Cornell University
Computer Systems Laboratory

AN O/S PERSPECTIVE ON NETWORKS

Xiaodong Wang

10/10/2013

The history of Internet

- **First digital communication system: telegraph!**
- **Fundamental theory:**
  - Information theory by Claude Shannon, Harry Nyquist, and Ralph Hartley
- **Early computers are able to communicate over long distances --- physical links required**
  - Do not allow direct communication between arbitrary systems
  - Susceptible to enemy attack!

Source: wikipedia.org

- **Call for global network**
  - A mandate to interconnect the United States Department of Defense's main computers at Cheyenne Mountain, the Pentagon, and SAC HQ
  - Fundamental pioneer: Joseph Carl Robnett Licklider
  - Three network terminals

Source: wikipedia.org

- **Packet switching**
  - By Paul Baran, Donald Davies, Leonard Klenirock
  - Better bandwidth utilization, faster response
  - Be able to resist a nuclear "holocaust"
- **ARPANET**
  - Oct. 29, 1965: the first ARPANET link was established
  - Led by IPTO of DARPA
  - Request for Comments (RFC): official docs for Internet specs

> "We set up a telephone connection between us and the guys at SRI ...", Kleinrock ... said in an interview: "We typed the L and we asked on the phone,
>
> "Do you see the L?"
> "Yes, we see the L," came the response.
> We typed the O, and we asked, "Do you see the O."
> "Yes, we see the O."
> Then we typed the G, and the system crashed ...
>
> Yet a revolution had begun" ....[10]

Source: wikipedia.org

The history of Internet

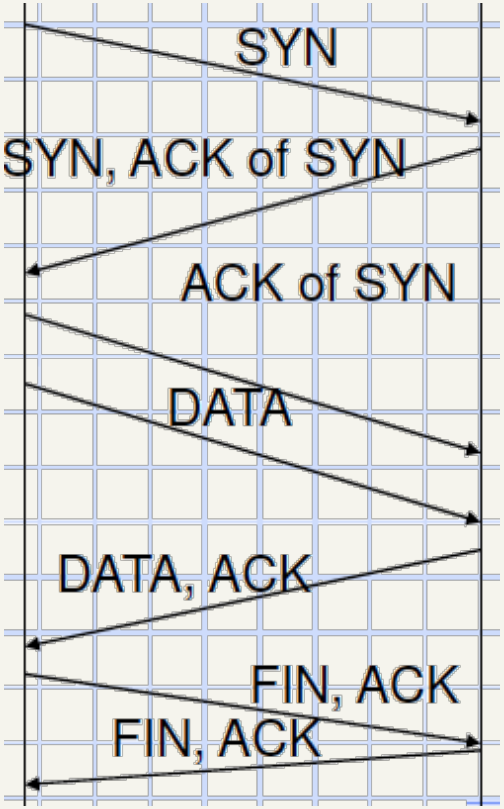
- **Packet switching networks since ARPANET**
  - NPL, Merit Network, CYCLADES, X.25, UUCP
- **TCP: first spec comes out in 1974**
  - Unify all the different networks!
  - The role of the network reduced to the bare minimum --- end to end argument (1984) applies
  - TCP/IP protocols became the only approved protocol on the ARPANET, replacing the earlier NCP protocol on Jan. 1, 1983

Source: wikipedia.org

- **NSFNET**
  - Aimed to create an academic research network facilitating access by researchers to the supercomputing centers funded by NSF in the United States
  - TCP/IP based
  - 56kbits/s (1986) → 1.5Mbits/s (1988) → 45Mbits/s (1991)
  - Decommissioned in 1995 when it was replaced by backbones operated by several commercial Internet Service Providers.
- **Internet**
  - Any network using TCP/IP
  - Network agnostic

Source: wikipedia.org

- **TCP/IP before Jacobson**
  - The role of the network reduced to the bare minimum
  - Packet-switching network
  - Three-way handshake
  - TCP is split into TCP and IP
    - » IP: routing the packets
    - » TCP: packeting, error control, re-transmission and reassembly



Source: wikipedia.org

Problems of TCP

- **TCP is susceptible to congestion**
  - Oct, 1986: data throughput dropped from 32kbps to 40bps
  - Call for congestion control under abysmal congestion for it to be wide spread
- **TCP is susceptible to malicious attack**
  - RFC is public so that everyone can read it and try to find loopholes
  - Simple changes on receiver side may bring huge benefit while hurting others

- **Van Jacobson**
  - Redesigning TCP/IP's flow control algorithms (Jacobson's algorithm) to better handle congestion is said to have saved the Internet from collapsing in the late 1980s and early 1990s
  - 2001 ACM SIGCOMM Award for Lifetime Achievement
  - National Academy of Engineering
  - Internet Hall of Fame
  - Lawrence Berkeley Laboratory → CISCO →PARC

Source: wikipedia.org

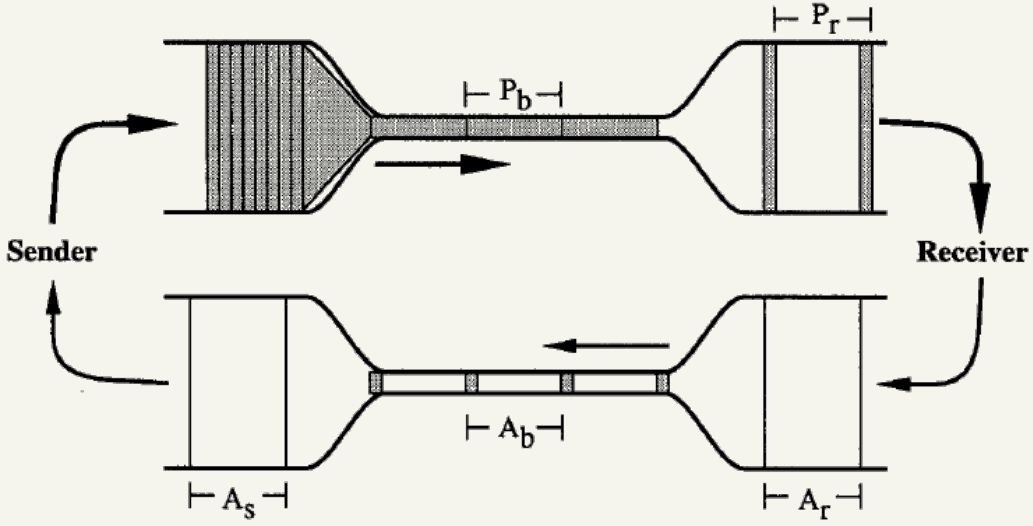
- **TCP before Jacobson**
  - The role of the network reduced to the bare minimum
  - Packet-switching network
  - Three-way handshake
- **TCP after Jacobson**
  - Everything above plus congestion control
    - » Slow start
    - » RTT (round trip time) estimate
    - » Additive increase, exponential backoff

Source: wikipedia.org

- **Conservation of Packets Principle**
  - TCP connection should be in equilibrium
  - Running stably with a full window of data in transit
  - A new packet comes in only when an old one leaves
- **TCP doesn't necessarily obey these principles**
  - Oct. 1986, throughput between LBL and UC Berkeley dropped from 32Kbps to 40bps

- **Three ways to break the principle:**
  - Case I. The connection doesn't get to equilibrium
  - Case II. A sender injects a new packet before an old packet has exited --- doesn't keep equilibrium
  - Case III. The equilibrium can't be reached because of resource limits along the path

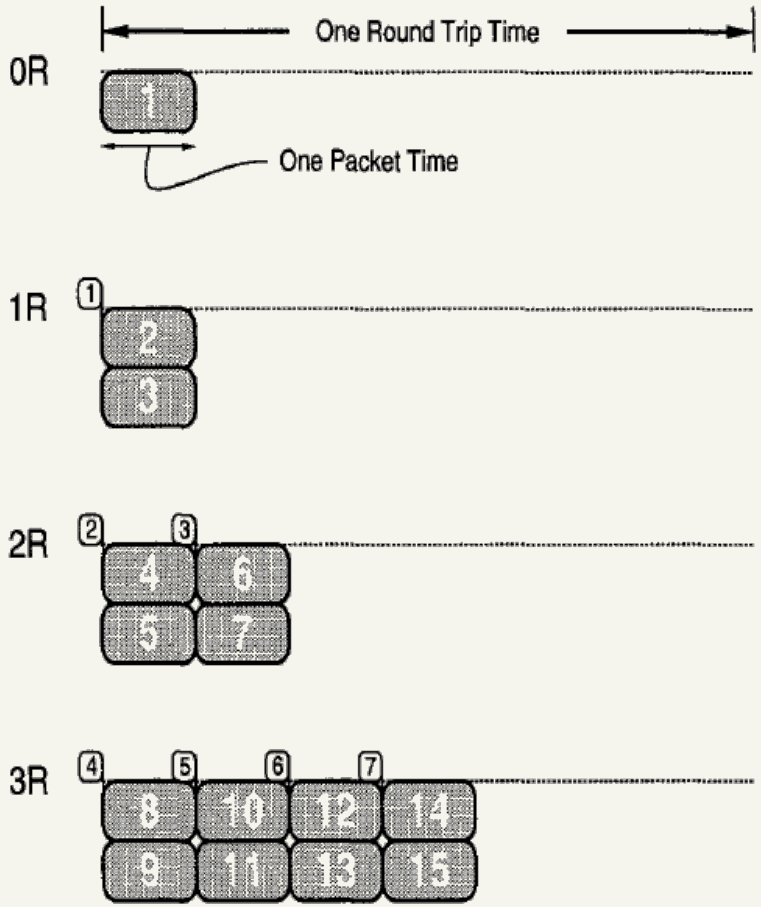
- **Case I. Getting to equilibrium**
  - Self-clocking: using 'ack' as the clock



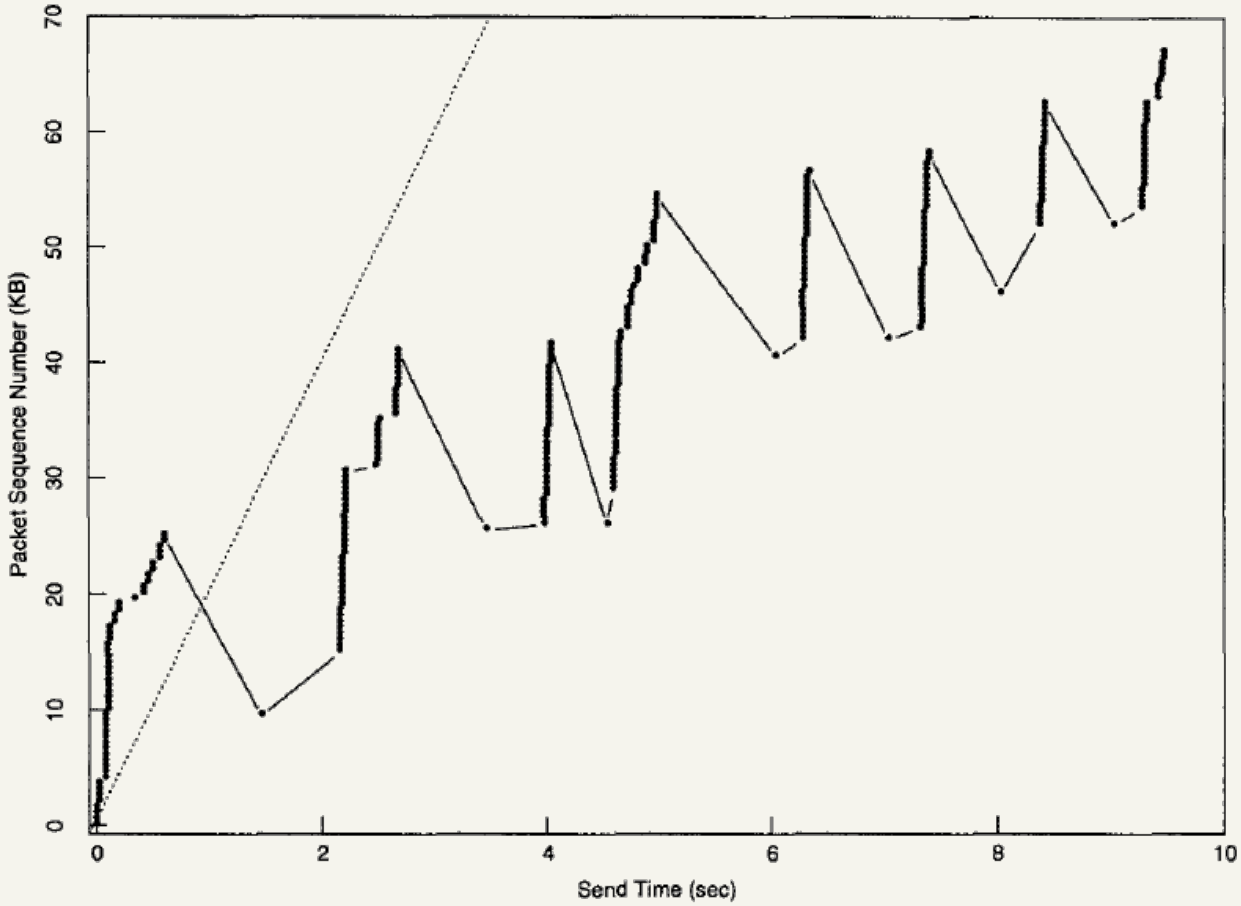
  - How to start the clock

Case I. Getting to equilibrium

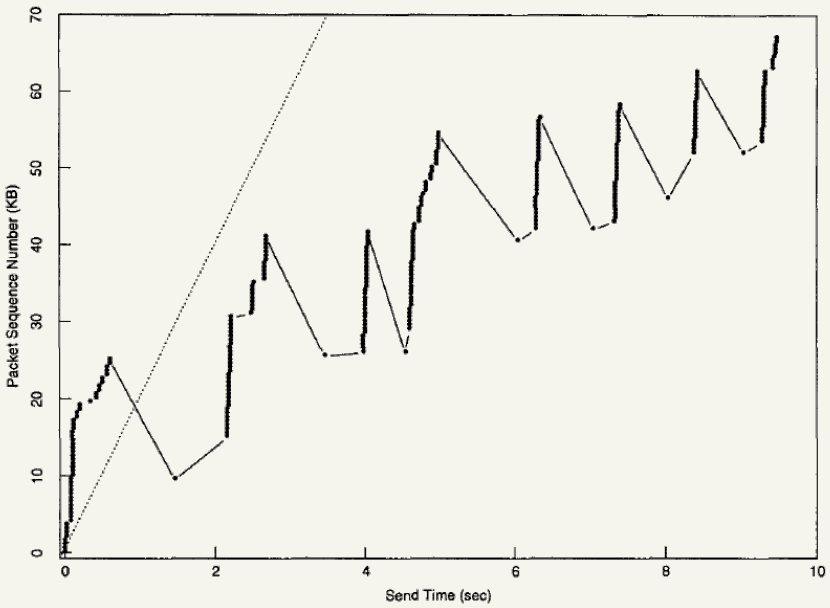
- Slow start to "try" where the equilibrium is
- Congestion window, *cwnd*
- Start with *cwnd*=1
- Increase *cwnd* by 1 for each ack
- Exponential increase: not really slow



- Case I. Getting to equilibrium: w/o slow start

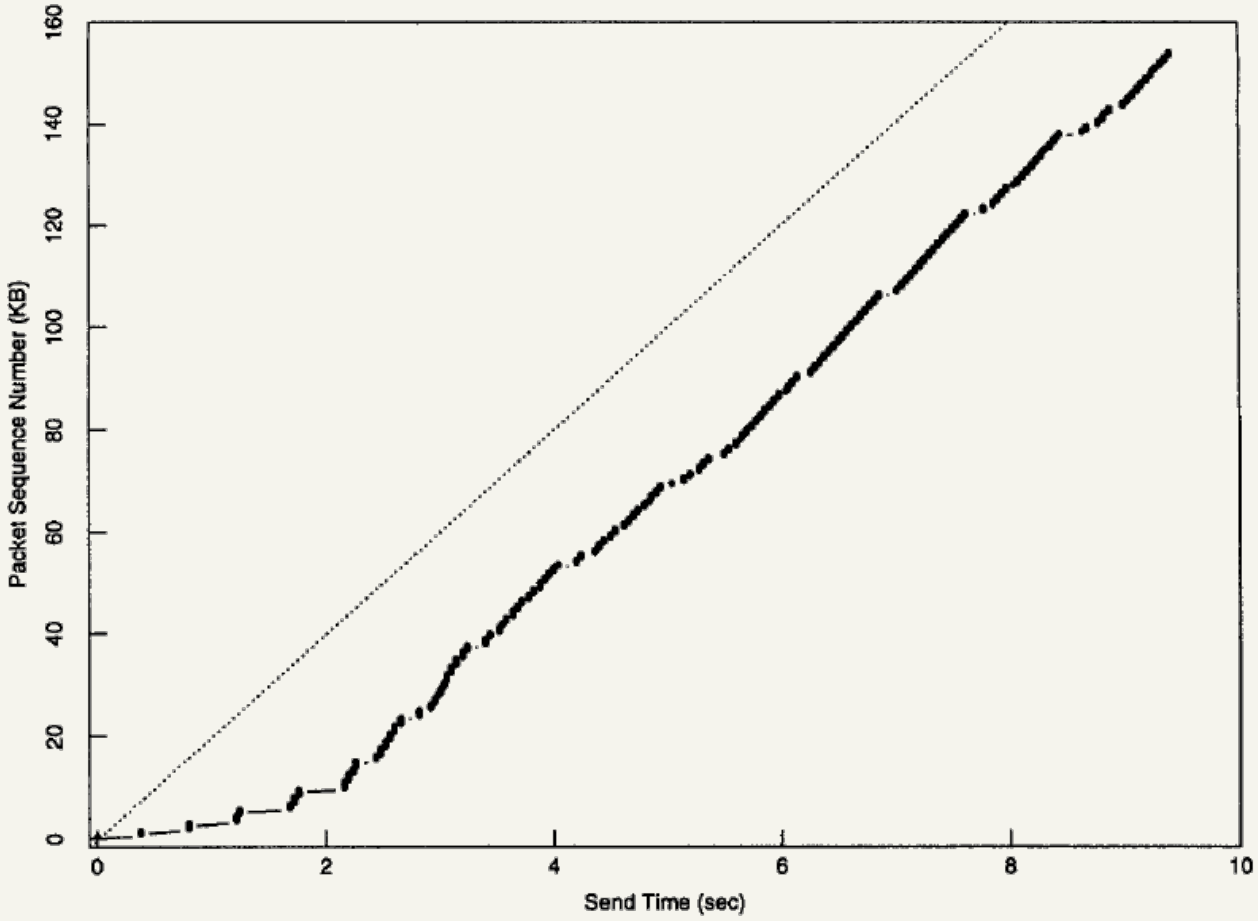


- **Case I. Getting to equilibrium: w/o slow start**

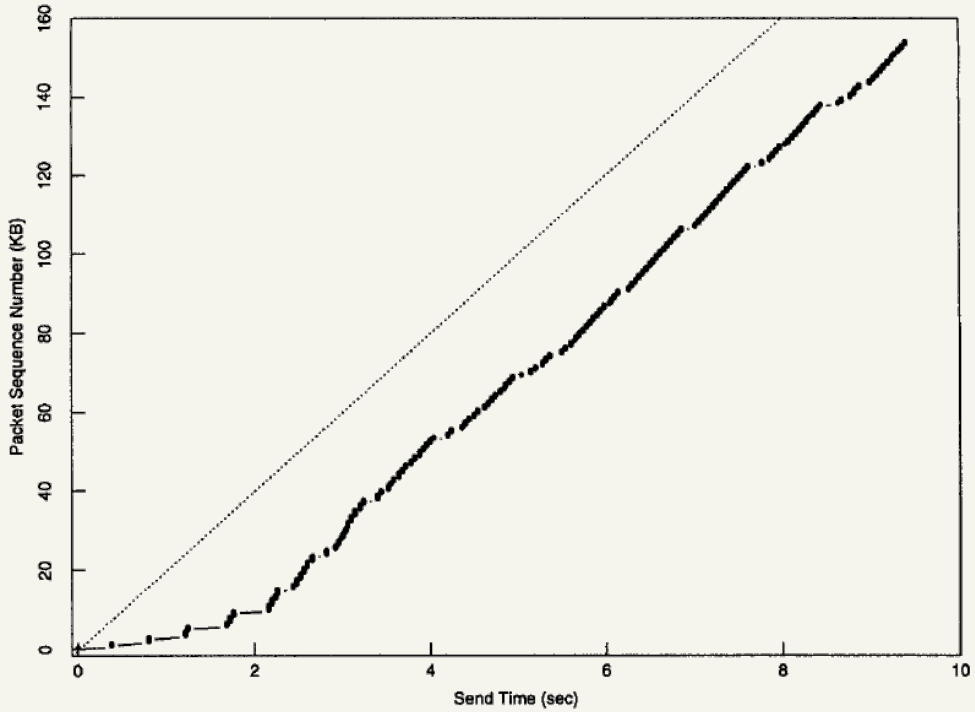


- Almost all packets are retransmitted
- Data from 64 to 58kb is transmitted 5 times
- Only 35% actual bandwidth is used

- Case I. Getting to equilibrium: w/ slow start



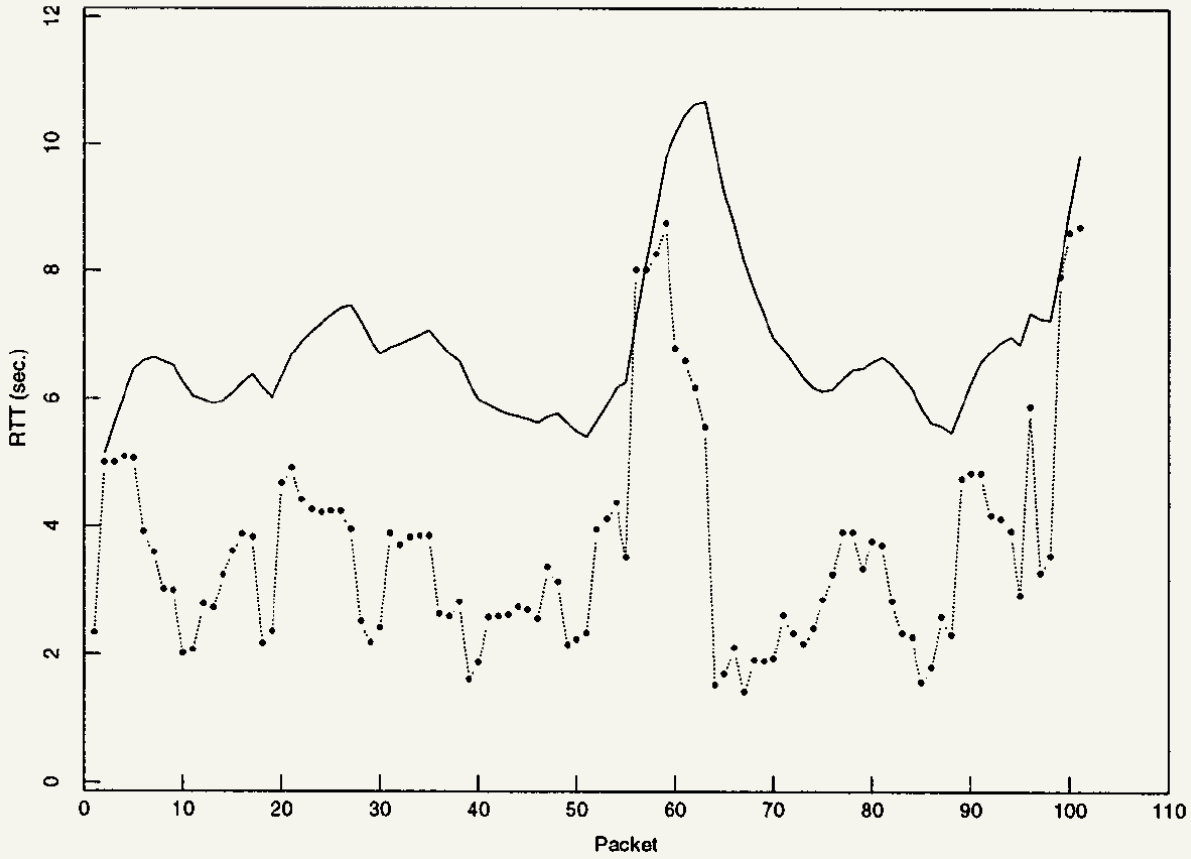
- **Case I. Getting to equilibrium: w/ slow start**



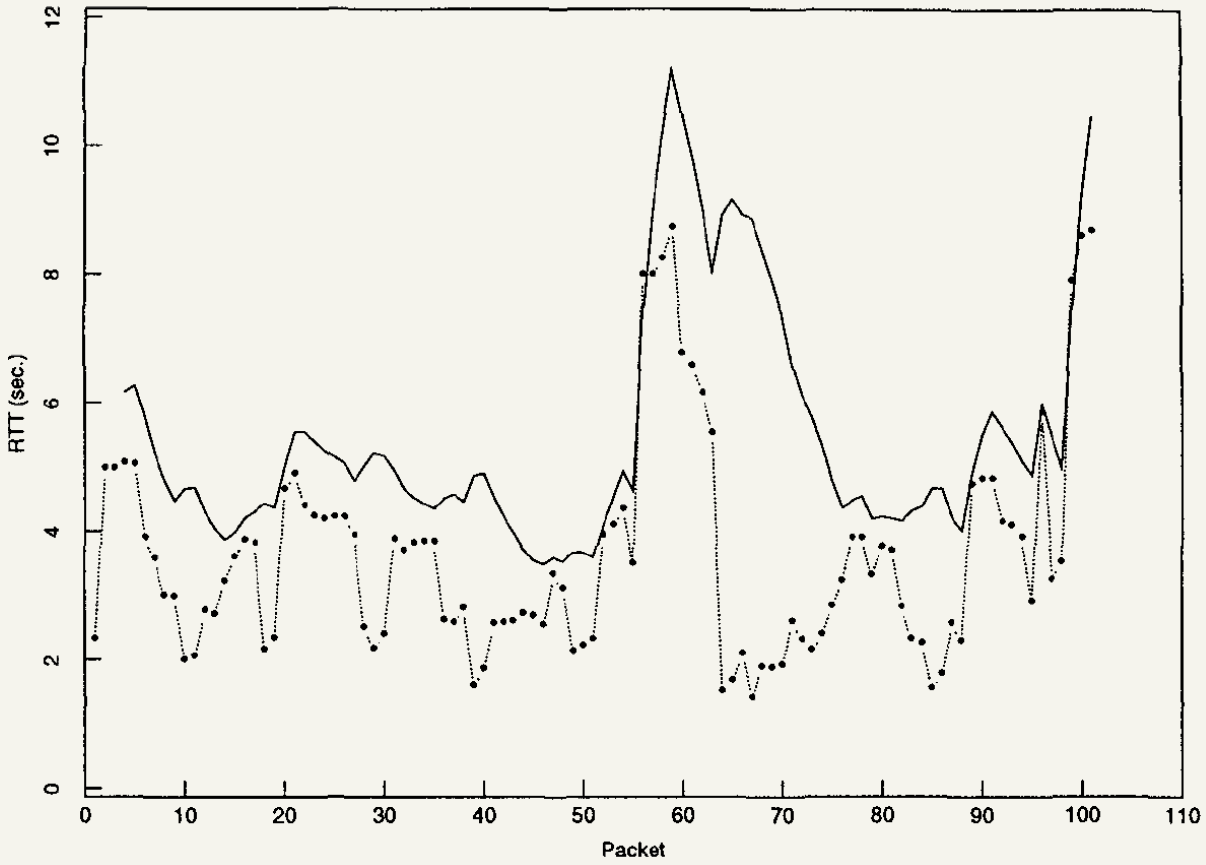
- Window size doubles in burst
- No bandwidth is wasted on retransmission

- **Case II. Round-trip timing for timeout**
  - No "ack" is received
  - Timeout: retransmit the packet
  - [RFC793] $R = \alpha R + (1 - \alpha)M$
    - » Weighted history using α
    - » Must model the variation of R: β
    - » re-transmit timeout interval rto=βR
    - » β=2 only adapt to loads of *at most* 30%

- **Case II. Round-trip timing for timeout**
  - [RFC793] $R = \alpha R + (1 - \alpha)M$



- **Case II. Round-trip timing for timeout**
  - Estimate the variance rather than fixed β=2



- **Case III. Adapting to path**
  - Packets can get lost:
    - » Limited bandwidth
    - » Insufficient buffer
  - Network is responsible to signal the endpoint
  - The endpoint must decrease utilization

- **Case III. Adapting to path**
  - On congestion:
    - » Queue length will increase exponentially
    - » Exponential stability in LTI (linear-time invariant) system
  - Back-off exponentially:

$$cwnd_n = d \cdot cwnd_{n-1}$$

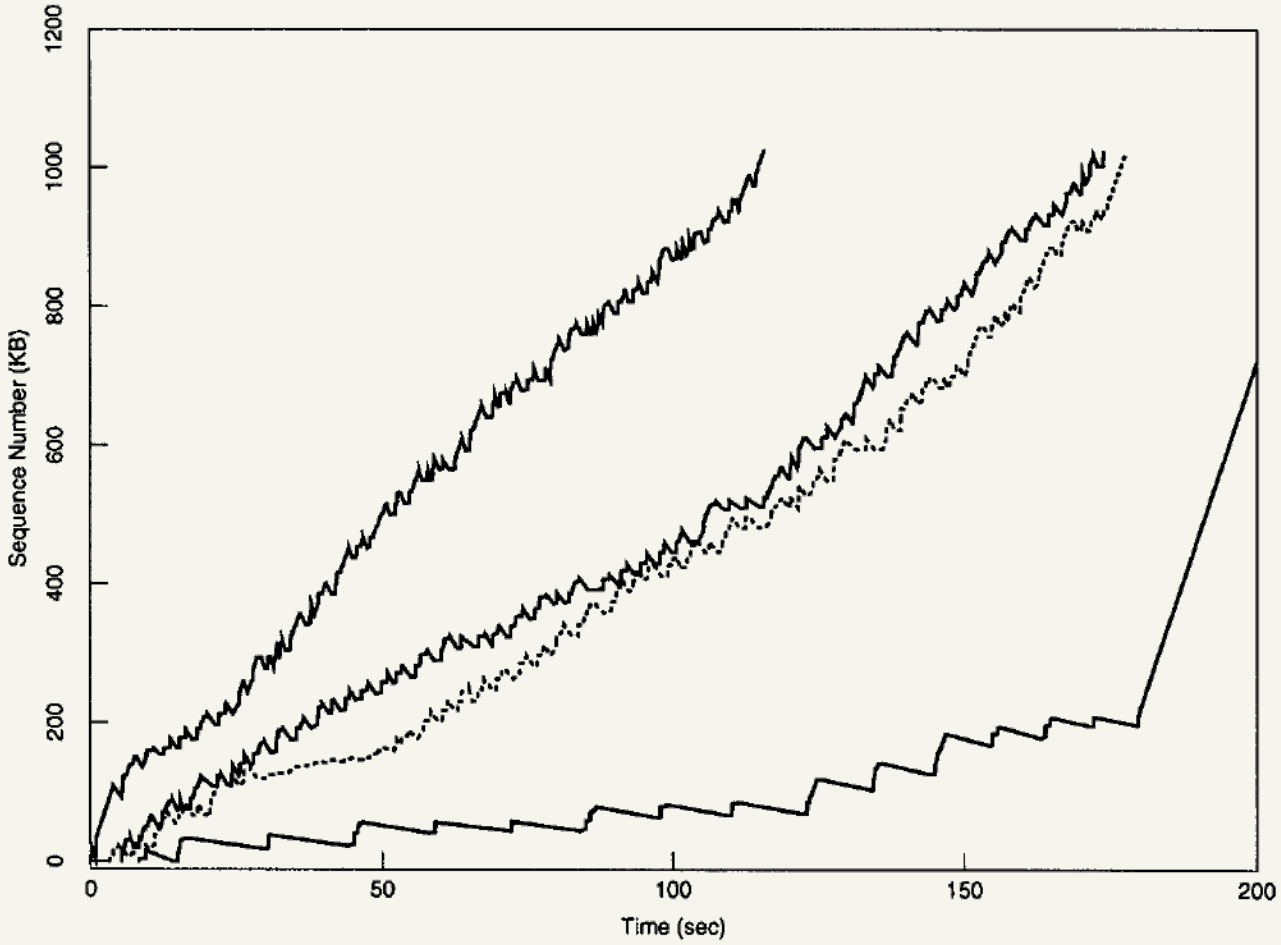
- **Case III. Adapting to path**
  - When no congestion:
    - » Exponential increase window: will tragically oscillate
    - » Additive increase:

$$cwnd_n = cwnd_{n-1} + u$$

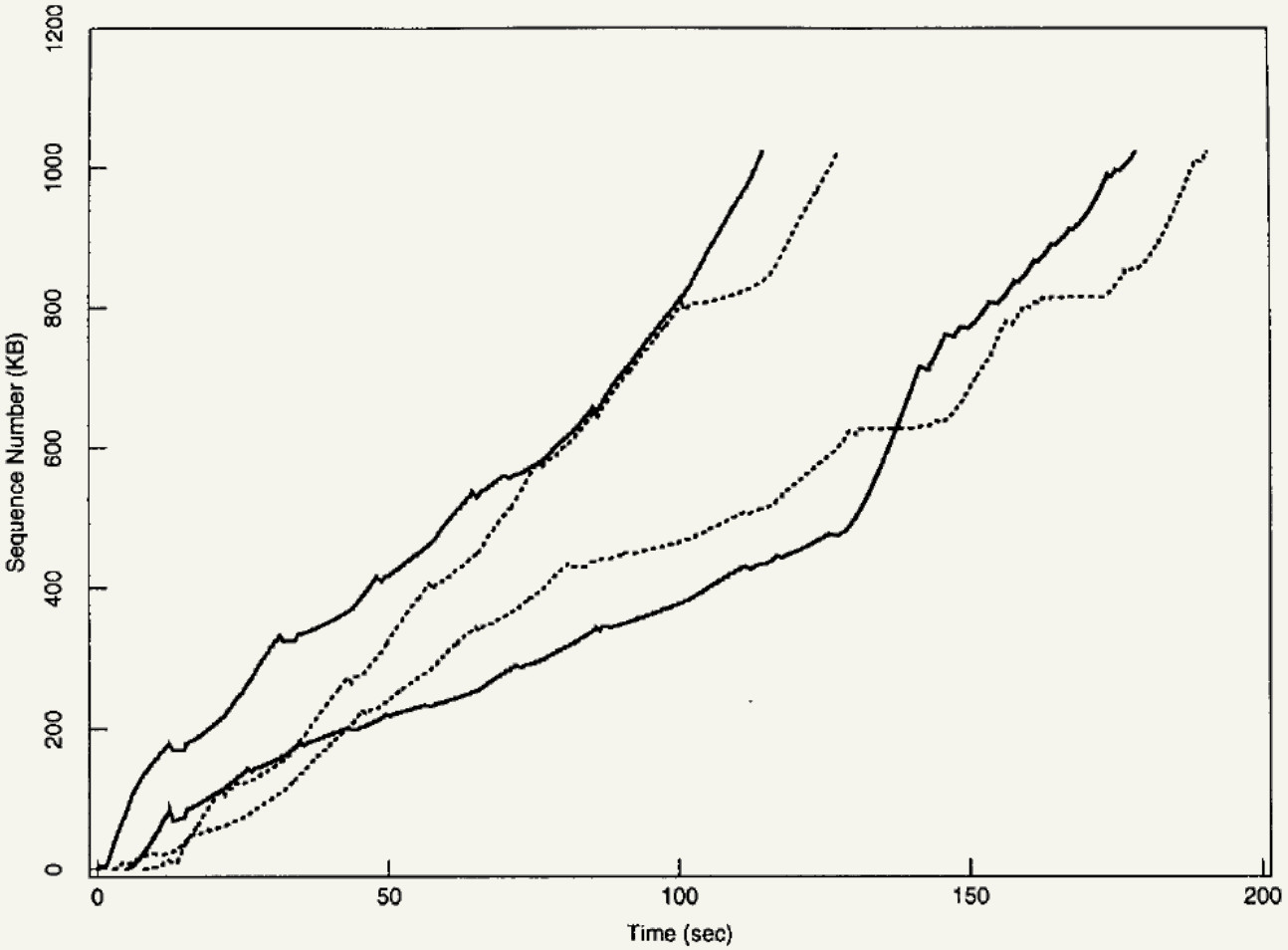
- **Conclusion: 7 changes to TCP**
  - round-trip-time variance estimation
  - Exponential retransmit timer backoff
  - Slow start
  - More aggressive receive ack policy
  - Dynamic window sizing on congestion
  - Karn's clamped retransmit backoff
  - Fast retransmit

- **Putting it altogether**
  - Start with *cwnd* = 1
  - Slow start: Increase *cwnd* by 1 for each ack
  - On a time-out (package loss):
    - » *ssthresh* = *cwnd* / 2
    - » *cwnd* = 1
    - » *cwnd* < *ssthresh*: *cwnd* += 1 for each ack (slow start)
    - » *cwnd* > *ssthresh*: *cwnd* += 1 / *cwnd* for each ack (additive increase)

- **Results**


Sequence Number (KB)
0
200
400
600
800
1000
1200
0
50
100
150
200
Time (sec)

- **Results**


Sequence Number (KB)
0
200
400
600
800
1000
1200
0
50
100
150
200
Time (sec)

- **Takeaways**
  - This congestion avoidance technique is said to save the Internet from collapsing in the late 1980s and early 1990s
  - Be conservative when start: slow-start and additive increase
  - Be aggressive when back-off: exponential

- Exponential back-off is proven to be quite successful
- Any potential for performance improvement?

- Other congestion control algorithm

Technique	Target	Feature
TCP Tahoe	General	Slow start, congestion window, fast retransmit
TCP Reno	General	Fast recovery
TCP Hybla	High-latency connection	Analytical evaluation of the congestion window dynamics, remove RTT
TCP BIC	Linux 2.6.8-18	Tries to find the maximum window size to keep for a long period of time, by using a binary search
TCP CUBIC	Linux 2.6.19	The window is a cubic function of time since the last congestion event, independent of RTT
Compound TCP	Microsoft	Use estimate of queuing delay to measure congestion
Fast TCP	High latency	Improved speed of convergence and stability
Data center TCP	General	Allows end-to-end notification of network congestion without dropping packets

Source: wikipedia.org

How to break TCP

- TCP congestion control requires that everybody follow the rules
- Unfortunately, never assume people are self-disciplined!

How to break TCP

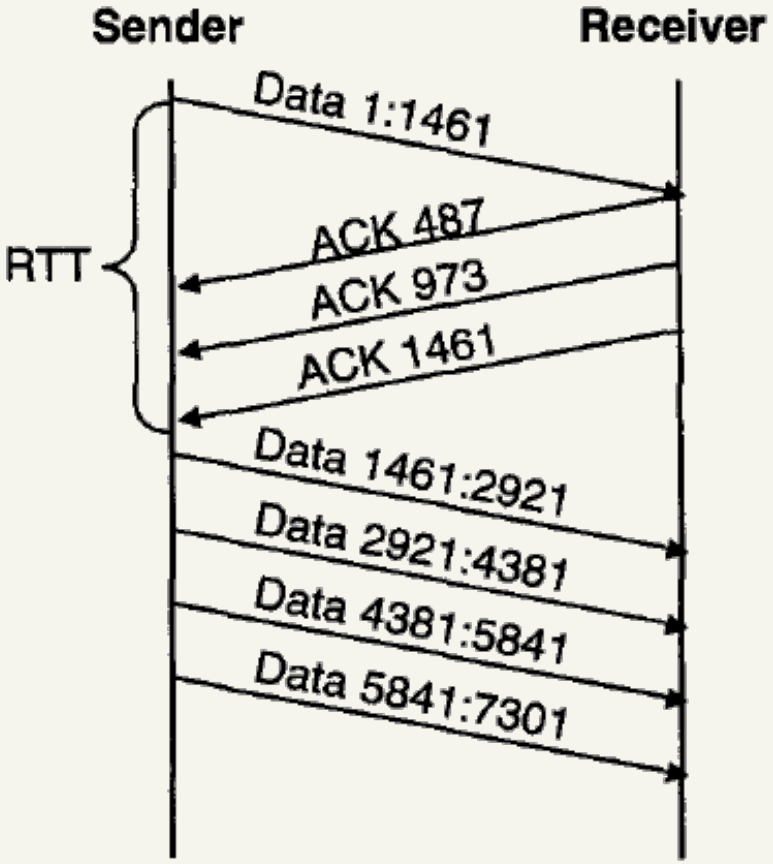
- TCP Congestion Control with a Misbehaving Receiver
- Stefan Savage: professor @ UCSD
- Neal Cardwell: Google research
- David Wetherall: professor @ UW-Seattle
- Tom Anderson: co-founder of myspace.com

How to break TCP

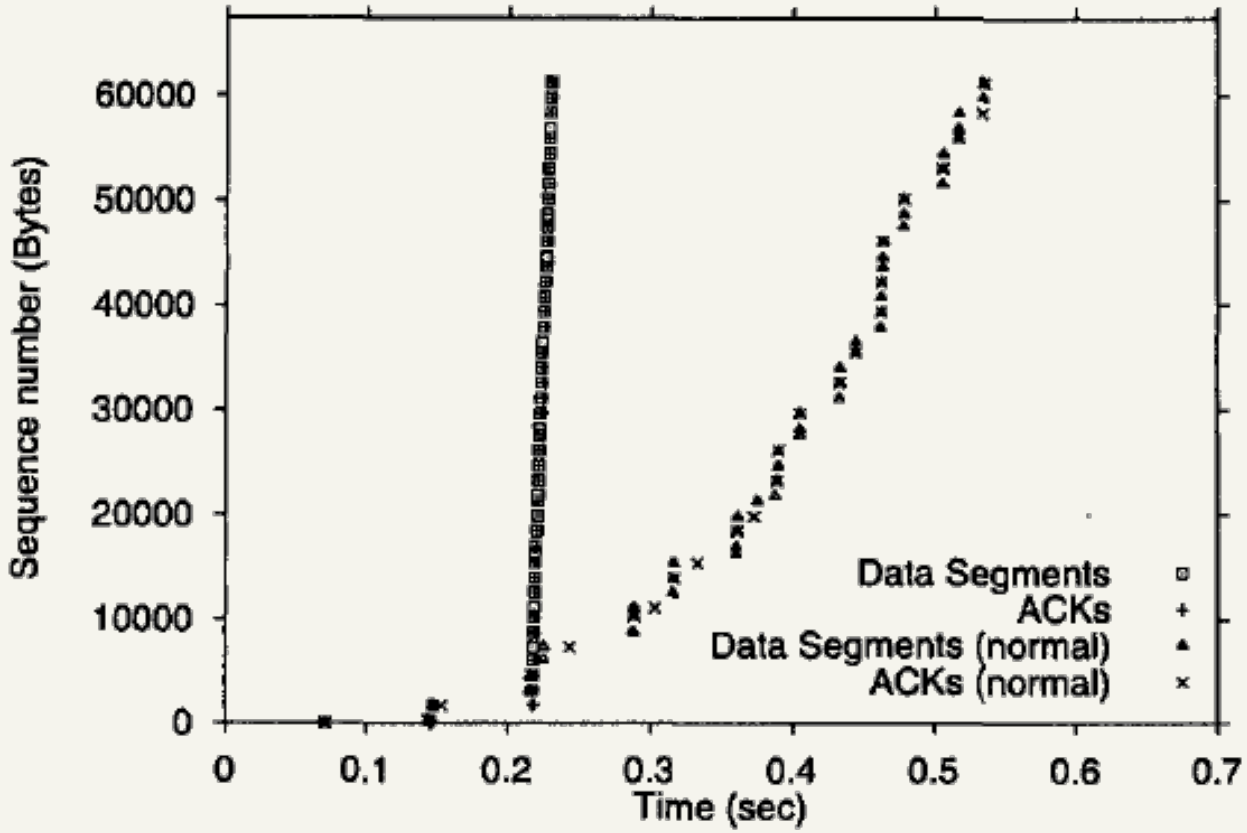
- **RFC tries to maximize performance**
  - E.g. fast retransmit and fast recovery
- **RFC expects people to follow the rules**
- **People are selfish: they have the motivation to manipulate the rules to gain benefit!**
- **As a receiver, you can steal the bandwidth by:**
  - ACK division
  - DupACK spoofing
  - Optimistic ACKing

- **ACK division**
  - TCP Specification:
    - » During slow start, TCP increments *cwnd* by at most SMSS bytes for each ACK received that acknowledges new data
  - Attack:
    - » On receiving a data segment, divide into M pieces and ACK each one.
    - » *cwnd* of the sender will increase by M * SMSS instead of SMSS

- ACK division



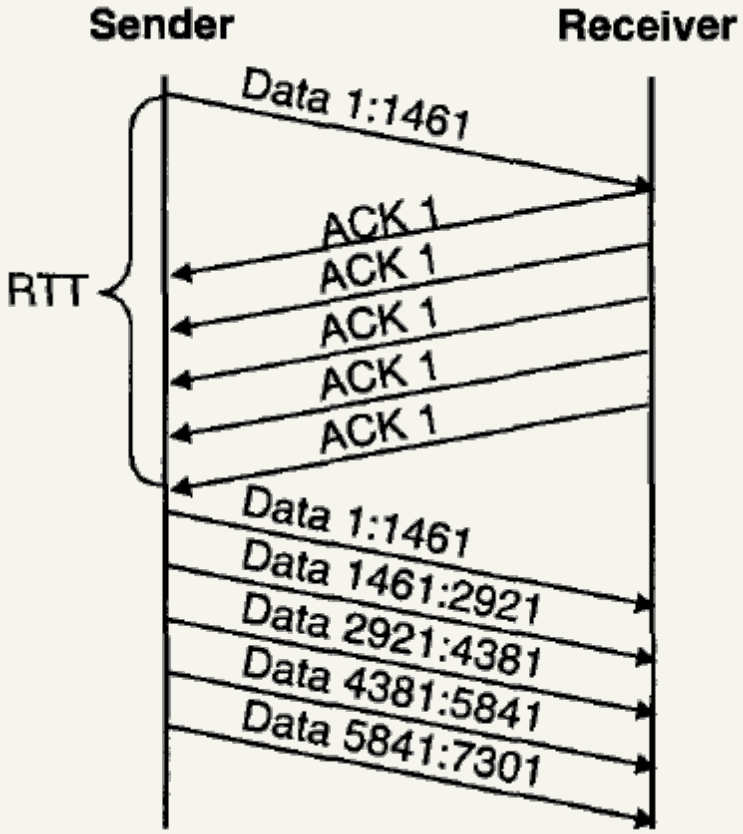
- ACK division



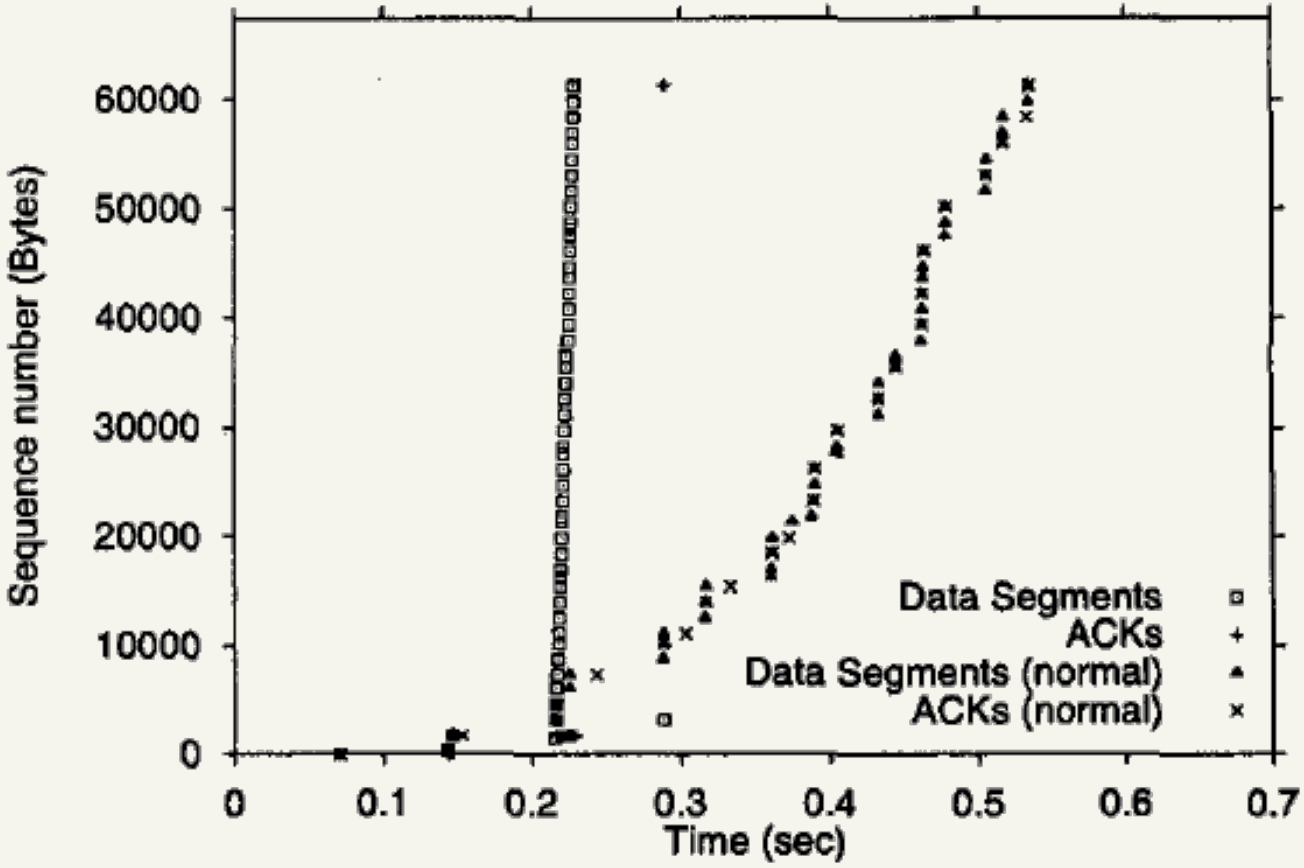
- **ACK division: solution**
  - Problem caused by ambiguity in ACK interpretation
    - » Congestion control operates at byte granularity
    - » Or, operate at segment granularity always --- from Linux 2.2

- **DupACK spoofing**
  - TCP fast recovery: on receiving a duplicate ACK
    - » Set *cwnd* = *ssthresh* + 3*SMSS
    - » Increment *cwnd* by SMSS for each additional duplicate
  - Attack:
    - » Upon receiving a data segment, the receiver sends a long stream of acknowledgments for the last sequence number received

- DupACK spoofing



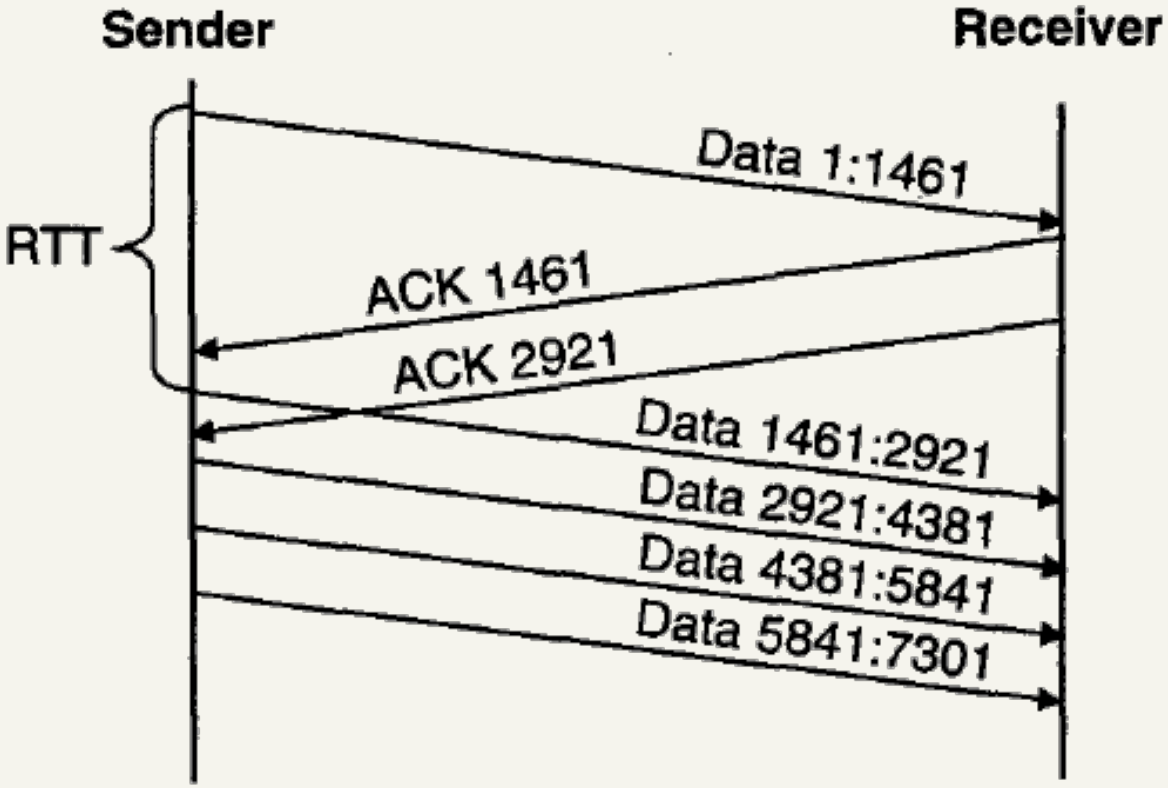
- DupACK spoofing



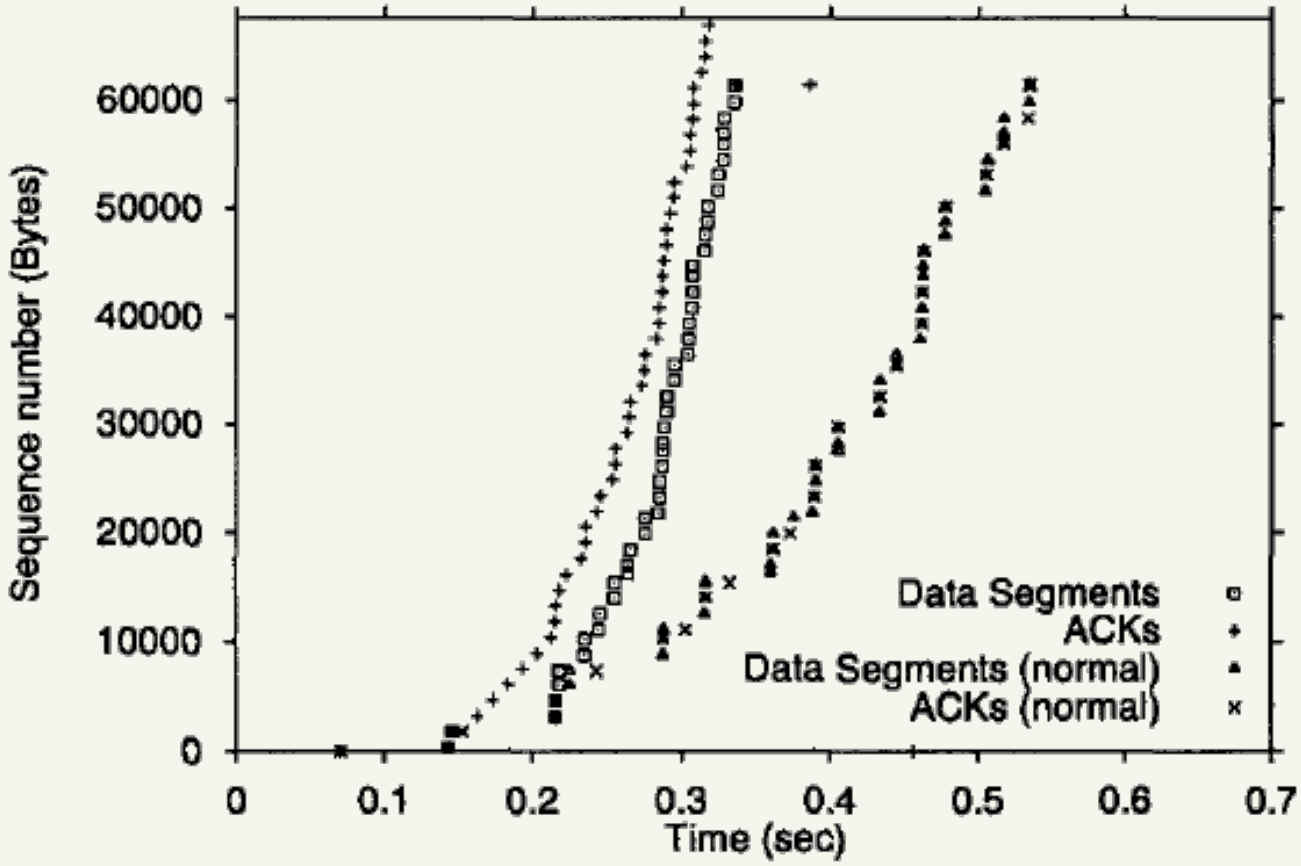
- **DupACK spoofing: solution**
  - Get rid of fast recovery! Or:
  - Need to identify the data segment that lead to the duplicate ACK
  - Only way to do this is to add new TCP fields: nonce and nonce reply
  - The nonce field in a data packet is a random number
  - The nonce reply field is supposed to send back the same number

- **Optimistic ACKing**
  - TCP specification: the speed of *cwnd* increase is proportional to the response time of ack
  - Attack:
    - » Upon receiving a data packet, send a stream of ACKs in anticipation of future data packets

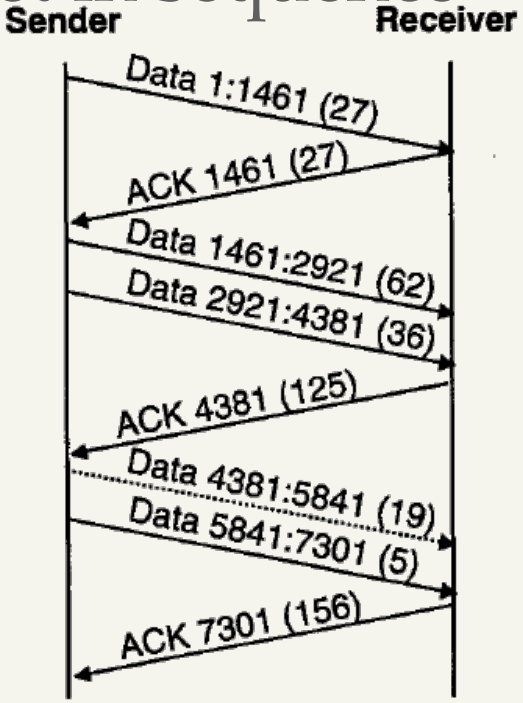
- Optimistic ACKing



- Optimistic ACKing



- **Optimistic ACKing: solution**
  - ACKs do not contain proof about which data segment they represent
  - Adding a cumulative nonce for packet in sequence
  - Actually in use in today's TCP
    - » RFC3540



How to break TCP

- **Takeaways:**
  - TCP is built under cooperative assumption
  - Problem can be fixed once the attack is identified
  - Always some loopholes in RFC?
  - Design principle needs a change?

TAKEAWAY FOR TCP DESIGN

- **Maintain equilibrium is the top priority for congestion control**
  - Be conservative when start: slow-start and additive increase
  - Be aggressive when back-off: exponential
- **Cannot assume everyone follows the rule**
  - Assumptions should be clearly stated
  - Unstated assumptions are easy to cause loopholes

That's all, Folks !

Thank you!