Networking (from an OS perspective)

David Goff

Congestion and TCP

- Even between trustworthy individuals, TCP can collapse when under severe congestion
  - 3-hop link between LBL and UC Berkley
  - October 1986, throughput dropped from 32Kbps to 40bps
  - Thousandfold drop in bandwidth
- Malicious receivers can take advantage of oversights in TCP to increase their own throughput at the expense of others

Congestion Avoidance and Control

- Van Jacobson
  - Cisco → Parc

Source: http://en.wikipedia.org/wiki/Van_Jacobson

Conservation of Packets Principle

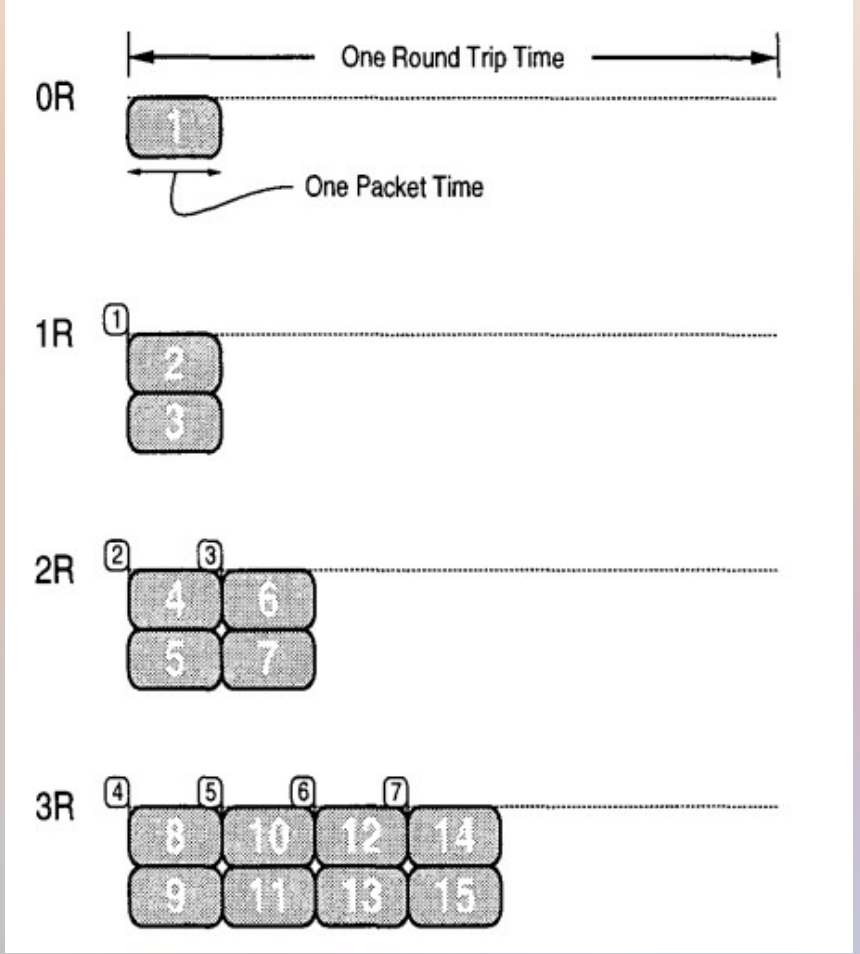
- Packets reach an equilibrium
- Once reached, packets enter and leave the network at the same rate
- So, how do we uphold this principle?
  - Well, what conditions might cause it to be violated?
  - And, how do we prevent these conditions?

Failure Cases for Packet Conservation

- Case I: The connection does not get to equilibrium
- Case II: Sender adds packets faster than they are removed
- Case III: Resource limits within the network path prevent equilibrium from being reached

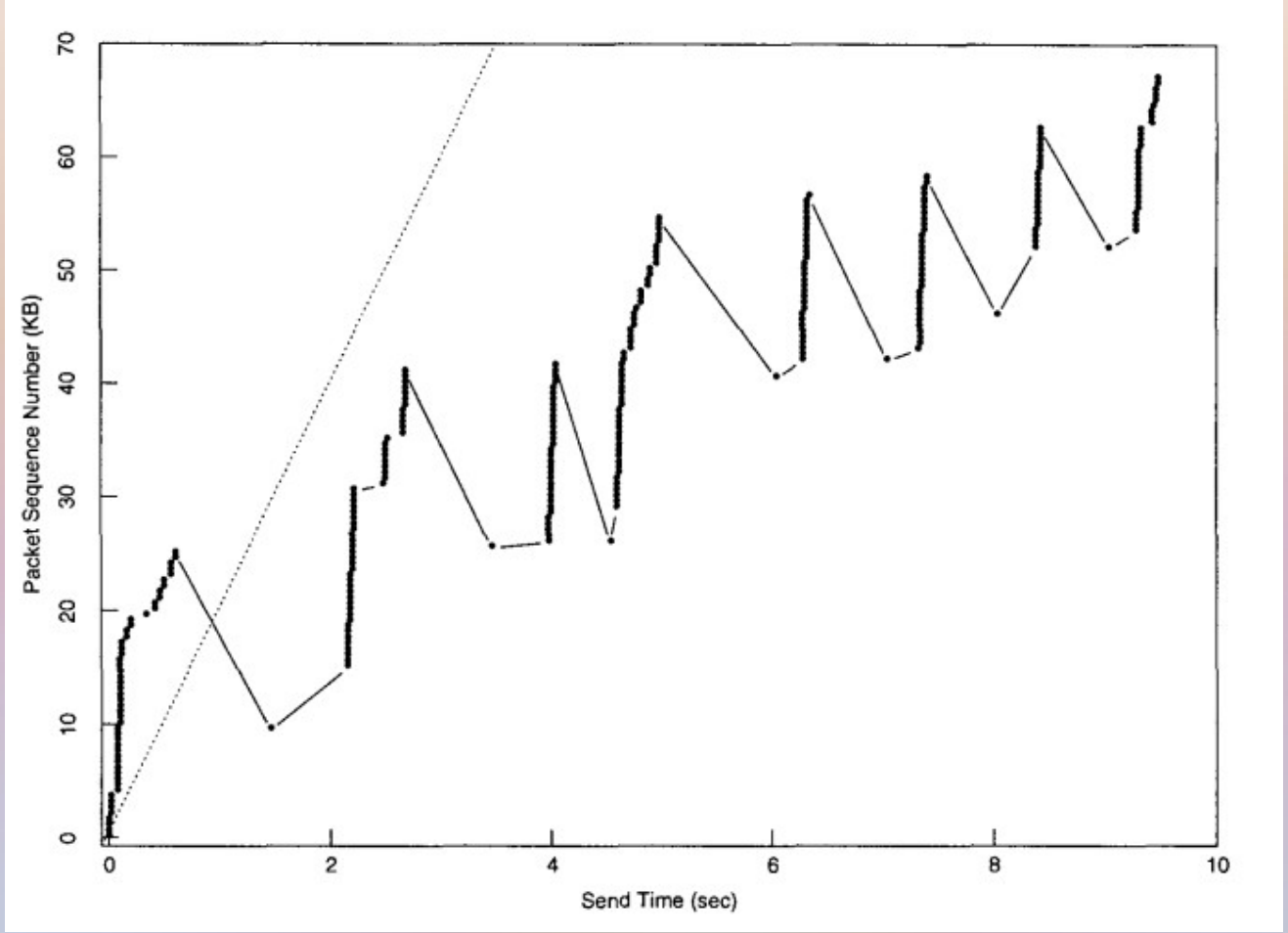
Case I - Getting to Equilibrium

- Slow Start
- Keep a congestion window, *cwnd*
- Start with *cwnd* = 1
- Increase *cwnd* by 1 for each ACK
- Not actually that slow



Congestion Avoidance and Control, Van Jacobson. Appears in Proceedings of ACM SIGCOMM, Vo1ume 18, Number 4, (August 1988).

Startup behavior without Slow Start

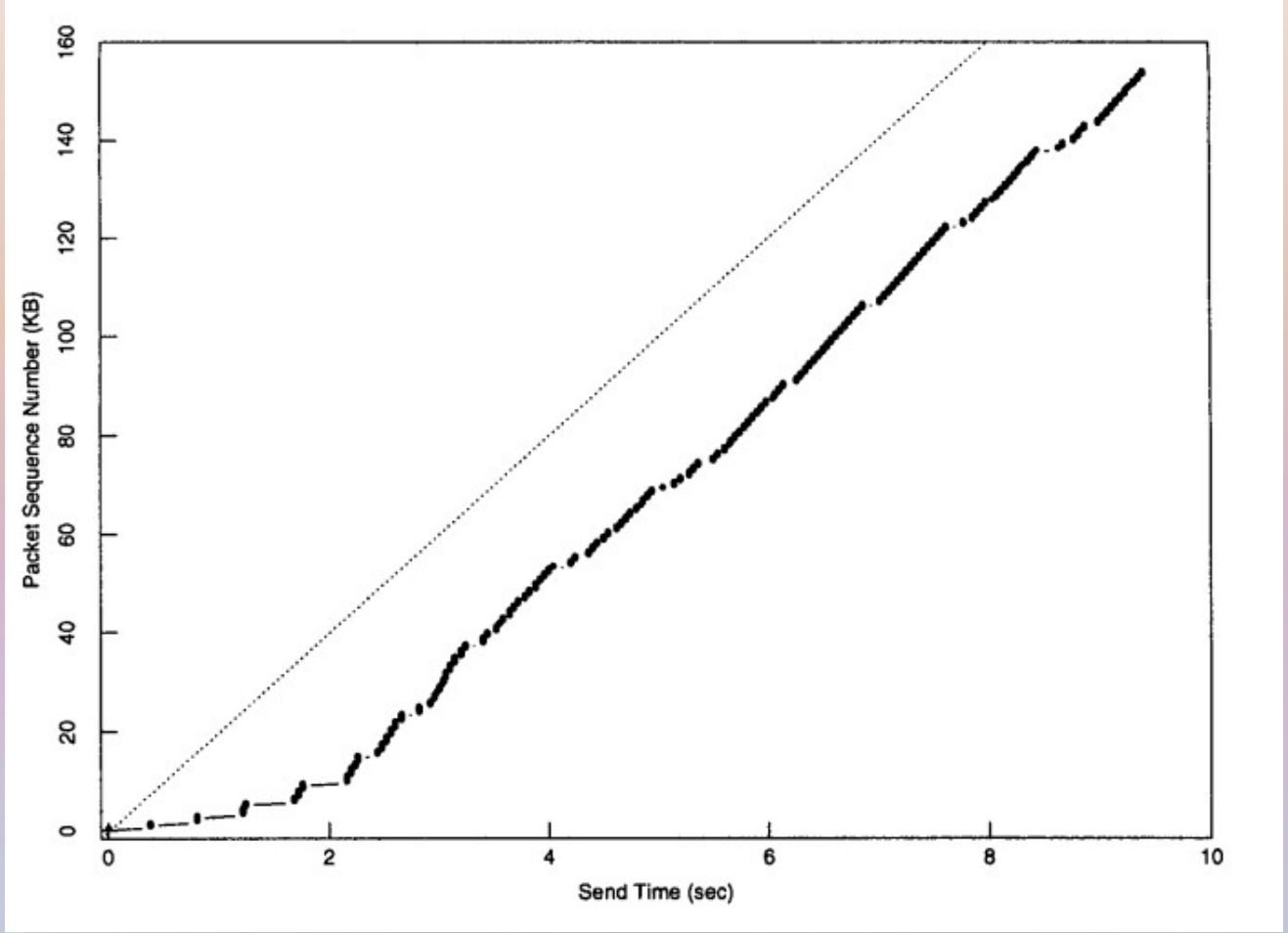


Congestion Avoidance and Control, Van Jacobson. Appears in Proceedings of ACM SIGCOMM, Vo1ume 18, Number 4, (August 1988).

Points of Interest About Previous Graph

- Each dot is a 512 byte data packet
- 20Kbps network
- Almost all packets must be retransmitted
- Data from 54 to 58 Kb is transmitted 5 times
- Only 35% of actual bandwidth is used

Startup Behavior with Slow Start



Congestion Avoidance and Control, Van Jacobson. Appears in Proceedings of ACM SIGCOMM, Vo1ume 18, Number 4, (August 1988).

Points of Interest about Previous Graph

- Same parameters as previous experiment
- Notice how the window size doubles in each burst
- No bandwidth wasted on retransmits
- Over time, effective bandwidth approaches ideal bandwidth

Case II – Round Trip Time

- Need to know RTT to know when to add next packet
- Importance of estimating σ_R
- Old method
  - $R = \alpha R + (1 - \alpha)M$
  - $rto = \beta R$
  - $\beta = 2$
  - Only works for loads of 30%

Case III - Congestion Avoidance

- Sender needs a way to detect network congestion
- Sender must reduce load when congestion is detected
- Packet loss is almost always due to congestion

Response to Congestion

- L_i = load during time interval i
- N = load at some sampling time
- $L_i = N + \gamma L_{i-1}$

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

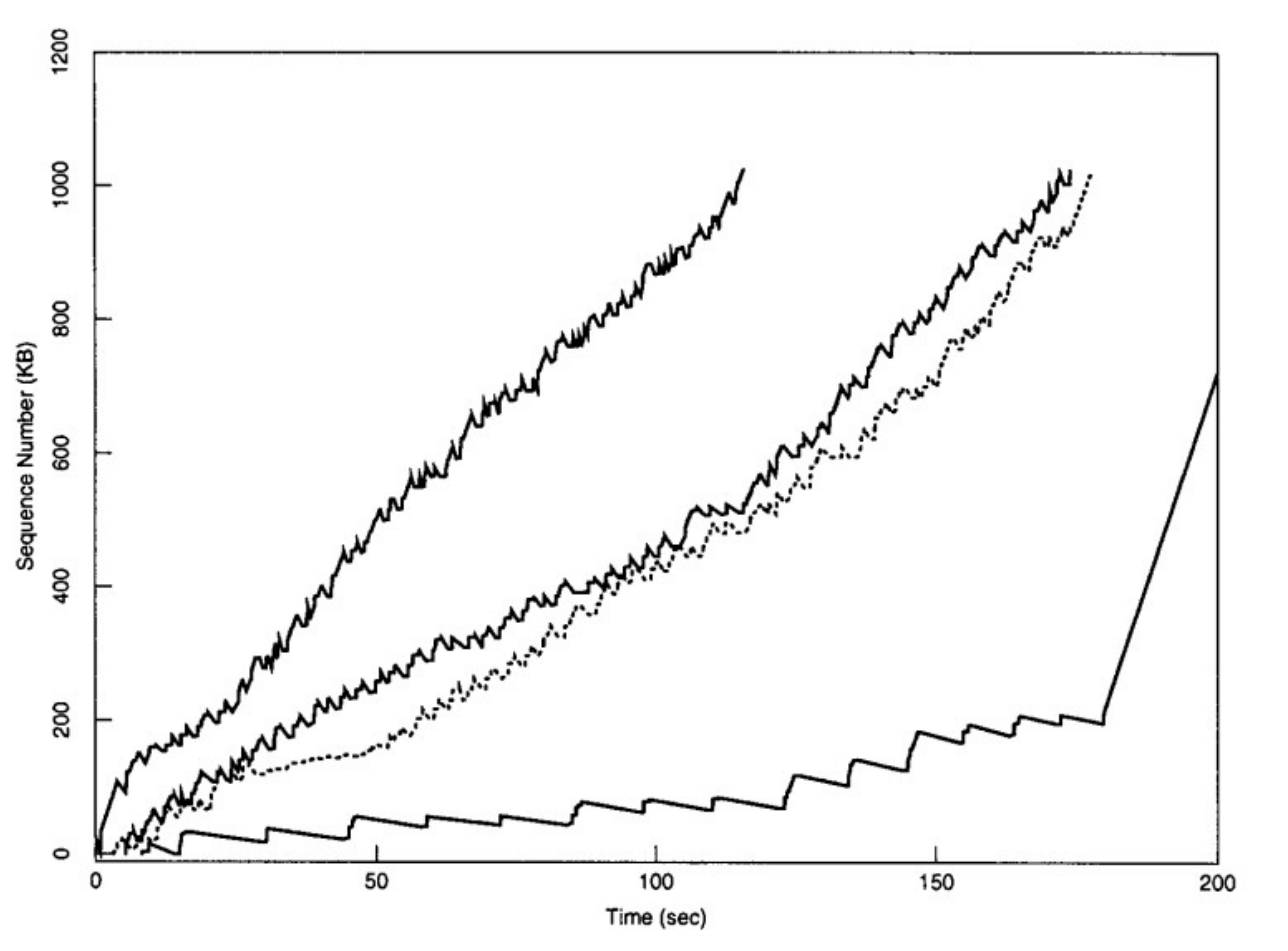
What about On No Congestion?

- Bad solution: $W_i = bW_{i-1}$ $(1 < b < 1/d)$
- Overestimating is much worse than underestimating
- Need small increases in W_i

Final Congestion Control Algorithm

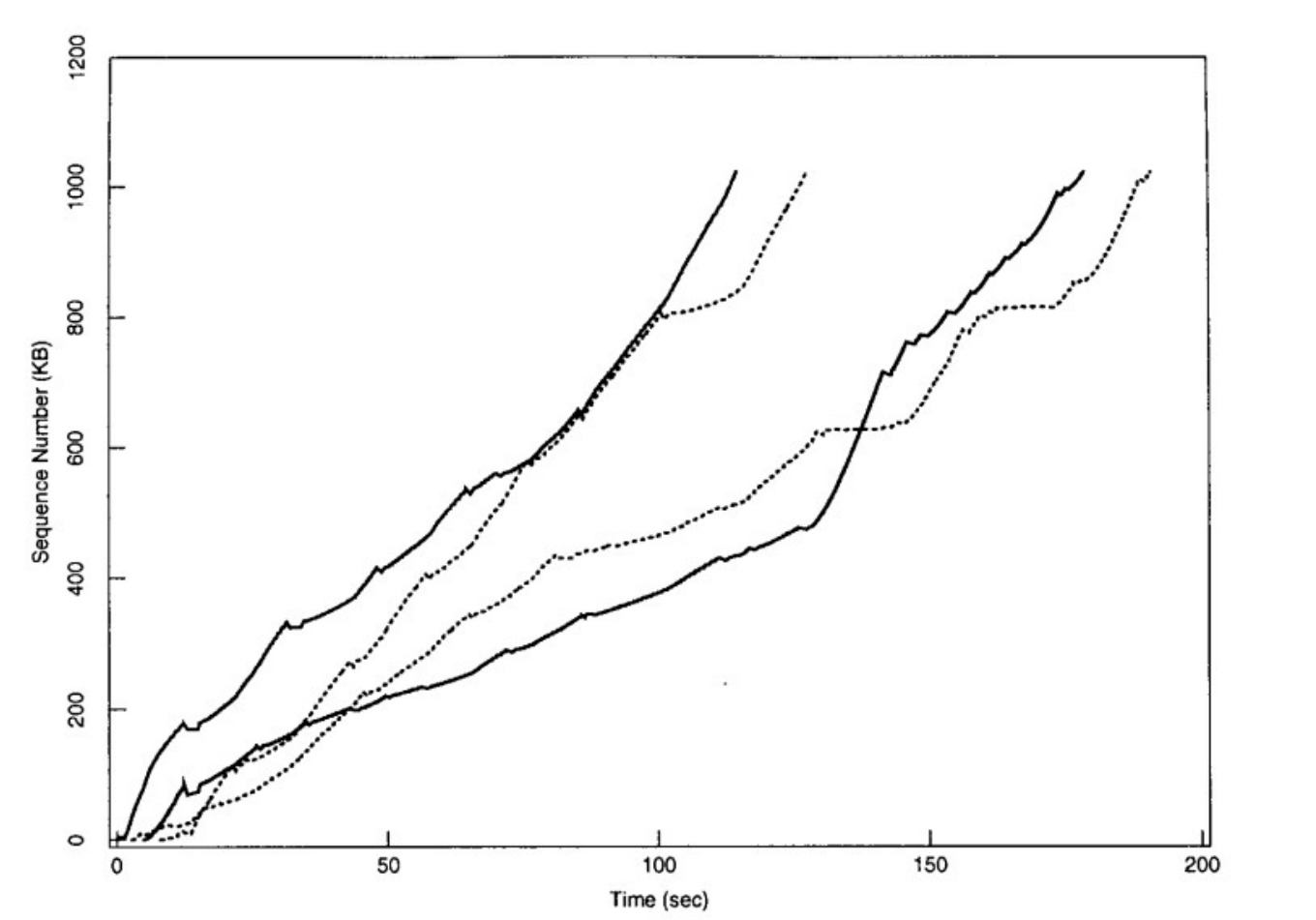
- On timeout, cut *cwnd* in half (multiplicative decrease)
- On each ACK received for new data, increase *cwnd* by 1/*cwnd* (additive increase)
- On sending, send the minimum of *cwnd* and the recipient's advertised *cwnd*

4 TCP Instances, No Congestion Avoidance



Congestion Avoidance and Control, Van Jacobson. Appears in Proceedings of ACM SIGCOMM, Vo1ume 18, Number 4, (August 1988).

4 TCP instances, Congestion Avoidance



Congestion Avoidance and Control, Van Jacobson. Appears in Proceedings of ACM SIGCOMM, Vo1ume 18, Number 4, (August 1988).

Next Steps: Congestion Control at Gateways

- Algorithms at the endpoints of communication cannot ensure fairness
- Only gateways have enough information
- No need to modify endpoint code
- Early detection is important

Changes to TCP

- round-trip-time variance estimation
- exponential retransmit timer backoff
- slow-start
- more aggressive receiver ack policy
- dynamic window sizing on congestion
- Karn's clamped retransmit backoff (not described)
- fast retransmit (not described)

Takeaways

- TCP can scale
- Needs
  - Slow start
  - Exponential backoff
  - Slow probe for additional bandwidth

So...

- Now that TCP is congestion avoidant..
- ...Let's try to break it...

TCP Congestion Control with a Misbehaving Receiver

- Stefan Savage
  - University of Washington → University of California
- Neal Cardwell
  - University of California → University of Washington
- David Wetherall
  - University of Washington
- Tom Anderson
  - University of Washington → MySpace

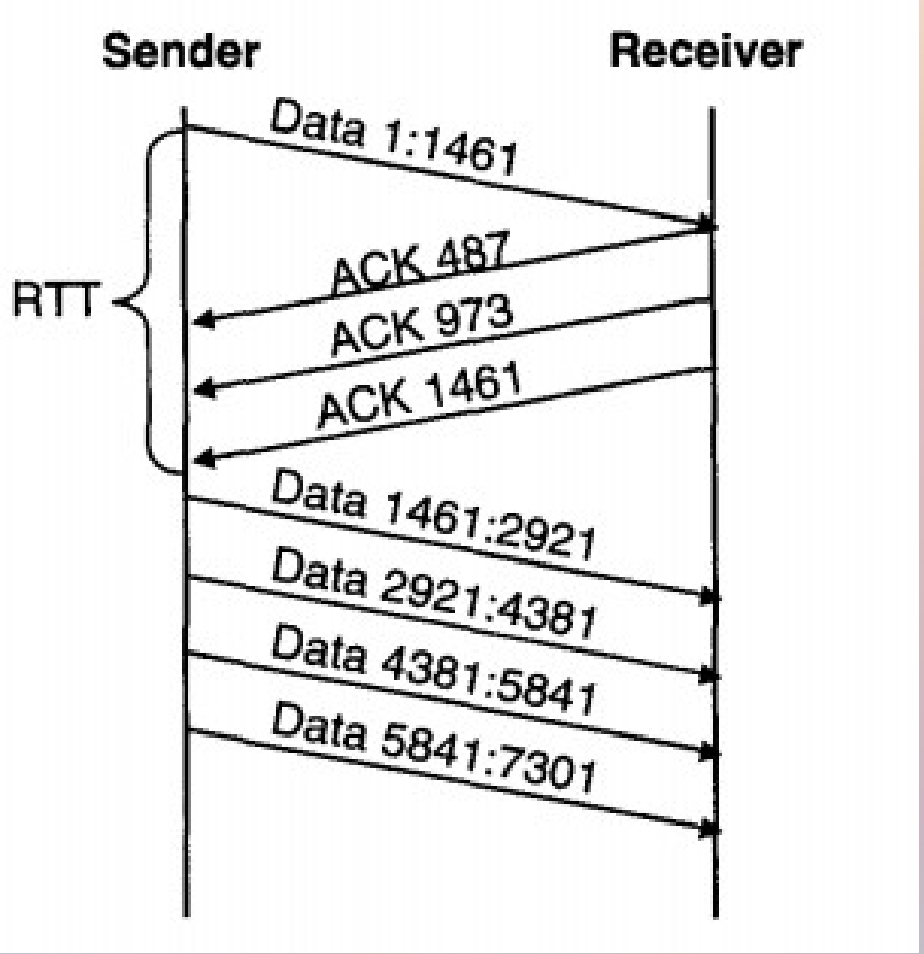
Problem: TCP Receiver is Trusted

- Receivers outnumber Senders
- *Receivers* have the motive and the means to break *Sender* congestion control
- This allows receivers to steal bandwidth from each other
- Tested 3 Attacks
  - ACK Division
  - DupACK Spoof
  - Optimistic ACKing

Attack 1: ACK Division

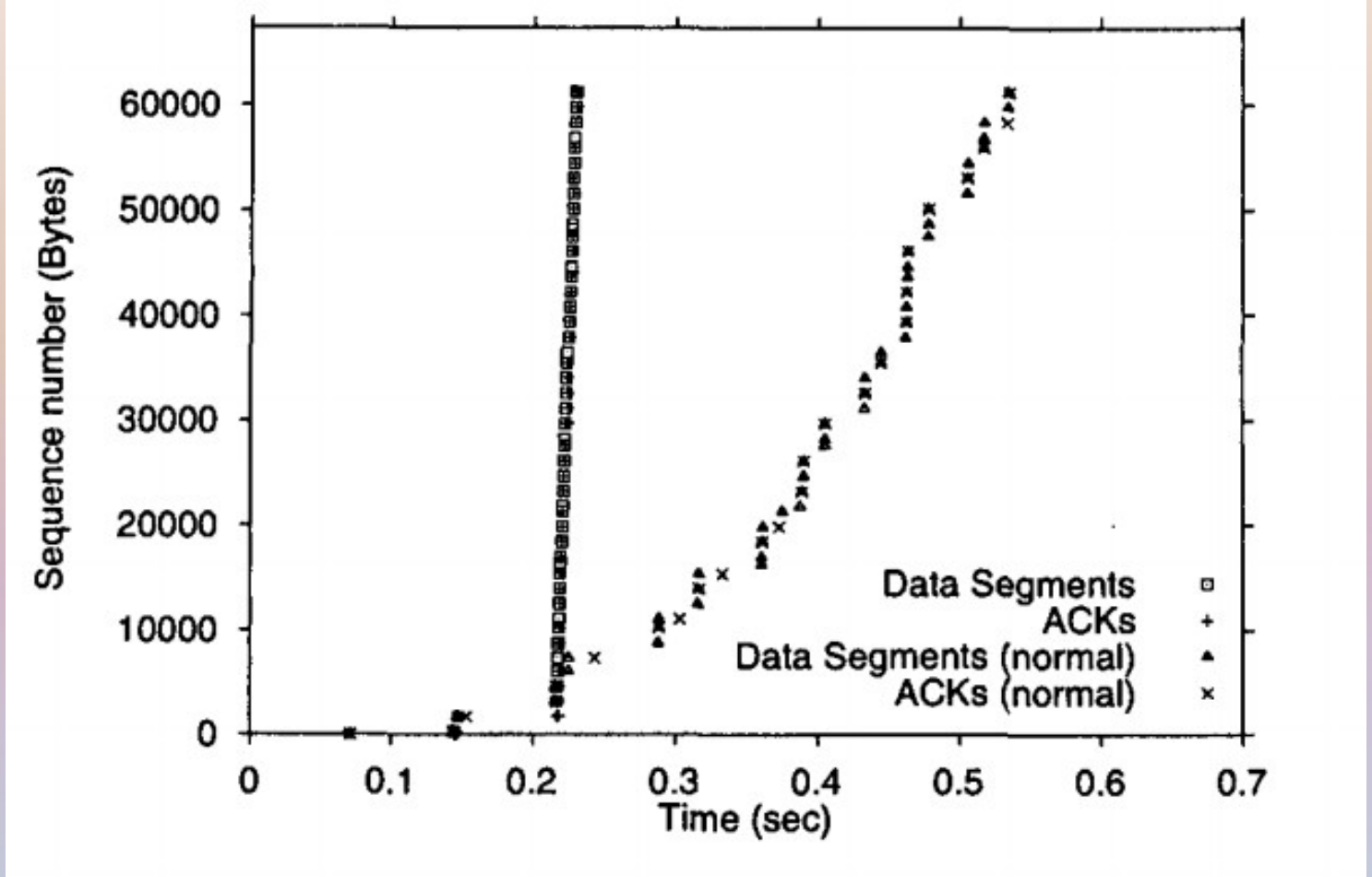
- TCP Specification:
  - During slow start, TCP increments cwin by at most SMSS bytes for each ACK received that acknowledges new data
- Attack:
  - On receiving a data segment, divide into M pieces and ACK each one.
  - cwin of the sender will increase by M * SMSS instead of SMSS

Attack 1: ACK Division



TCP congestion control with a misbehaving receiver

Attack 1: ACK Division



TCP congestion control with a misbehaving receiver

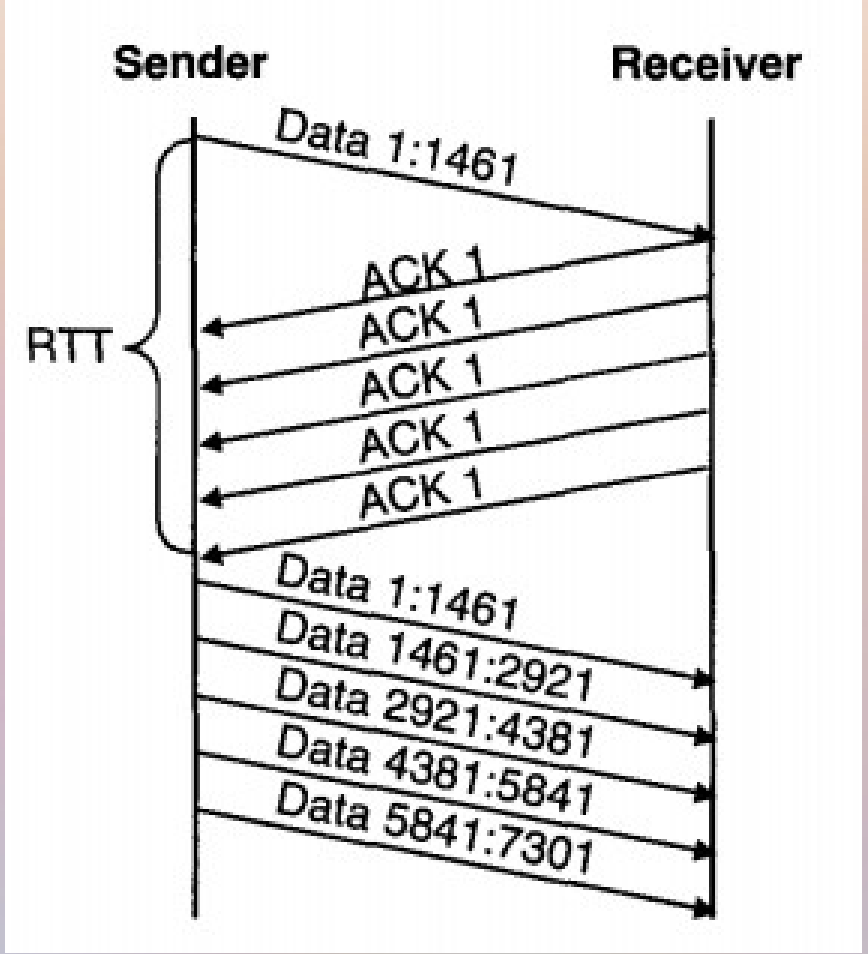
Solution – ACK division

- Problem caused by ambiguity in ACK interpretation
  - TCP error-control allows specific byte offsets to be ACKed
  - Congestion control expects ACKs to cover entire segments
- Solution: pick one of these and stick to it
- Backwards compatible: only sender protocol needs modification

Attack 2: DupACK Spoof

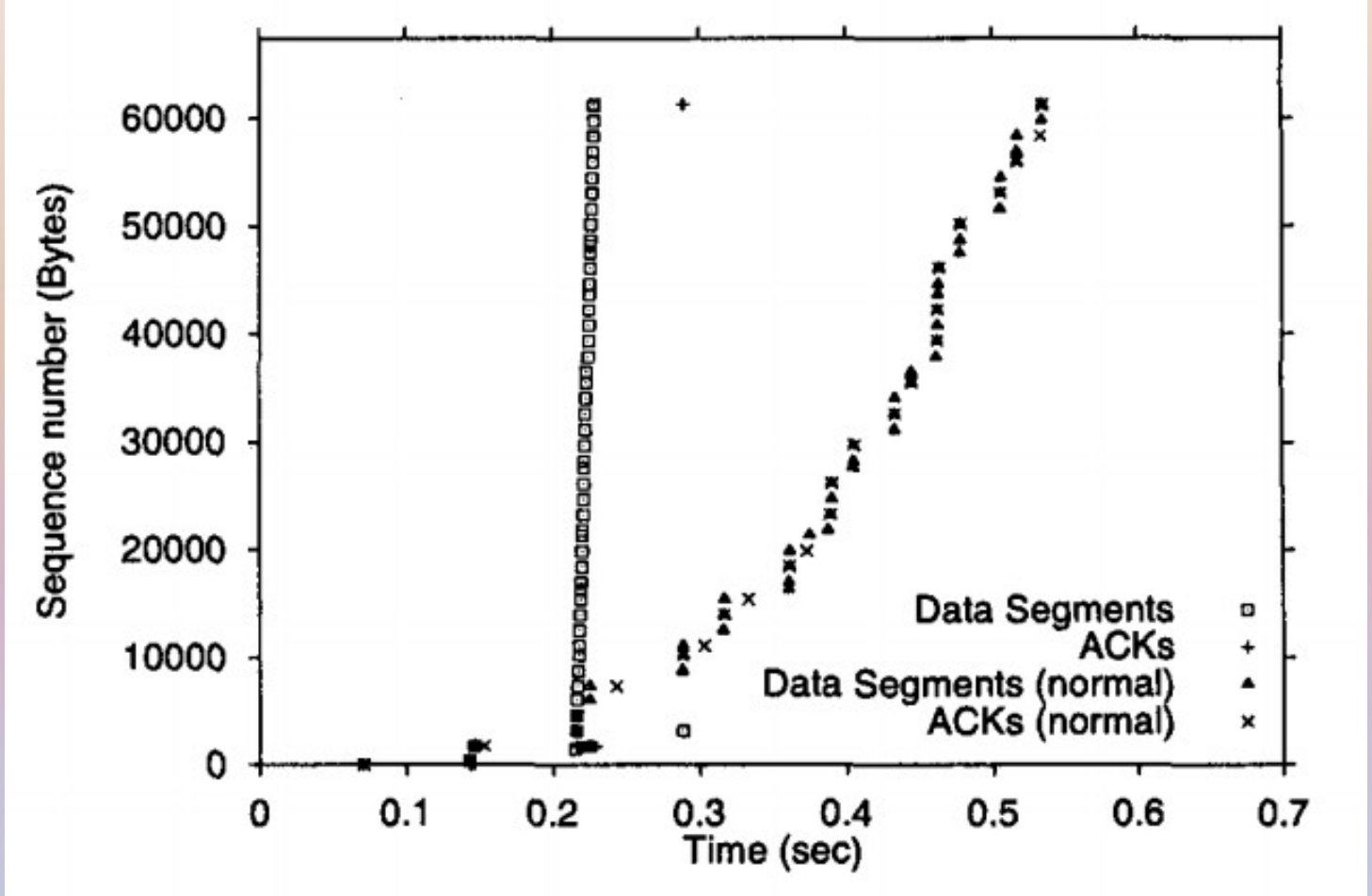
- TCP specification, on receiving a duplicate ACK
  - Set cwnd to ssthresh plus 3*SMSS
  - Increment cwnd by SMSS for each additional duplicate
- Justification: supposed to mean that packets left the network
- Attack:
  - Upon receiving a data segment, the receiver sends a long stream of acknowledgments for the last sequence number received

Attack 2: DupACK Spoof



TCP congestion control with a misbehaving receiver

Attack 2: DupACK Spoof



TCP congestion control with a misbehaving receiver

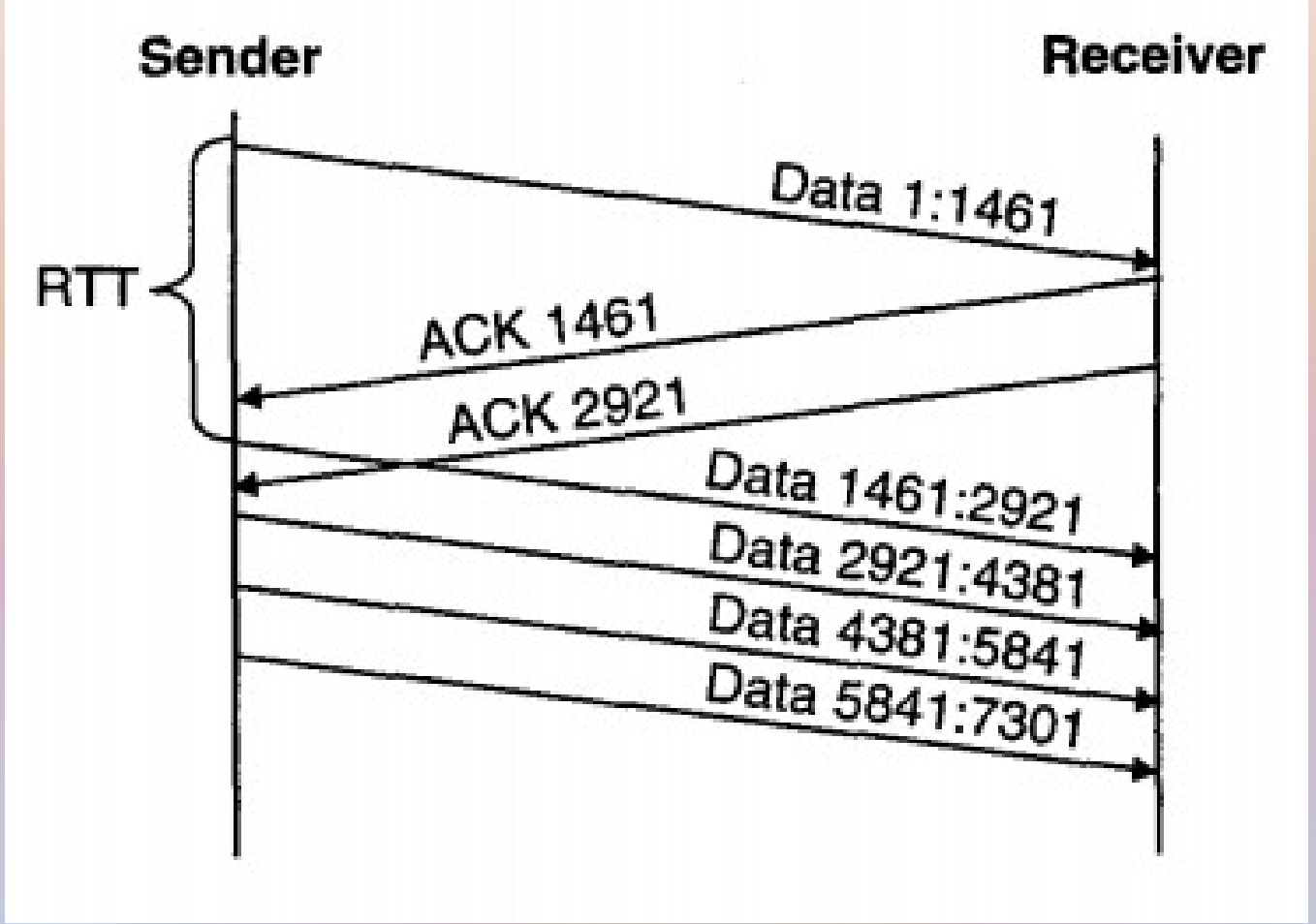
Solution – DupACK Spoof

- 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
- Unfortunately, not backwards compatible

Attack 3: Optimistic ACKing

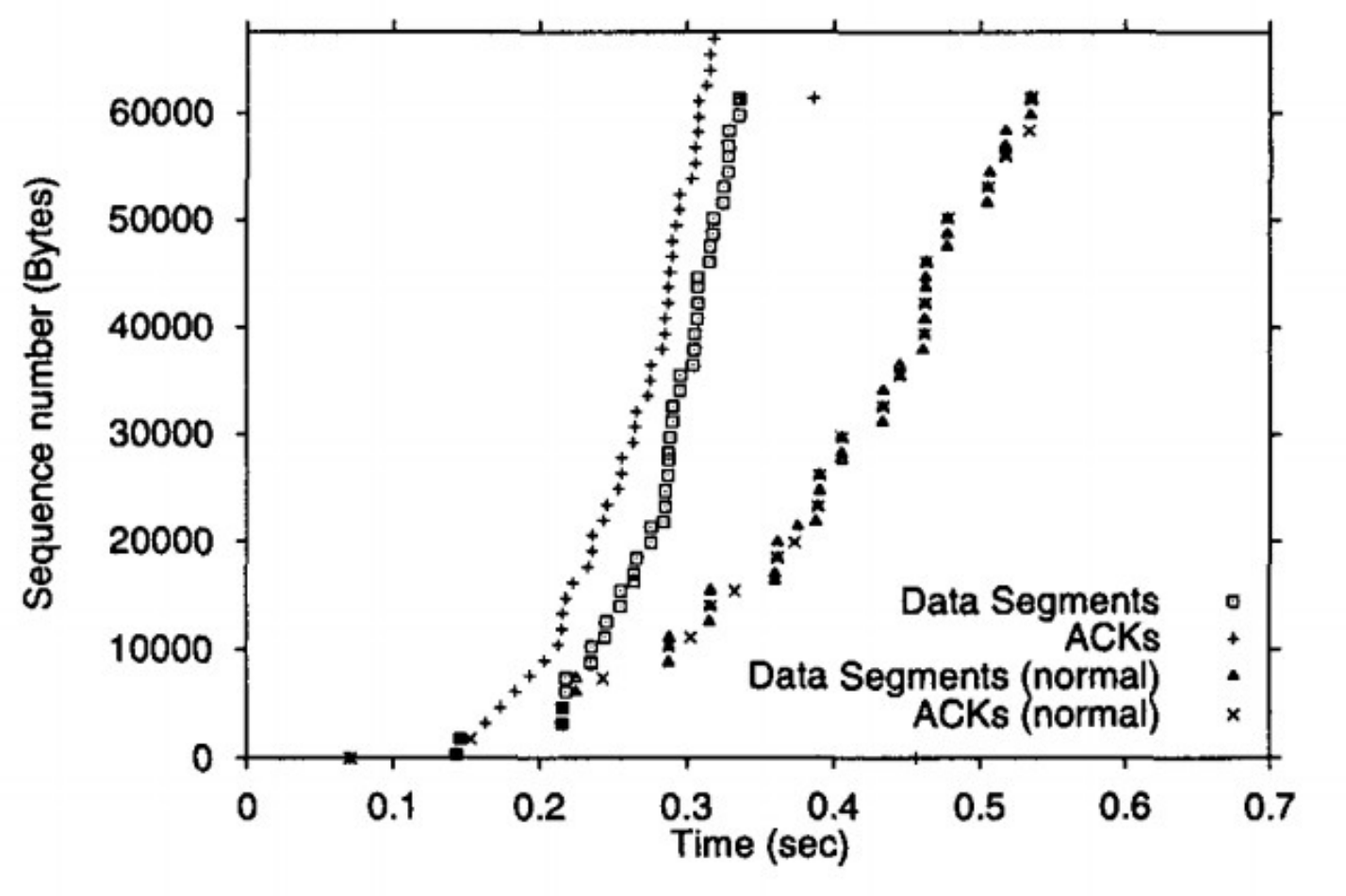
- TCP assumes that the time between message and its ACK is at least the round trip time of the network
- Attack:
  - Upon receiving a data packet, send a stream of ACKs in anticipation of future data packets
- Breaks end-to-end semantics

Attack 3: Optimistic ACKing



TCP congestion control with a misbehaving receiver

Attack 3: Optimistic ACKing

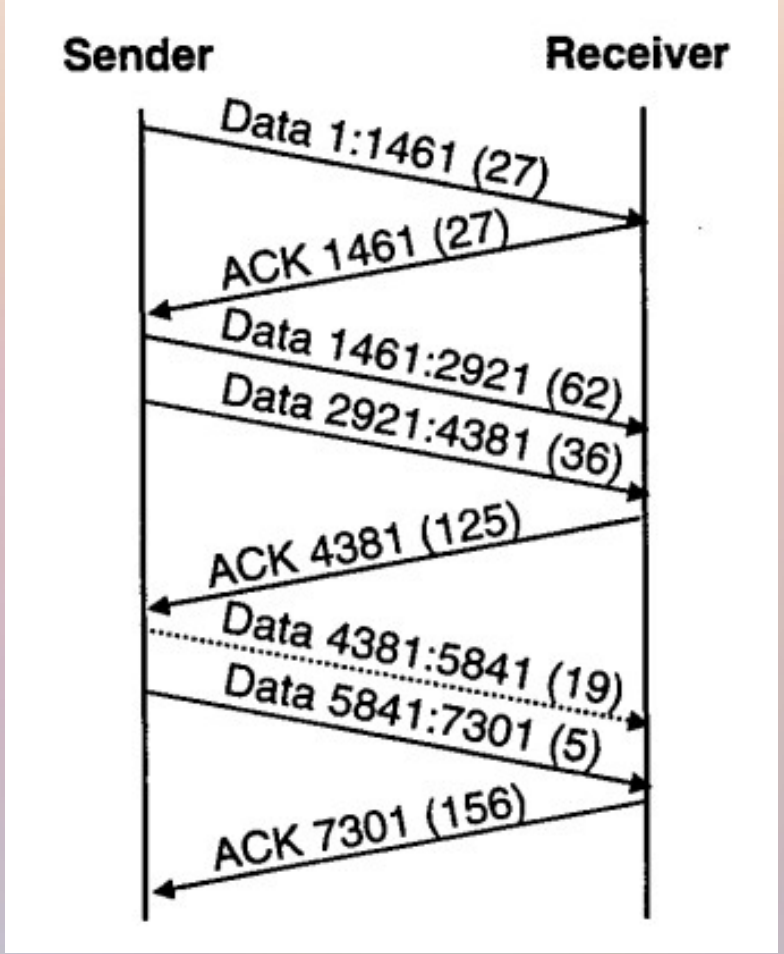


TCP congestion control with a misbehaving receiver

Solution – Optimistic ACKing

- Only possible because ACKs do not contain proof about which data segment they represent
- Also solvable by adding a nonce
- TCP is cumulative, so a cumulative nonce should be used

Cumulative Nonce



TCP congestion control with a misbehaving receiver

Takeaways

- TCP should not require that both parties be trusted
- Most systems vulnerable to attacks
- Simple fixes can prevent many kinds of attacks

Questions?