Implementation issues for high-speed TCPs

1st October 2003

UCL

Tom Kelly

ctk21@cam.ac.uk

Laboratory for Communication Engineering

University of Cambridge

Motivation for new high-speed TCPs

- Some Internet users (mainly scientific) want to perform very large bulk transfers
- TCP congestion control performs poorly at high-speeds in wide area networks
- Even “turning off” congestion control functions unreliably on some implementations

Problem area 1: algorithm

- Poor performance of TCP in high bandwidth wide area networks due to TCP congestion control algorithm

Throughput	Window	Loss recovery time	Supporting loss rate
10Mbps	170pkts	17s	5.4×10^{-5}
100Mbps	1700pkts	2mins 50s	5.4×10^{-7}
1Gbps	17000pkts	28mins	5.4×10^{-9}
10Gbps	170000pkts	4hrs 43mins	5.4×10^{-11}

Characteristics of a 200ms, MTU 1500 bytes TCP connection

Problem area 2: OS implementation

- What is causing the Linux TCP stack to become unpredicatable with large windows?
  - Problem with PAWs implementation?
  - Hardware/software driver issues?
  - SACK implementation problems?

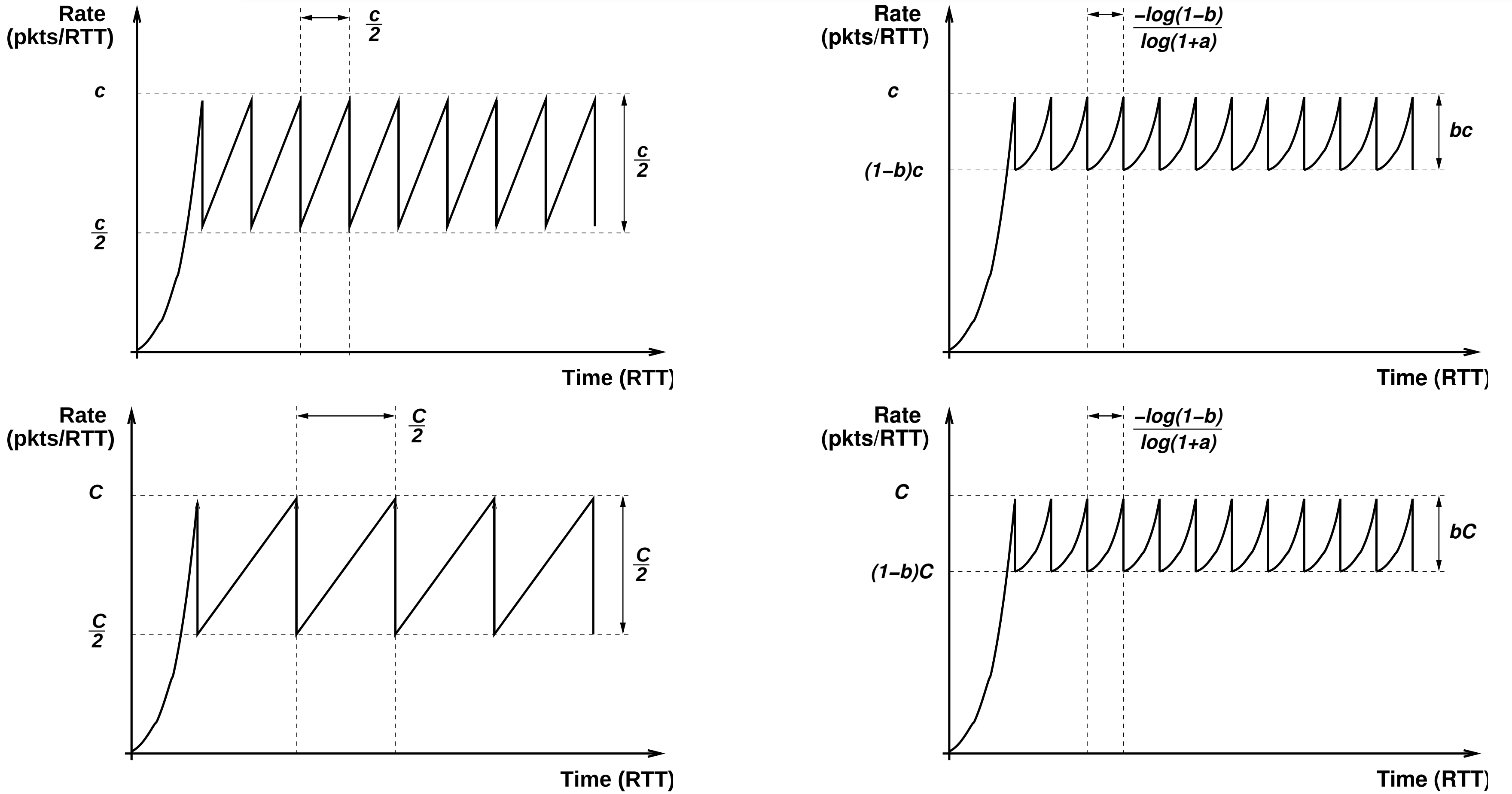
Changing the algorithm - aims and assumptions

- Make effective use of high bandwidth links
- Changes need to be robust in a wide variety of networks and traffic conditions
  - L2 switches, bugs, packet corruption, reordering and jitter
- Do not adversely damage existing network traffic
- Do not require manual tuning to achieve reasonable performance
  - 80% of maximal performance for 95% of the people for the foreseeable future

The generalised Scalable TCP algorithm

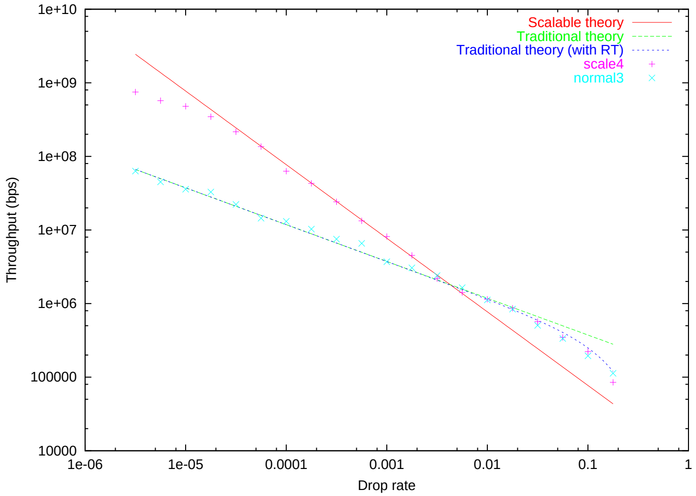
- Let a and b be constants
  - for each ack in a RTT without loss:
    $cwnd_r \leftarrow cwnd_r + a$
  - for each window experiencing loss:
    $cwnd_r \leftarrow cwnd_r - b \times cwnd_r$
- Loss recovery times for RTT 200ms and MTU 1500 bytes
  - Scalable TCP: $\frac{log(1-b)}{log(1+a)}$ RTTs
    e.g. if $a = 0.01, b = 0.125$ then it is about 2.7s
  - Traditional: at 50Mbps about 1min 38s, at 500Mbps about 27min 47s!

The Scalable TCP algorithm



Fairness

- Choose a legacy window size, $lwnd$
- When $cwnd > lwnd$ use the Scalable TCP algorithm
- When $cwnd \leq lwnd$ use traditional TCP algorithm



- Same argument used in the HighSpeed TCP proposal
- Fixing $lwnd$, fixes the ratio $\frac{a}{b}$

Variance and Convergence

- increasing b, more variable flows but faster backoff
- increasing a, instability but more agressive ramp up
- $lwnd = 16$, $a = 0.01$, and $b = 0.125$ represents a good trade off of concerns

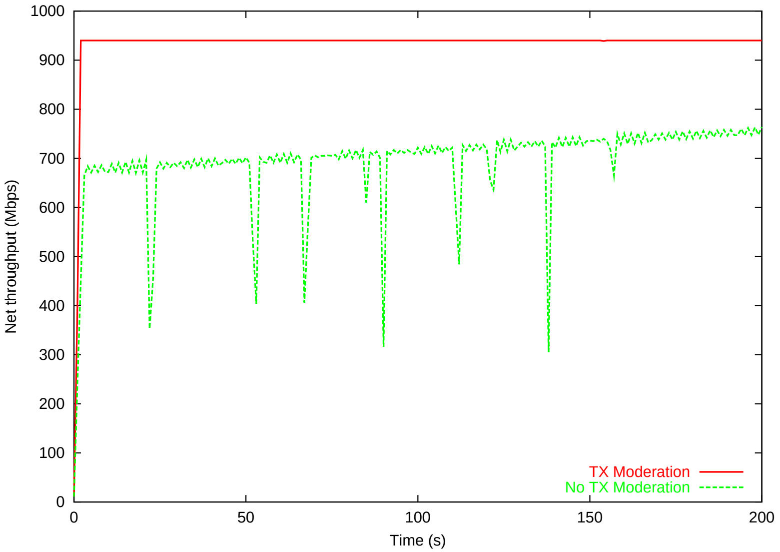
b	a	Rate CoV	Loss recovery time	Rate halving time	Rate doubling time
$\frac{1}{2}$	$\frac{1}{25}$	0.50	$17.7T_r$ (3.54s)	T_r (0.20s)	$17.7T_r$ (3.54s)
$\frac{1}{4}$	$\frac{1}{50}$	0.35	$14.5T_r$ (2.91s)	$2.41T_r$ (0.48s)	$35T_r$ (7.00s)
$\frac{1}{8}$	$\frac{1}{100}$	0.25	$13.4T_r$ (2.68s)	$5.19T_r$ (1.04s)	$69.7T_r$ (13.9s)
$\frac{1}{16}$	$\frac{1}{200}$	0.18	$12.9T_r$ (2.59s)	$10.7T_r$ (2.15s)	$139T_r$ (27.8s)

Linux TCP implementation

- Includes all standard high-speed TCP extensions; PAWs, timestamps, SACK
- Includes some experimentatal non-standard features:
  - Forward acknowledgement (FACK) to capture flight size during recovery
  - Rate-halving; send packet every other acknowledgement during recovery
  - Aggressive RTO checking on sent segements when receiving duplicate acknowledgements
  - Mechanisms for undoing congestion window decreases if thought to be due to bogus loss detection

Impact of driver TX interrupts

- Default Linux SysKonnect does no transmit interrupt moderation
- By altering the driver TX interrupts can be moderated



Linux NAPI driver model

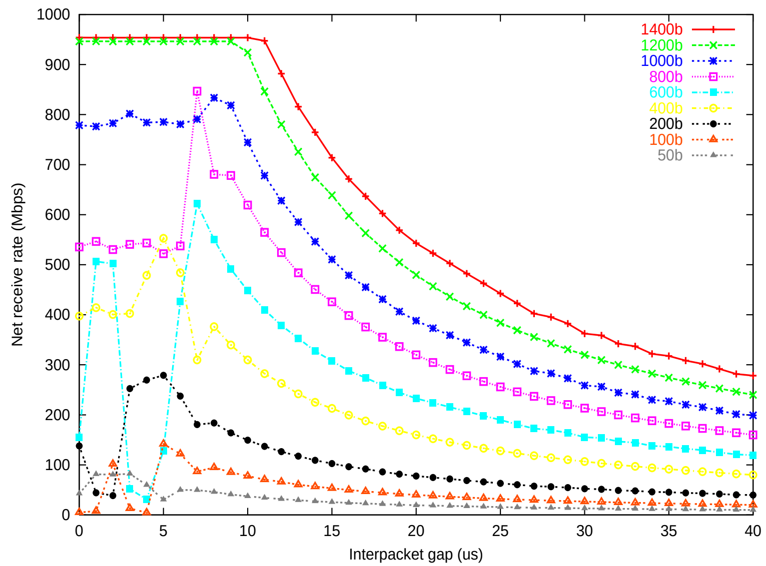
- Around for some time in 2.5.x and incorporated in 2.4.20
- On receiving a packet, NIC raises interrupt
- Driver switches off RX interrupts and schedules RX DMA ring poll
- Frames are pulled off DMA ring and is processed up to application
- When all frames are processed RX interrupts are re-enabled
- Dramatic reduction in RX interrupts under load

Experimental SysKonnect NAPI driver implemented

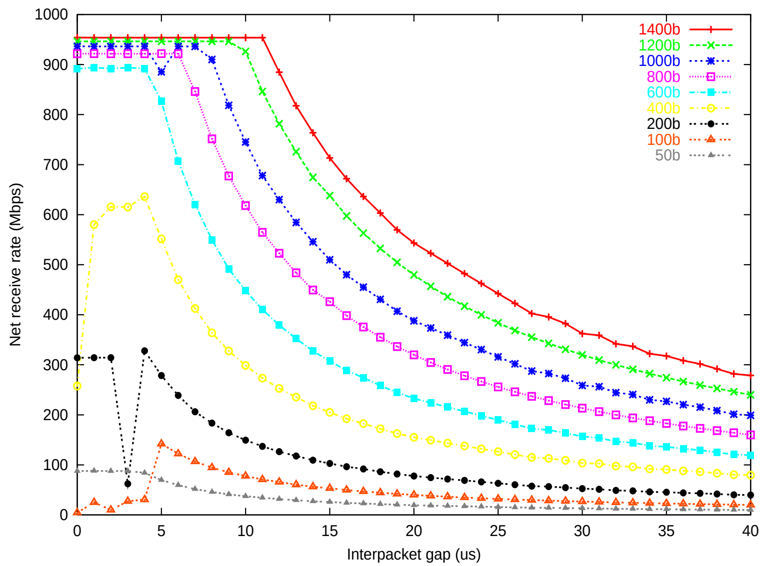
- No spec sheet for PCI card ASIC since SysKonnect was bought by Marvell
- Still some RX flagged interrupts appearing; appears benign but makes me su spect there is a bug somewhere
- Bottom line is improved performance under heavy load

NAPI receiver results

- 2.4Ghz machines connected through router with 2.4.20 sender using TX inter rupt moderation



2.4.19 non-NAPI receiver



2.4.20 NAPI receiver

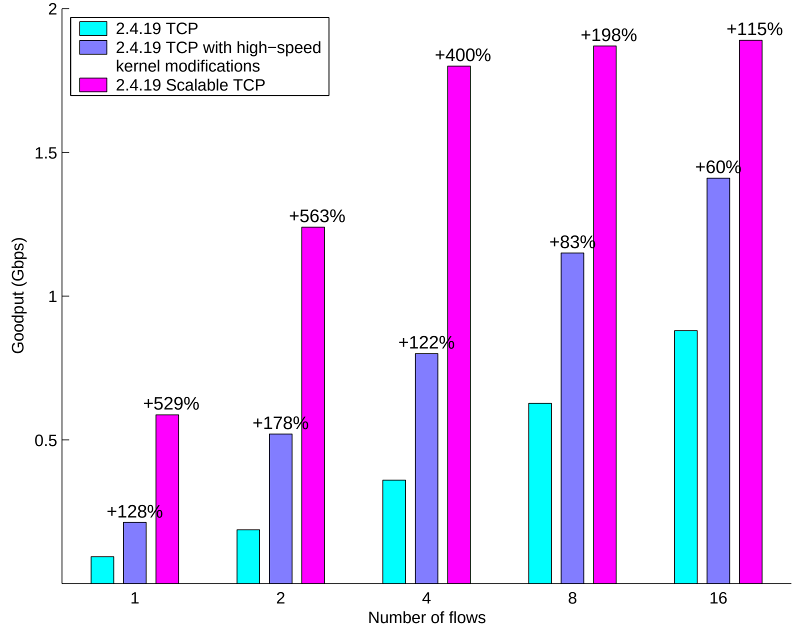
- Better throughput for NAPI receiver under load
- Some strange behavior with 100b and 50b packets...

Scaling loss detection and recovery algorithms

- SACK block processing and segment retranmission both involve trawling the send queue
- Trawling the send queue can be $O(cwnd)$ for each acknowledgement
  - The queues are there to avoid copying packets
- A fix (hack) is to exploit likely fastpath
  - Packets delivery in order
  - SACK blocks in acks only change in first block
  - Cache pointers and assume incremental changes each ack

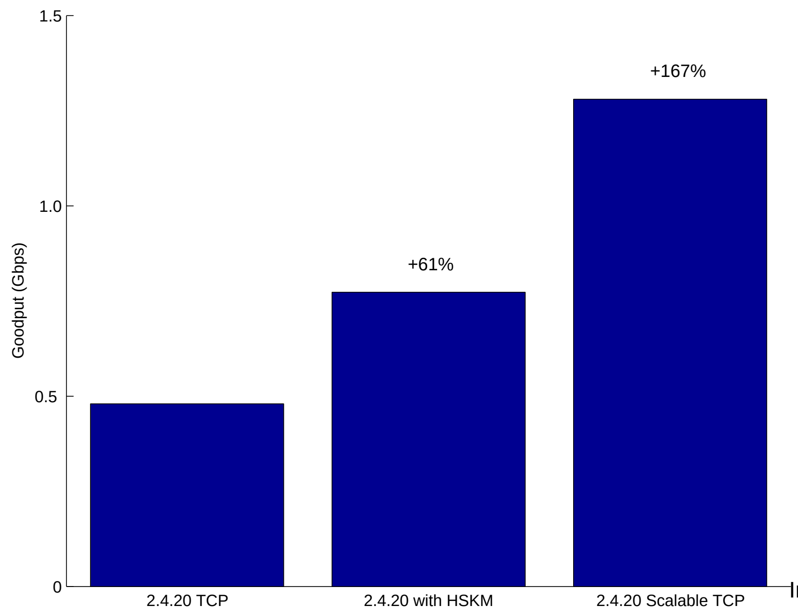
Bulk throughput

- DataTAG 2.4Gbps link and minimal buffers (2048/40)
- Flows transfer 2 gigabytes and start again for 1200s



Web traffic results

- DataTAG 2.4Gbps link and minimal buffers (2048/40)
  - 4 bulk concurrent flows across 2 machines for 1200s
  - 4200 concurrent web users across 3 machines
- No change in web traffic with and without bulk transfers in all scenarios



Problems

- Result set is small!
  - Difficult to conduct controlled implementation experiments
- Linux TCP implementation a mess for high-speed
  - Should split data segments from packet headers and protocol state (e.g. OpenBSD)
  - Need scatter-gather I/O to do this with minimal copies
- Scalable TCP
  - Synchronisation problems due to design, worse than TCP but simulations don't match reality
  - Which workloads and topologies m?

Conclusion

- Linux implementation can be greatly improved for high-speed operation
- Scalable TCP an easy evolution from the traditional TCP AMID scheme which can improve performance
- Much more to be done deciding between schemes; HSTCP, Vegas/FAST, Westwood, etc.
- Freely available working code `http://www-lce.eng.cam.ac.uk/~ctk21/scalable`