Congestion Avoidance and Control

Ricardo Sexauer

4th Semester B.Sc. in Computer Science

Summary of Van Jacobson and Michael J. Karels 1988s paper.[1]

Albert-Ludwigs-Universität Freiburg

UNI FREIBURG

Proseminar Algorithms for Computer Networks

Prof. Dr. Christian Schindelhauer

Johannes Wendeberg

Autors

- Van Jacobson is a primary contributor to technological foundation of today´s Internet.[2][3][4]
  - Renowned for pioneering achievements in network performance and scaling.
  - Enabled Internet to expand to support increasing demands of speed & size.
  - Received 2001 ACM Sigcomm Award for Lifetime Achievement.

- Michael J. Karels is one of the key people in the history of BSD.[5][6]
  - Information Week magazine:
    4.3BSD the "Greatest Software Ever Written" (Aug. 2006).[7]

Motivation

Why is this paper so important?

- Strategy to handle TCP congestion used in 90 % of hosts today.
- Helped the Internet to survive a major traffic surge (1988-89) without collapsing.

Overview

UNI FREIBURG

- Introduction
- Getting to equilibrium.
- Conservation at equilibrium.
- Adapting to the path.
- Summary.
- Future work.

Introduction

- Explosive growth of computer networks in the 80´s led to severe congestion problems.
  - E.g. buffer overflow in gateways, 10 % of packets dropped common.
- “Obvious” implementations of window-based transport protocols resulted in wrong behavior on congestion.
- October 1986: first “congestion collapse” of the Internet:
  - Data throughput from LBL to UCB (two IMP hops, 300 m) dropped from 32 Kbps to 40 bps.

Introduction

- Why had things gotten so bad?
  - Was the 4.3 BSD TCP implementation mis-behaving?
  - Could it be tuned to work under abysmal network conditions?

- The solution: Congestion Avoidance and Control.
  - New extensions to the TCP Protocol to achieve network stability, based on “packet conservation” principle.

Introduction

- Seven new extensions to the TCP Protocol:
  i. Round-trip-time variance estimation.
  ii. Exponential retransmit timer backof.f
  iii. Slow-start.
  iv. More aggressive receiver acknowledgments (ACKs) policy.
  v. Dynamic window sizing on congestion.
  vi. Karn´s clamped retransmit backoff. (recently developed)
  vii. Fast retransmit. (soon-to-be-published RFC1122)

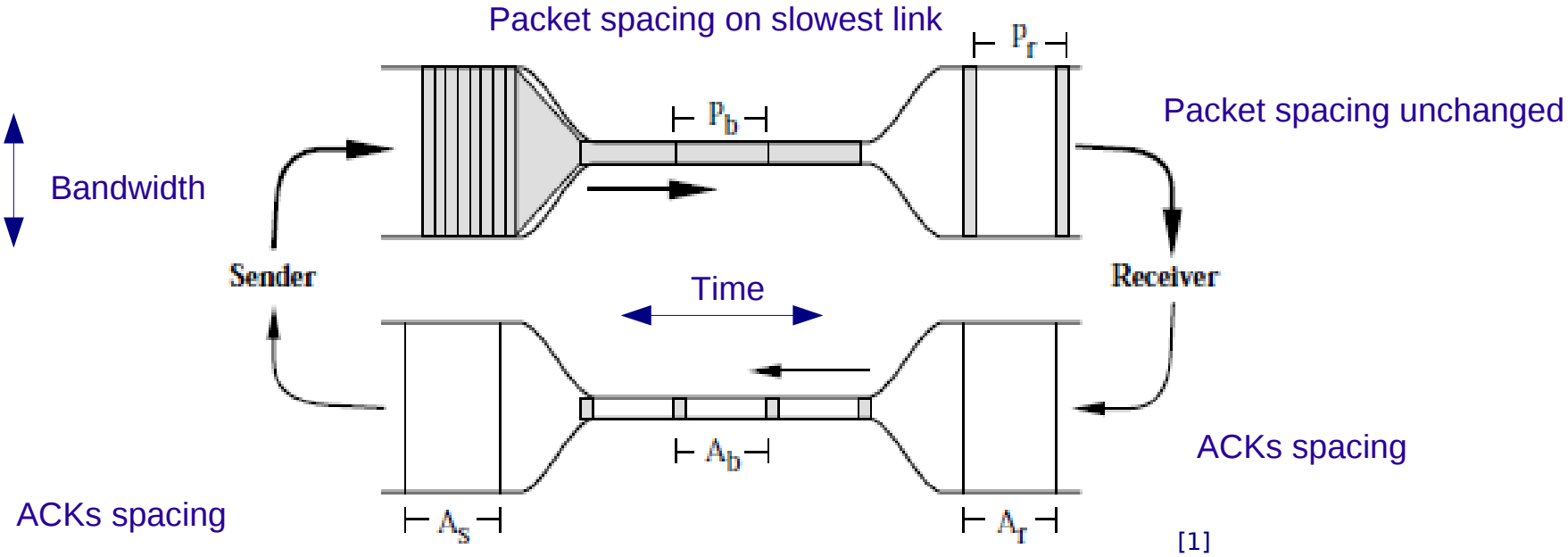
Introduction

- “Conservation of packets” principle: connection is “in equilibrium”.
  - No new packet is put into network until an old packet leaves.

- Reasons for packet conservation to fail:
  (1) Connection does not get to equilibrium.
  (2) Sender injects new packet before old packet leaves.
  (3) Equilibrium can not be reached due to resource limits.

Getting to equilibrium: slow-start

- Connection starting or restarting.
- Conservation property: use ACKs as clock to send new packets.
  - Receiver cannot generate acks faster than packets get through network.
- Window Flow control “Self clocking”:



[1]

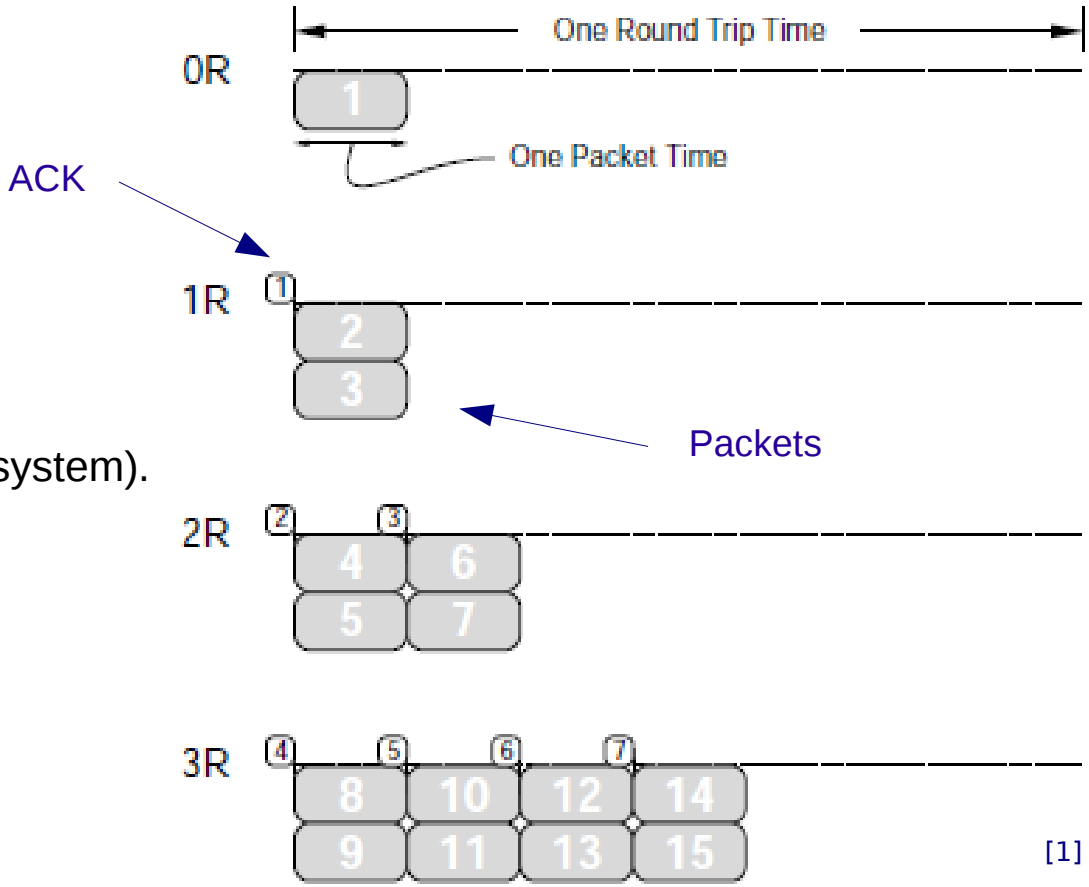
Getting to equilibrium: slow-start

- Packets sent only in response to ACKs:
  - Sender´s packet spacing matches packet time on slowest link in path (bottleneck).

- Self-clocking systems:
  - Automatic adjust to bandwidth and delay variations, wide dynamic range.
  - Stable when running, hard to start: need ACKs to clock out data, need data to get ACKs...

Getting to equilibrium: slow-start

- Start the clock : Slow-start.



- Generate two packets for each ACK:
  - one for the ACK (packet has left the system).
  - one to open up congestion window by one packet.

[1]

Getting to equilibrium: slow-start

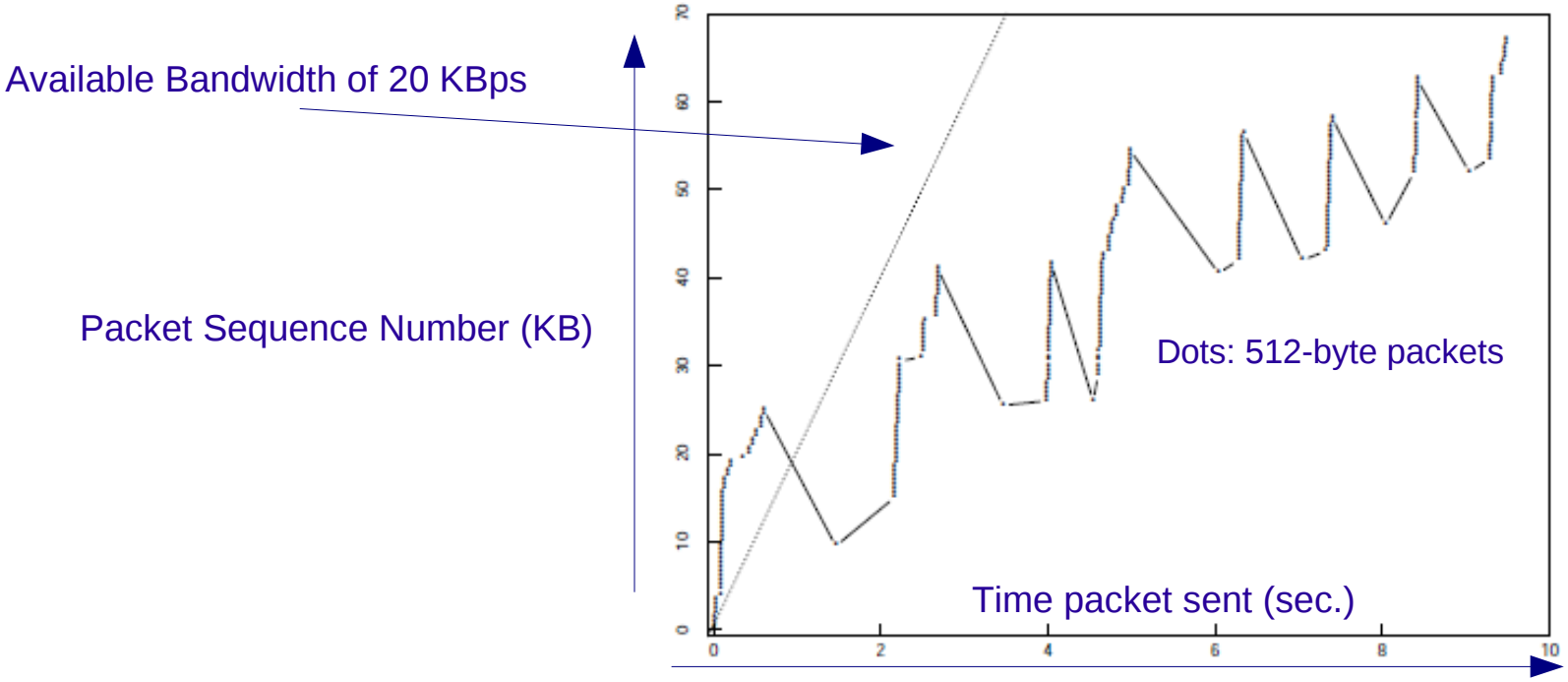
- Subtle algorithm, trivial implementation:
  - Add a congestion window, cwnd, to per-connection state.
  - When starting or restarting: cwnd = 1.
  - On each ACK: cwnd $\leftarrow$ cwnd + 1.
  - When sending: send minimum of receiver´s advertised window and cwnd.

- Window increase: $R \cdot \log_2 W$
  - R : round-trip-time.
  - W : window size.

- Guarantee: connection sources data at most twice max. possible on path.

Getting to equilibrium: slow-start

- Startup behavior of TCP without slow-start:

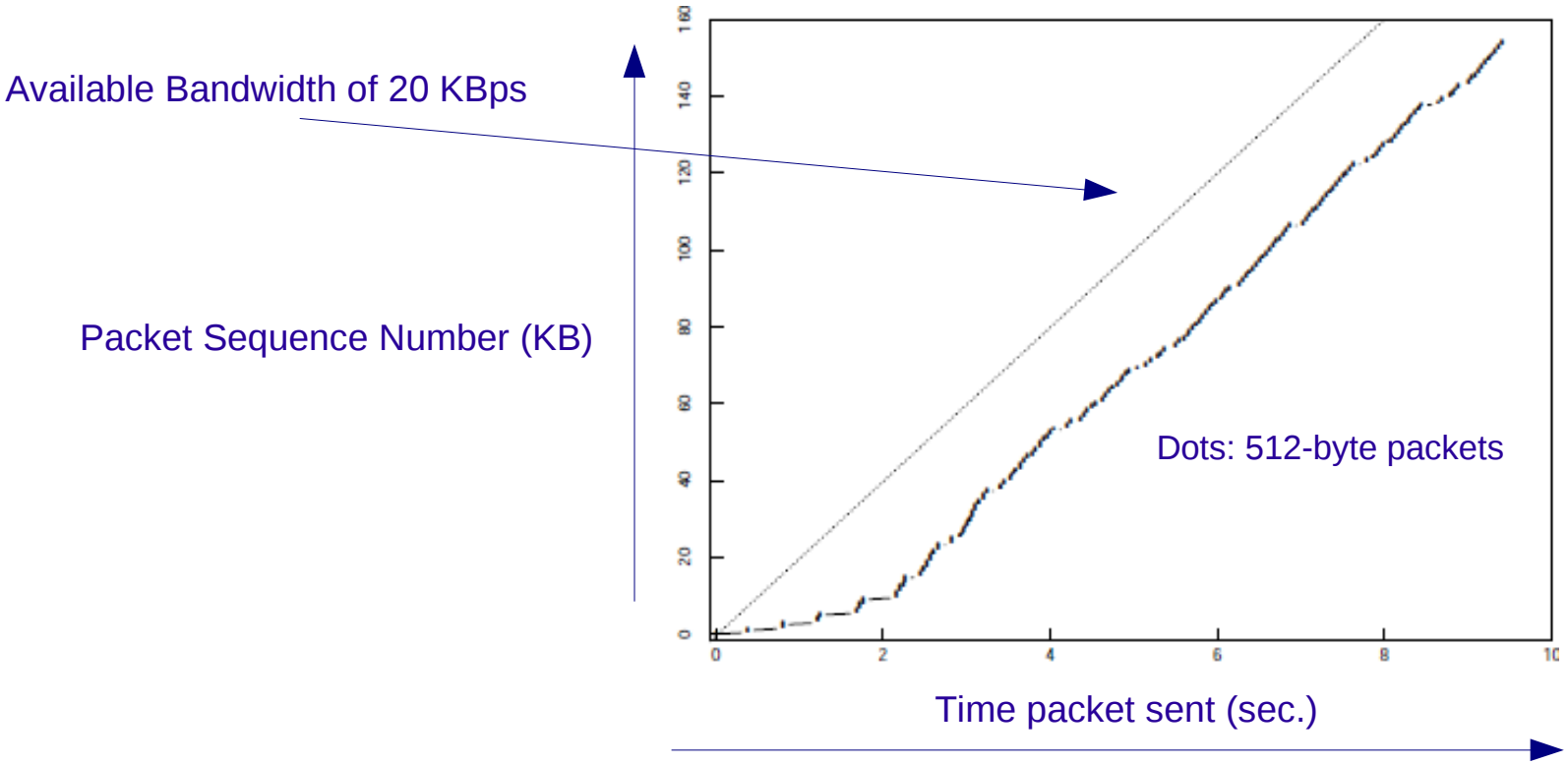


[1]

- Burst of packets puts connection into persistent failure mode of continuous retransmissions.
  - Only 35 % used (7 KBps), rest wasted on retransmits.
  - Almost everything retransmitted. Data from 54 to 58 KB: sent five times!

Getting to equilibrium: slow-start

- Startup behavior of TCP with slow-start:



[1]

- No retransmits. Only 2 seconds spent on slow-start.
- Effective bandwidth: 16 KBps. After 1 minute: 19 KBps.

Conservation at equilibrium: round-trip timing

- After data flows reliably (and correct protocol implementation):
  - Problem (2): sender injects new packet before old packet leaves
  - Means failure of sender´s retransmit timer.

- Round trip time estimator: core of retransmit timer, frequently botched.

- TCP protocol specification (RFC-793):
  - Estimation of round trip time (RTT) and retransmission timeout interval (rto):

$R \leftarrow \alpha \cdot R + (1 - \alpha) \cdot M$ (α : filter gain = 0.9)

$rto = \beta \cdot R$ (β = 2 fixed)

- β accounts for RTT variation.
  - β = 2 can adapt to loads of at most 30 %

Conservation at equilibrium: round-trip timing

UNI FREIBURG

- Problem: Average RTT R and variation in R σ increase quickly with load.
  - Load : ρ → R and σ scale like 1 / (1 - ρ).
  - Network at 75 % capacity: RTT to vary by factor of sixteen (-2σ to +2σ).
- Load above 30 %: retransmission of packets that have just been delayed.
  - Network equivalent of pouring gasoline on a fire.
- Solution: estimate variation instead of using fixed β.
  - Cheap method: use mean deviation mdev (average of | M - R |).
- Resulting timer:

$$R \leftarrow \alpha \cdot R + (1 - \alpha) \cdot M \qquad (\alpha = 0.875 = 1 - 1/8)$$

$$mdev \leftarrow \alpha \cdot mdev + (1 - \alpha) \cdot |M - R| \qquad (1 - \alpha = 0.125 = 1/4)$$

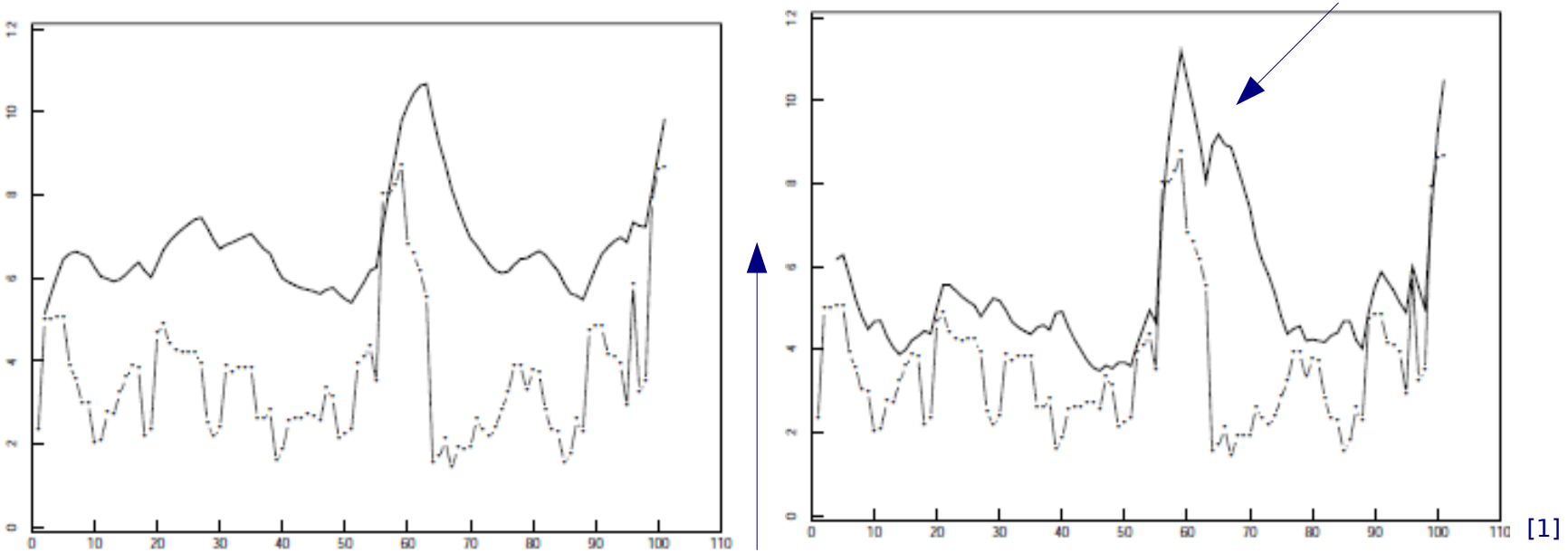
$$rto = R + 4 \cdot mdev$$

Conservation at equilibrium: round-trip timing

- Per packet RTT on well behaved Arpanet connection:

RFC-793 retransmit timer: Mean + Variance:

Behavior of retransmit timer



[1]

Packet number

Elapsed Time from packet sent to ACK receipt

Conservation at equilibrium: round-trip timing

- Second most important timer mistake: backoff after restransmit.
- Packet to be retransmitted more than once: how to space retransmits?
- For transport endpoint embedded in network of
  - unknown topology, unknown, unknowable and constantly changing population of competing conversations.
- Only scheme with any hope of working: exponential backoff.
- Network as linear system (linear operators: delays, gain stages, etc.).
  - Linear system theory: if system is stable, then stability is exponential.
  - Unstable system: network subject to load shocks and congestion collapse.
  - Stabilization: add exponential damping (exp. timer backoff) to primary excitation (senders, traffic sources).

Adapting to the path: congestion avoidance

- If timers good in shape: timeout = lost packet.
  - Lost packet: damaged in transit (<< 1%) or network is congested.

- Two components of “congestion avoidance” strategy:
  - Signal of congestion (delivered automatically: lost packet!).
  - Endnodes action: policy of decrease if signal received, policy of increase if signal not received.

- Network model:
  - Uncongested: $L_i = N$
  - On congestion: $L_i = N + \delta L_{i-1}$

    $L_{i+1} = N + \delta L_i = N + \delta(N + \delta L_{i-1})$

  - Queue lengths increase exponentially.
  - Stabilization: traffic sources must throttle back as quick as queues grow!

Adapting to the path: congestion avoidance

- Endnode action on congestion:
  - Multiplicative decrease of window size.
  - Window adjustment: $W_i = dW_{i-1}$ $(d < 1) \rightarrow (d = 2)$

- Endnode action on no congestion:
  - Increase bandwidth utilization to find out current limit.
  - Best policy: small, constant changes (additive increase).
  - Window adjustment: $W_i = W_{i-1} + u$ $(u << Wmax) \rightarrow (u = 1\ Packet)$

- Congestion control algorithm: additive increase, multiplicative decrease:
  - On ACK: cwnd $\leftarrow$ cwnd + 1/cwnd
  - On timeout: cwnd $\leftarrow$ cwnd/2
  - When sending: send minimum of cwnd and receiver´s advertised window.

Adapting to the path: congestion avoidance

- Combined slow-start with congestion avoidance algorithm:
  - slow-start / congestion window: cwnd
  - threshold size: sstresh
  - When sending: send min. of cwnd and receiver´s advertised window.

  - Start: cwnd ← 1 Packet
    sstresh ← receiver´s advertised window

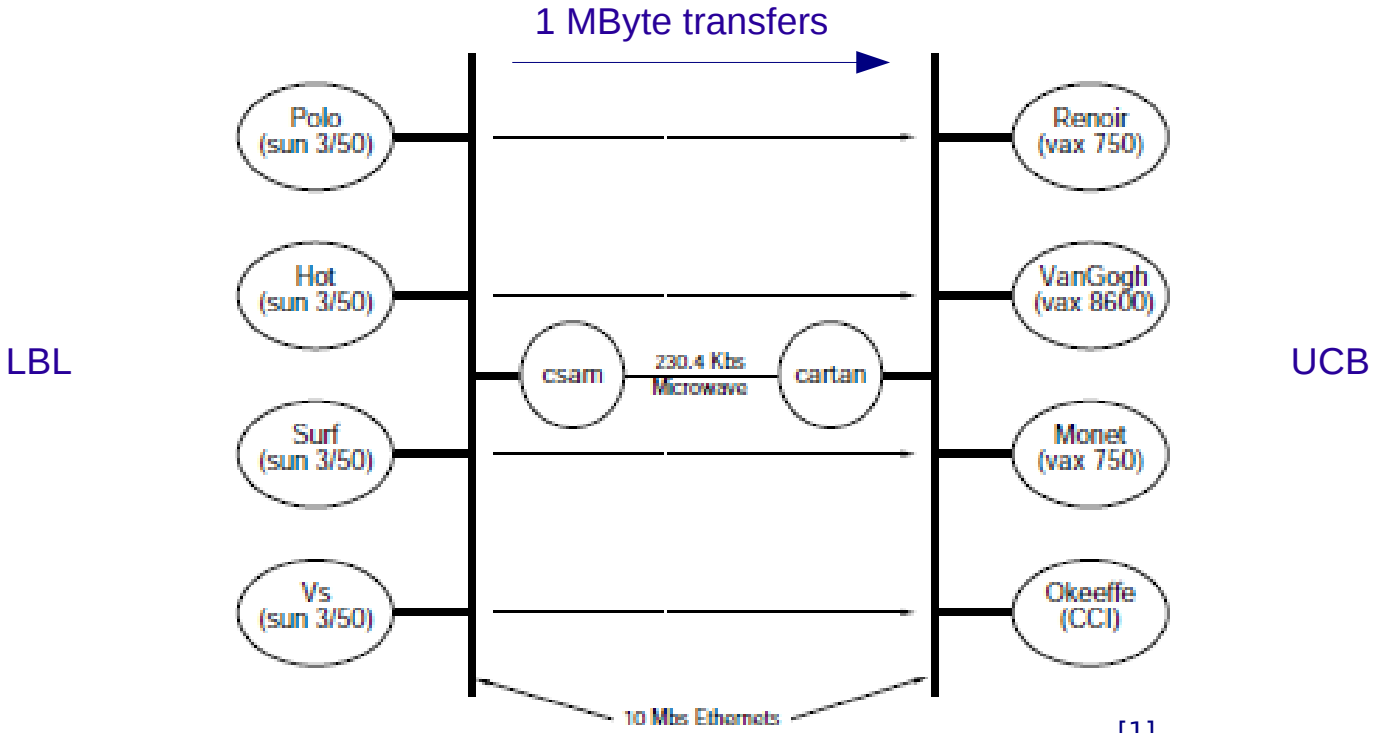
  - On timeout: sstresh ← cwnd / 2
    cwnd ← 1 Packet

  - On ACK: if (cwnd < sstresh)
    cwnd += 1

    else
    cwnd += 1/cwnd

Adapting to the path: congestion avoidance

- Test setup to examine interaction of multiple, simultaneous TCP conversations sharing a bottleneck link:

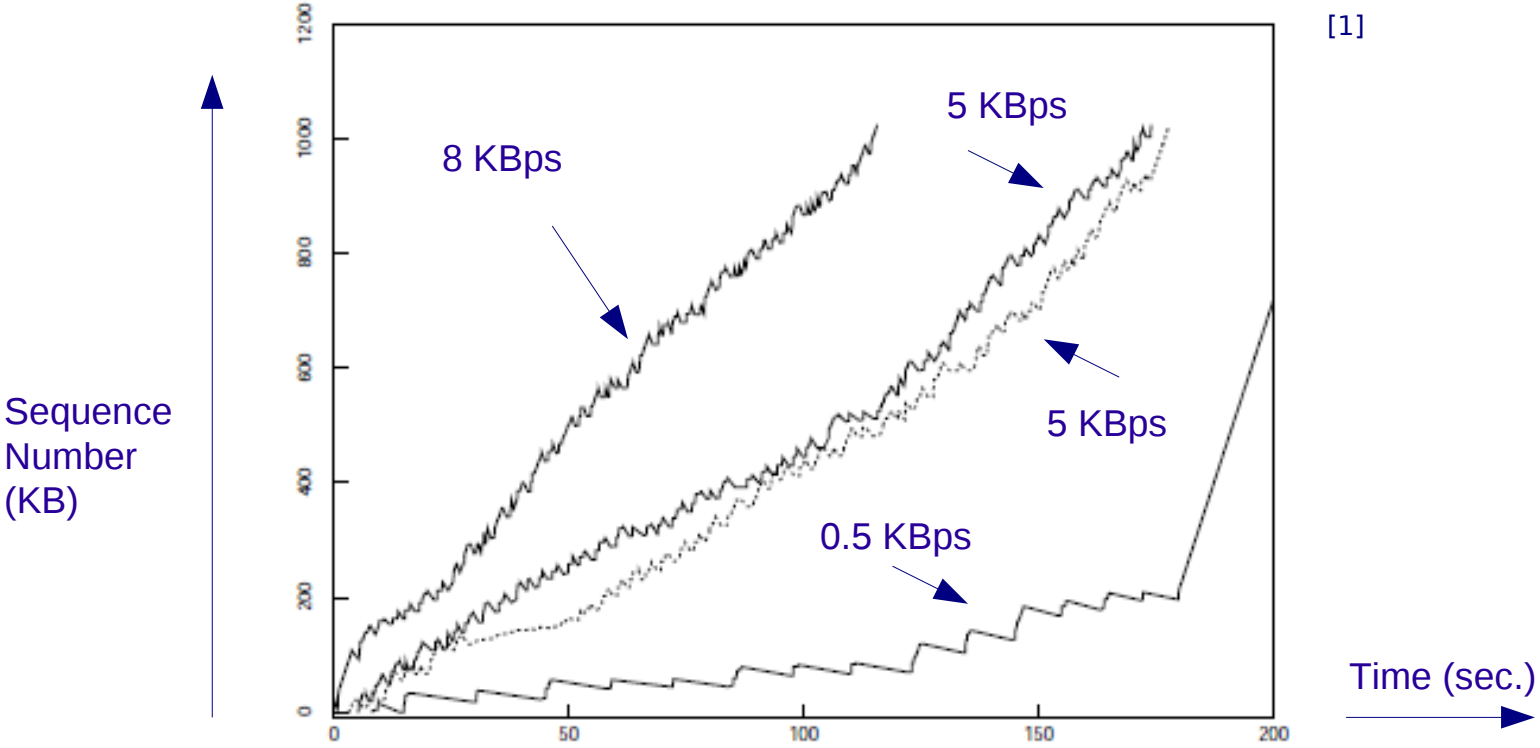


[1]

- Any two connections could overflow the available buffering.
- All four connections exceeded queue capacity by 160 %

Adapting to the path: congestion avoidance

- Simultaneous TCP conversations without congestion avoidance:

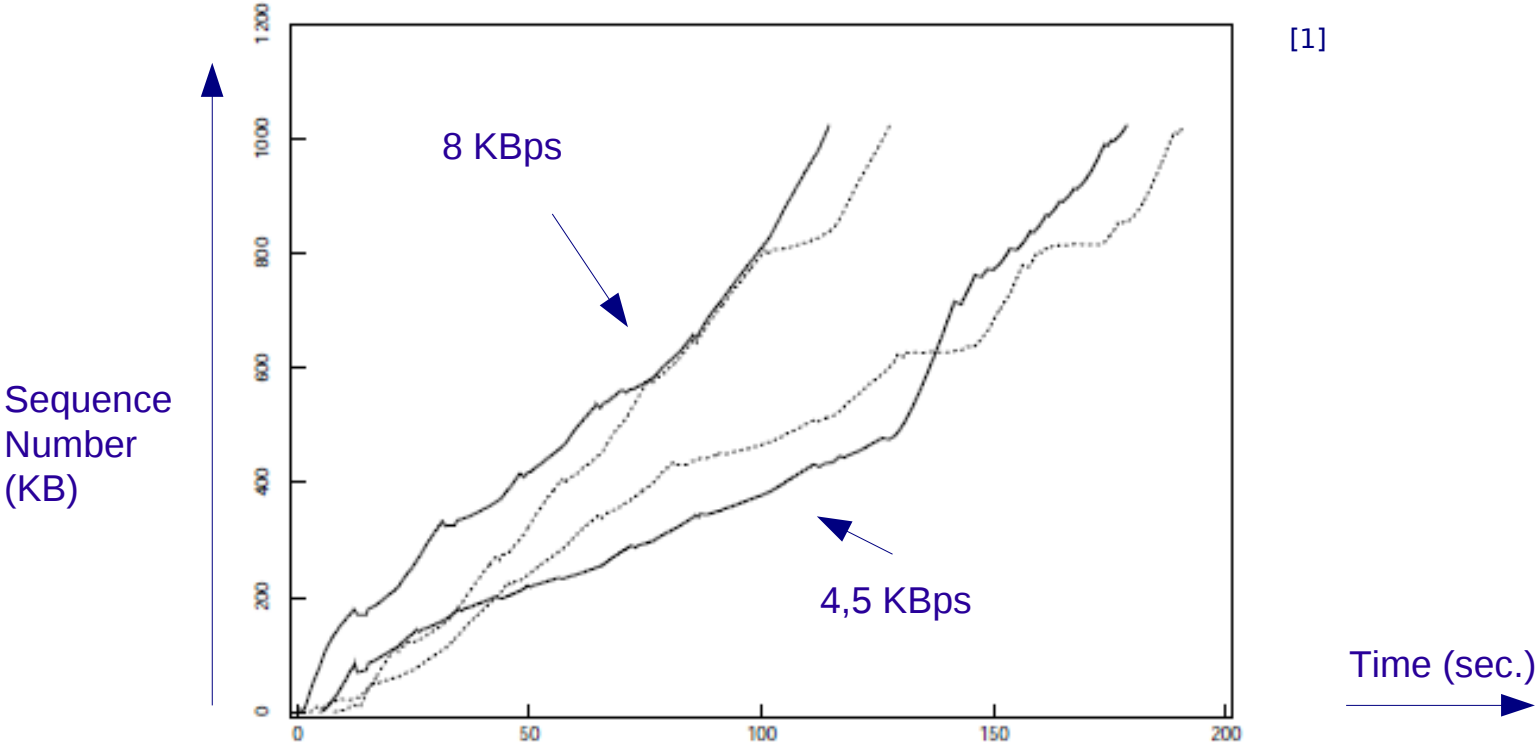


[1]

- 1 MByte transfers, each initiated 3 sec. apart.
- 4000 of 11000 packets sent were retransmissions.
- Link data bandwidth: 25 KBps (6 KBps vanished!).

Adapting to the path: congestion avoidance

- Simultaneous TCP conversations with congestion avoidance:

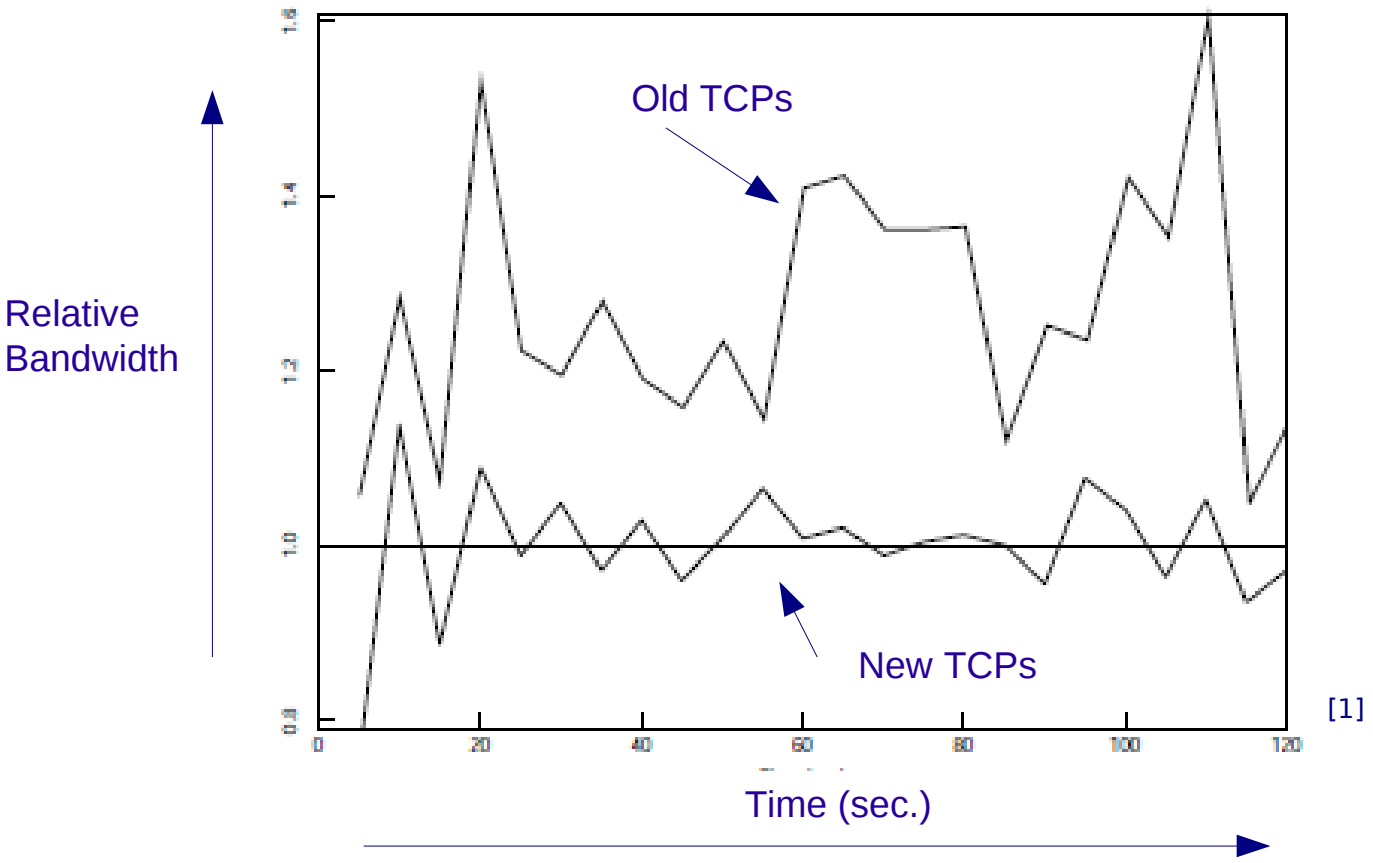


[1]

- 89 of 8291 packets sent were retransmissions (1 %).
- 4,5 KBps: 4.3 BSD receivers. Loss rate: 1,8 %.
- 8 KBps: 4.3+ BSD receivers. Loss rate: 0,5 %.

Adapting to the path: congestion avoidance

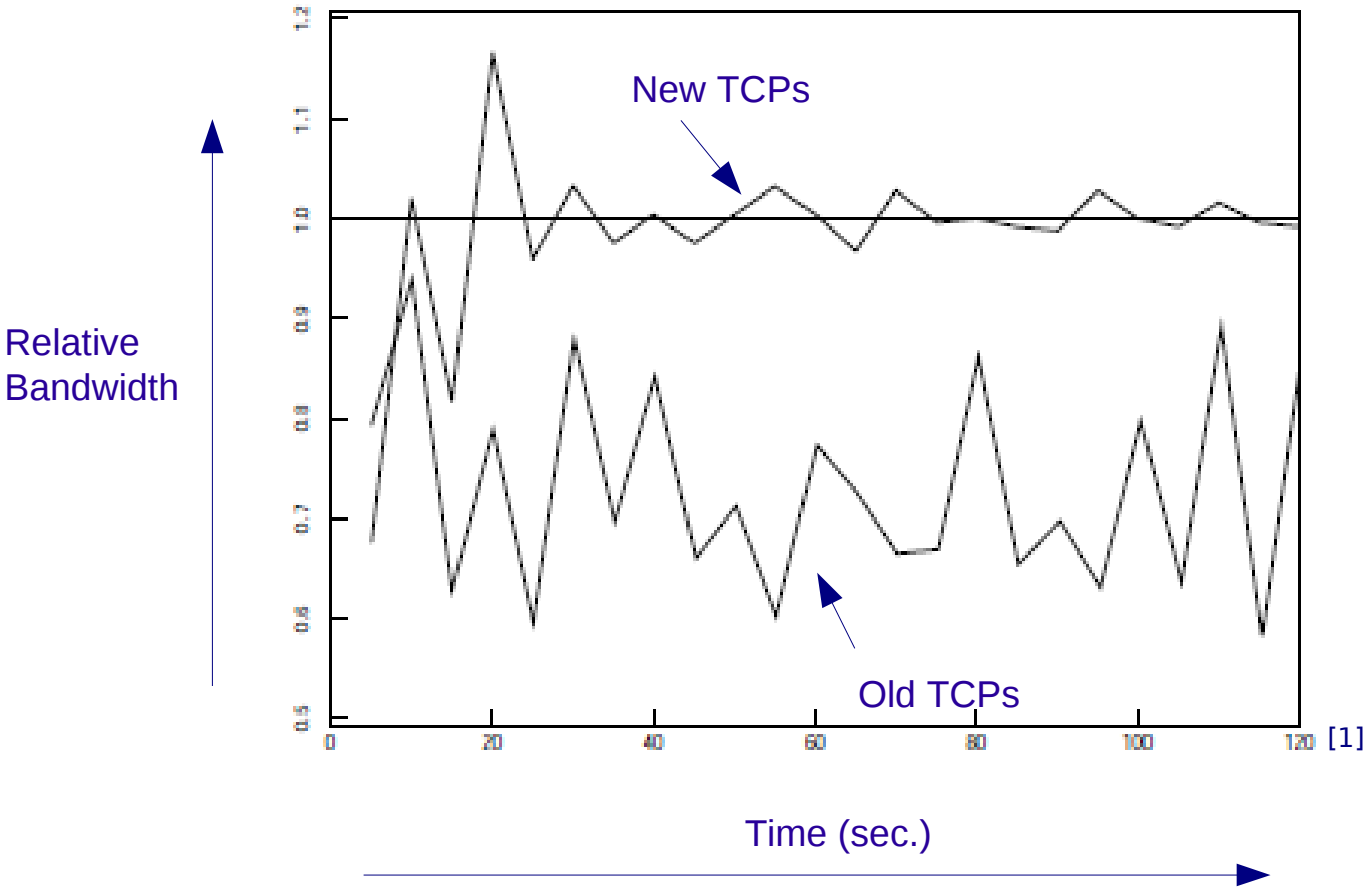
- Total bandwidth used by old and new TCPs:



[1]

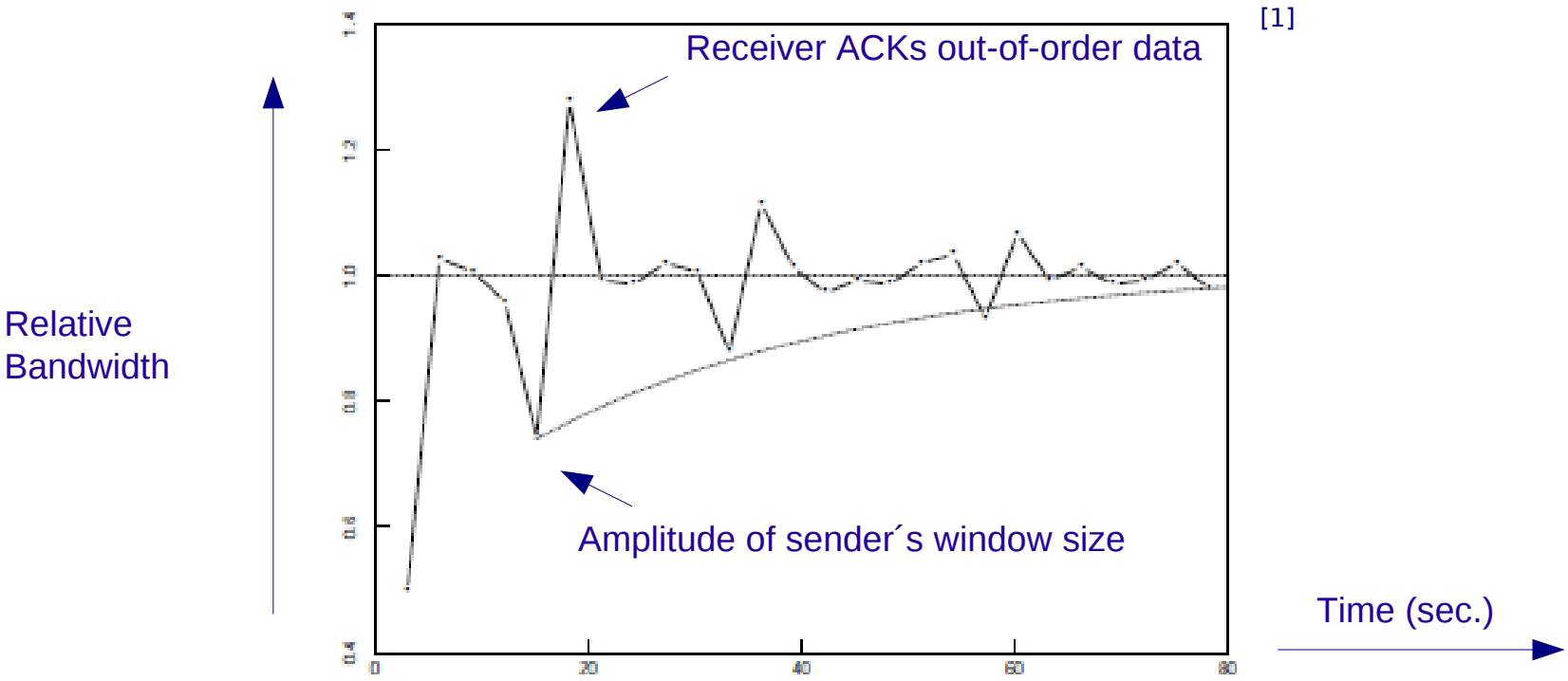
Adapting to the path: congestion avoidance

- Effective bandwidth used by old and new TCPs:



Adapting to the path: congestion avoidance

- Effective troughput for new TCPs. Window adjustment detail:



- When packet dropped: sender sends until window filled, then stops until rto.
- Receiver cannot ACK data beyond dropped packet.
- Spikes height: direct measure of sender´s window size (exponential decrease).

Summary

- Getting to equilibrium.
  - Self-clocking systems.
  - Connection starting or restarting: slow-start.
- Conservation at equilibrium.
  - Round-trip-timing: Mean + Variance timer.
  - Backoff after retransmit: Exponential timer backoff.
- Adapting to the path.
  - Congestion avoidance and control.
  - Additive increase, multiplicative decrease.

4) Future work: the gateway side of congestion control

UNI FREIBURG

- TCP extensions at endpoints insure network capacity is not exceeded.
  - Only in gateways: enough information to also insure fair sharing.

- Next big step: gateway “congestion detection” algorithm:
  - Send signal to endnodes as early as possible (packet drops).
  - Gateway “self-protection” from misbehaving hosts: drop hosts packets.
  - Congestion reduced even without congestion avoidance at endnodes.

- Congestion grows exponentially.
  - Early detection important. Otherwise massive adjustments necessary.
  - Reliable detection non-trivial problem due to bursty nature of traffic.
  - Use models for round-trip-time/queue length prediction as basis of detection.

5) References

[1] Van Jacobson. Congestion Avoidance and Control.
In ACM SIGCOMM Computer Communication Review, volume 18, pages 314-329. ACM, 1988

[2] "Van Jacobson: 2002 IEEE Koji Kobayashi Computers and Communications Award Recipient"
http://www.ieee.org/about/awards/bios/kobayashi_recipients.html#sect10

[3] 2001 SIGCOMM Award for Lifetime Achievement to Van Jacobson
http://www.sigcomm.org/awards/sigcomm-awards

[4] 2012 Inductees, Internet Hall of Fame Innovator
http://www.internethalloffame.org/inductees/van-jacobson

[5] Berkeley Software Distribution
http://en.wikipedia.org/wiki/Berkeley_Software_Distribution#cite_note-iw-5

[6] Unix Guru Universe. Unix contributors: Mike Karels.
http://www.ugu.com/sui/ugu/showclassic?I=info.Mike_Karels&F=11uemijss&G=Y

[7] Babcock, Charles (2006-08-14).
"What's The Greatest Software Ever Written?". InformationWeek.
http://www.informationweek.com/shared/printableArticle.jhtml?articleID=191901844