Congestion Avoidance and Control

Van Jacobson, Michael J. Karels

Presented by

Naveen Cherukuri

Overview

- Introduction
- Getting to equilibrium
- Conservation at equilibrium
- Adapting to the path
- Experimental evaluation
- Future work

Introduction

- Explosive growth of Networks
  - Gateway- Local buffer overflows
  - Cause lies in transport protocol implementation

- First Congestion collapse observed
  - October 1986, LBL to UCB
  - Data throughput dropped from 32Kbps to 40bps

Packet Conservation

- Packet Conservation Principle
  - 5 algorithms developed based on this
    1. Round-trip-time variance estimation
    2. Exponential retransmit timer backoff
    3. Slow-start
    4. More aggressive receiver ack policy
    5. Dynamic window sizing on congestion

  - Connection in equilibrium – Running stably with a full window of data in transit.

  - Conservative packet flow – Flow at equilibrium when " A new packet isn't put into the network until an old packet leaves.

Failures

- Three ways for "how packet conservation fails"

1)The connection doesn't get to equilibrium.

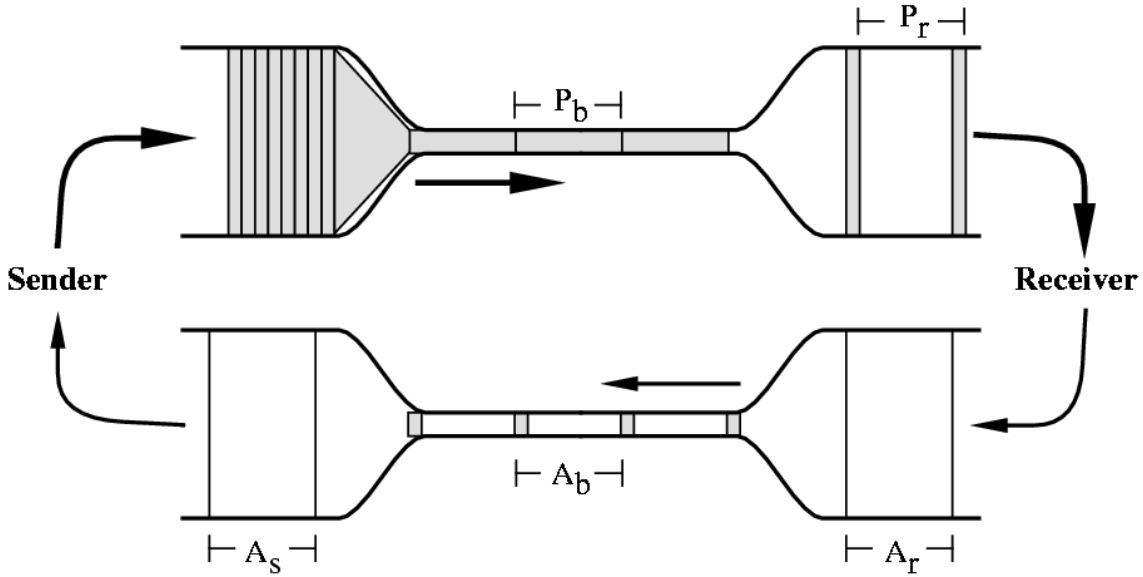
2)A sender injects a new packet before an old packet has exited.

3)The equilibrium can't be reached because of resource limits along the path.

Getting to Equilibrium: Slow-start

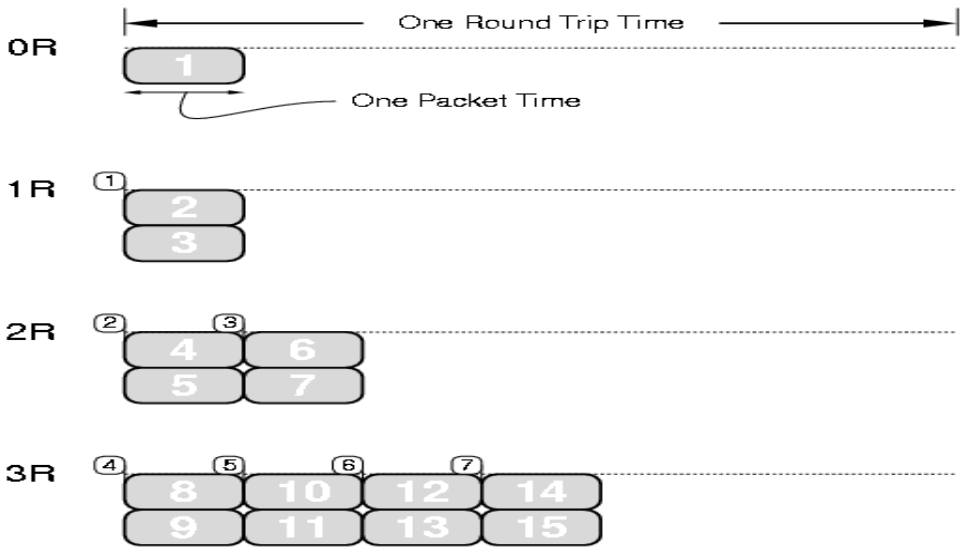
- Connection Starting or Restarting

- Self-Clocking Systems – adjust to bandwidth and delay variations



- To get data flowing acks to clock are required and to get acks there must be data flowing
  - Problem – How to start Clock
  - Solution – Slow-start Algorithm to gradually increase the amount of data-in-transit

Figure 2: The Chronology of a Slow-start



Slow-start Algorithm

- Add a congestion window, *cwnd*, to the per-connection state.
- When starting or restarting after a loss, set *cwnd* to one packet.
- On each ack for new data, increase *cwnd* by one packet
- When sending, send minimum of the receiver's advertised window and the *cwnd*.

Figure 3: Startup behavior of TCP without Slow-start

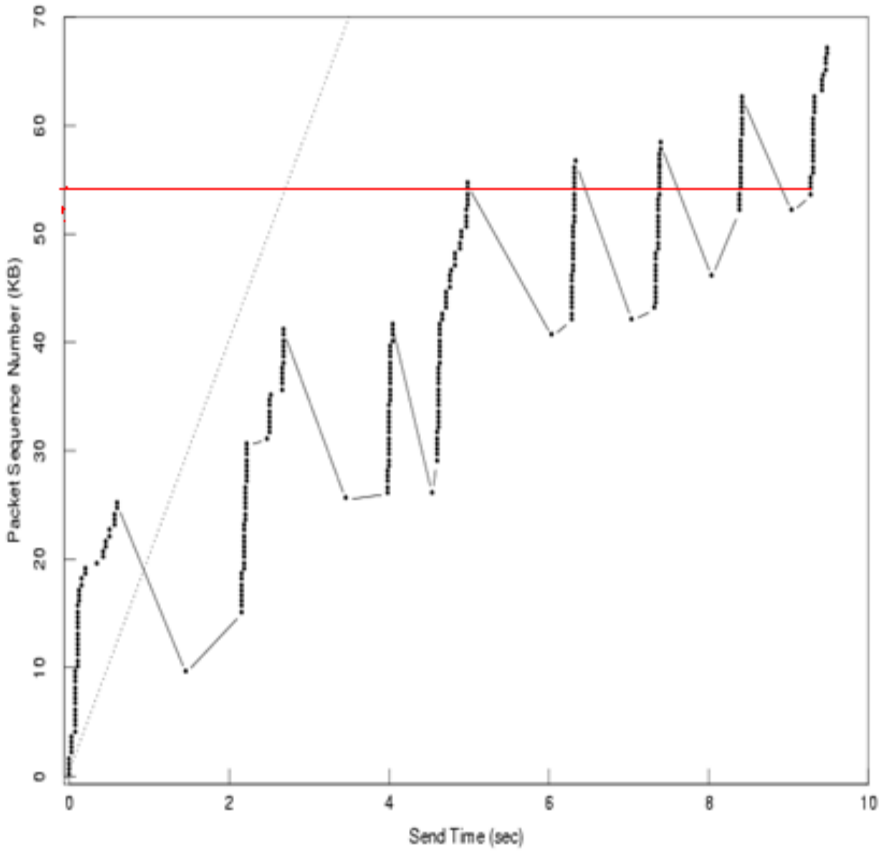
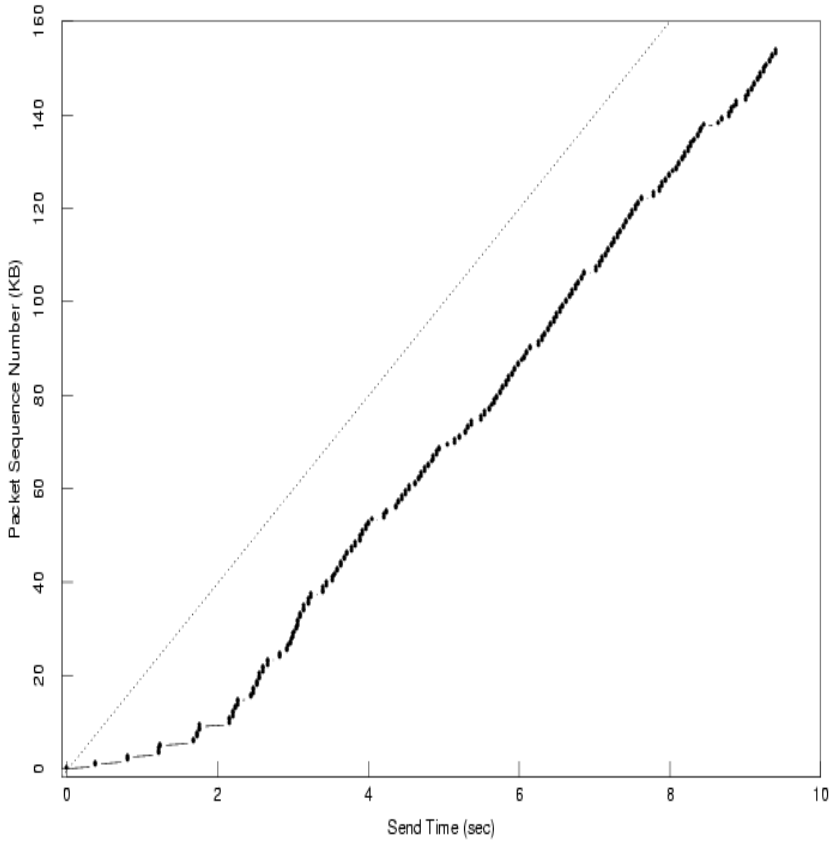


Figure 4: Startup behavior of TCP with Slow-start



- Analysis
  - Takes $R\log_2 W$ time to get to equilibrium where R – Round trip time and W – window size in packets.
  - Guarantees that a connection will source data at a rate at most twice the maximum possible on the path.

Conservation at equilibrium: Round-trip timing

- Failure of sender's retransmit timer.
- A good round trip time estimator – the core of the retransmit timer is the most important feature to survive heavy load.
- TCP estimates mean round trip time using

$$R = \alpha R + (1- \alpha)M$$

M – round trip time measurement from the most recently acked data packet.

α – Filter gain constant with suggested value of 0.9

- Retransmit timeout interval(rto) for the next packet sent is set to βR
- β accounts for RTT variation
- They argue that β should also be estimated to improve performance at low load as well as high load.
- Retransmit timer – Exponential back-off

Figure 5: Performance of an RFC793 retransmit timer

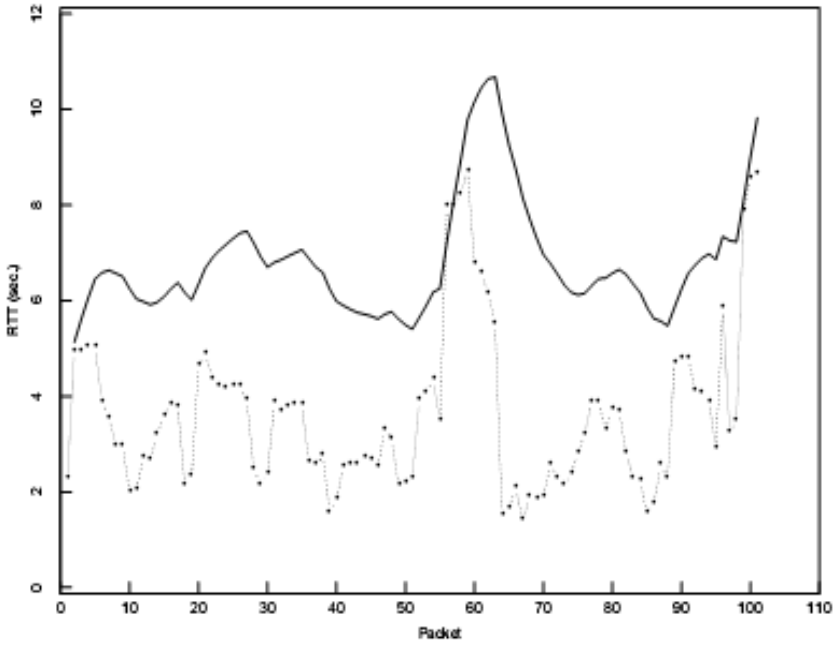
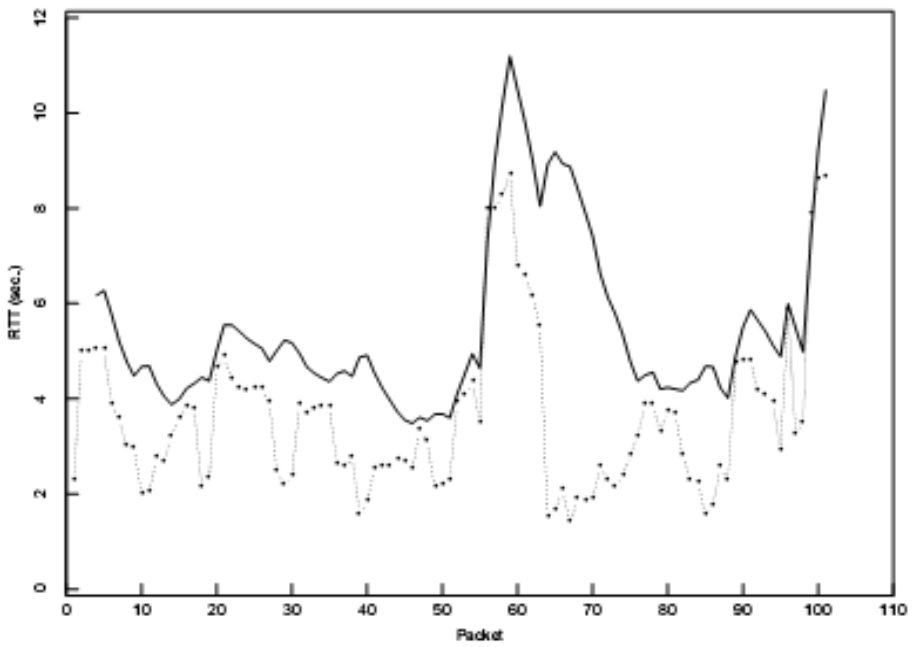


Figure 6: Performance of a Mean+Variance retransmit timer



Adapting to the path: congestion avoidance

- Packets get lost for two reasons
  1. They are damaged in-transit
  2. Network is congested.

- Congestion avoidance strategy
  - ✓ Network must signal to transport endpoints that congestion is occurring.
  - ✓ The endpoints must decrease or increase utilization based on whether the signal is received or not.

Congestion Window adjustment

- Good candidate for 'network is congested' signal?
- Network load is measured by average queue length over fixed intervals of length near the roundtrip time.
- $L_i = N$ (uncongested network)
- $L_i = N + \gamma L_{i-1}$ (congested network)
- γ indicates the amount of congestion
- Source controls its loads by adjusting its window.

  $W_i = dW_{i-1}$ $(d < 1)$ (On congestion)

Congestion Window adjustment

- Network doesn't indicate if a connection is using less than its fair share.
- Solution?
- How about $W_i = bW_{i-1}$ $1<b<=1/d$??
  - Rush hour effect
- Hence $W_i = W_{i-1} + u$ $(u << W_{max})$ (On no congestion)

Congestion Avoidance Algorithm

- On any timeout, set ***cwnd*** to half the current window size (Multiplicative decrease)

- On each ack for new data, increase cwnd by 1/cwnd (Additive increase)

- When sending, send the minimum of the receiver's advertised window and cwnd.

Experimental evaluation

Figure 7: Multiple conversation test setup

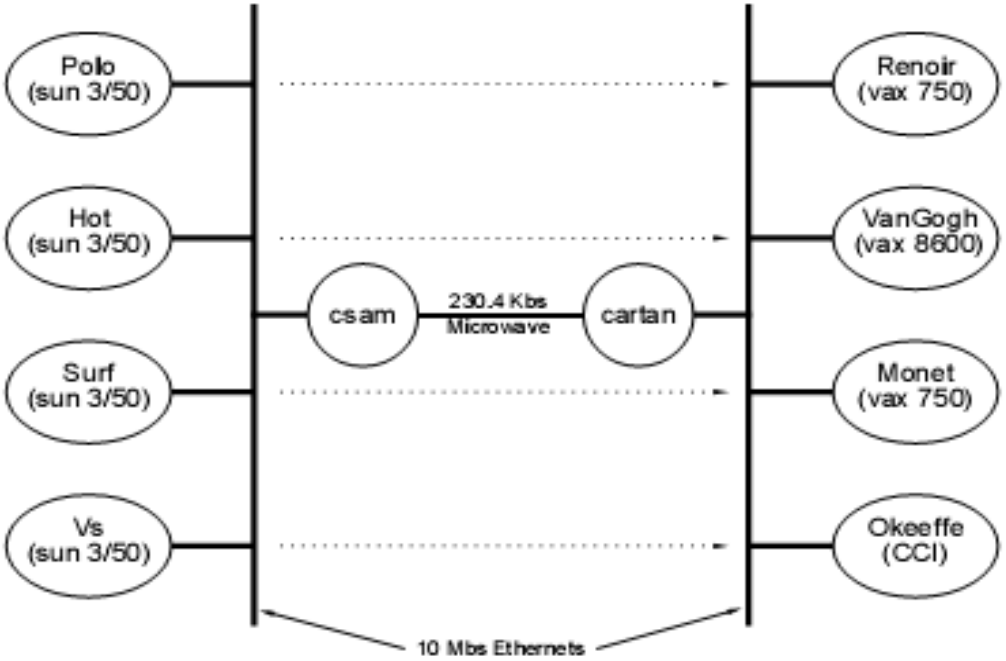


Figure 8: Multiple, simultaneous TCPs with no congestion avoidance

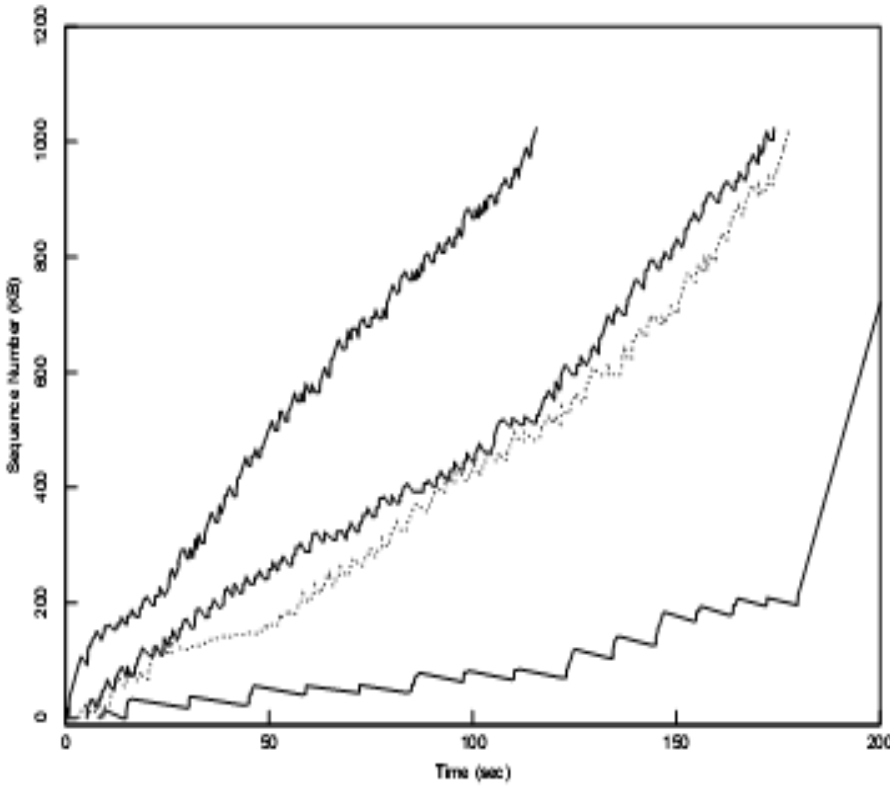
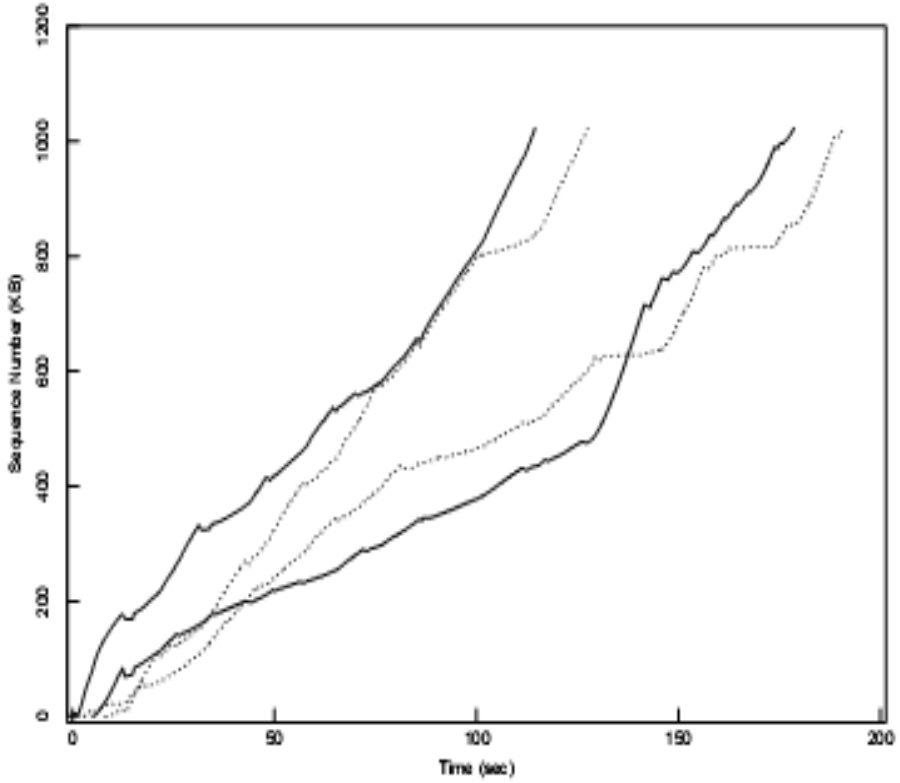


Figure 9: Multiple, simultaneous TCPs with congestion avoidance



Future Work : The gateway side of congestion control

- How to ensure "Fair sharing of the connection's capacity"?

- Solution - Congestion detection algorithm at the gateways.
  - Misbehaving endpoints that do not consider the request to decrease the window size can be taken care of.

Thank you

Questions and Suggestions ??