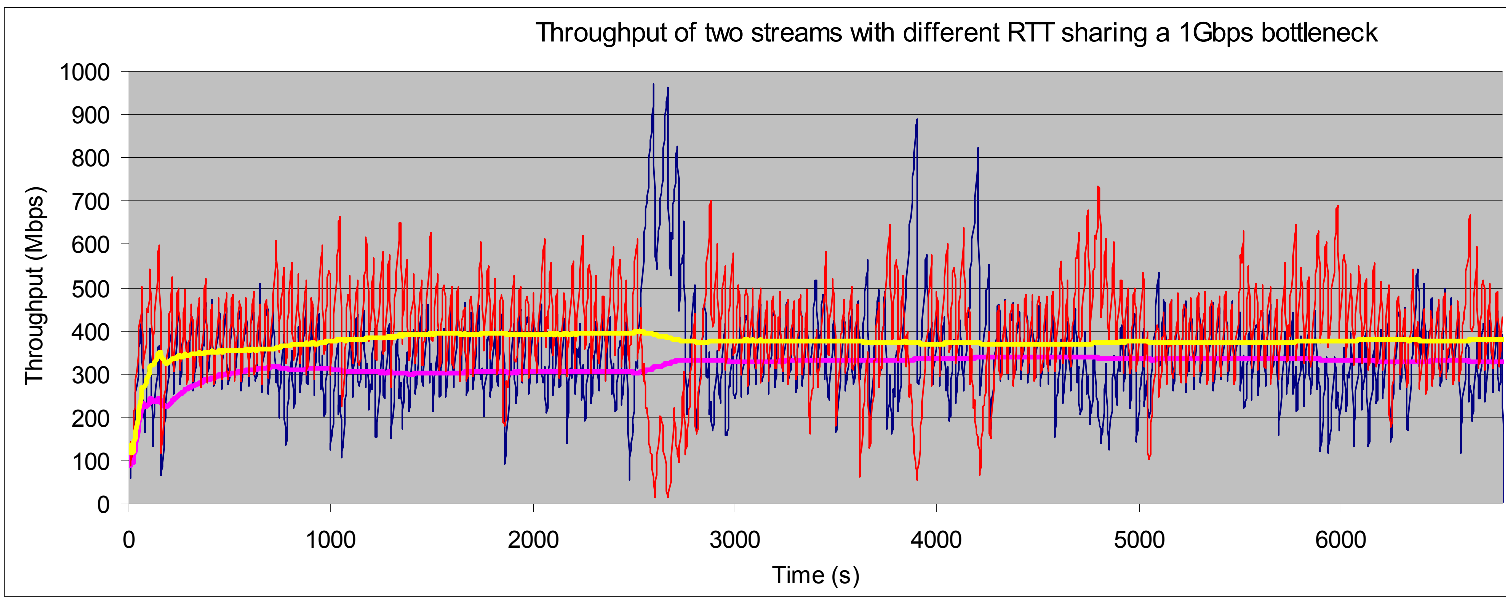
- Two TCP streams share a 1 Gbit/s bottleneck
- CERN-Sunnyvale: RTT=181ms. Avg. throughput over a period of 7000s = 202Mbit/s
- CERN-StarLight: RTT=117ms. Avg. throughput over a period of 7000s = 514Mbit/s
- MTU = 9000 bytes. Link utilization = 72%



GridDT Fairer than TCP NewReno



- 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 = 9000 bytes. Link utilization 72%



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

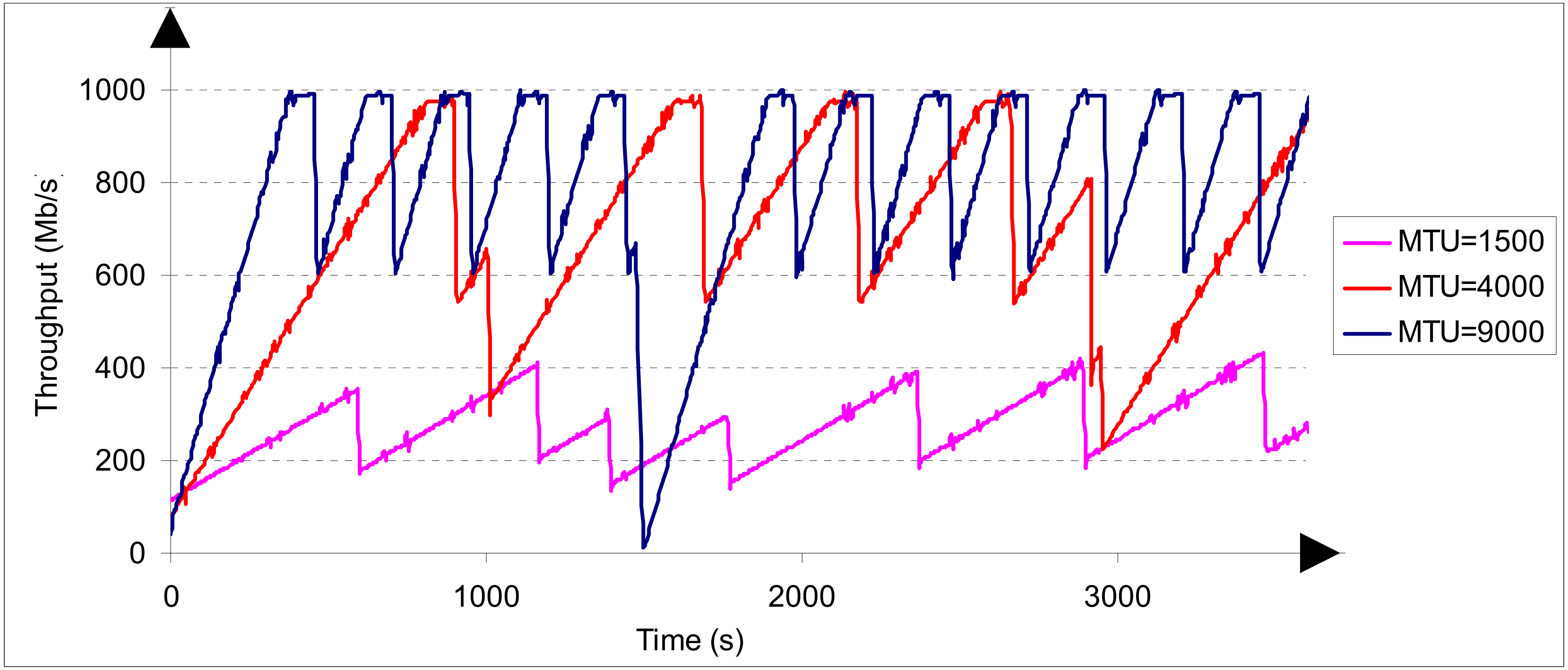
$$\frac{A1}{A2} = \frac{7}{3} = 2.33$$

Measurements with Different MTUs (1/2)

- Mathis advocates the use of larger MTUs
- 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:
  - set on the NIC of the PC (*ifconfig*)
  - POS MTU set to 9180
  - Max MTU with 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
- Delayed ACKs: one ACK every second packet
- Responsiveness multiplied by two:
  - Makes a bad situation worse when RTT and *cwnd* are large
- Allman preparing an RFC to fix this

Related Work

- Sally Floyd, ICIR: Internet-Draft “High Speed TCP for Large Congestion Windows”
- Steven Low, Caltech: Fast TCP
- Dina Katabi, MIT: XCP
- Web100 and Net100 projects
- PFLDnet 2003 workshop:
  - http://www.datatag.org/pfldnet2003/

Research Directions

- Compare the performance of different proposals
- More stringent definition of congestion:
  - Lose more than 1 packet per RTT
- ACK more than two packets in one go:
  - Decrease ACK bursts
- Use SCTP instead of TCP