TCP in Painful Detail

Michael Welzl http://www.welzl.at

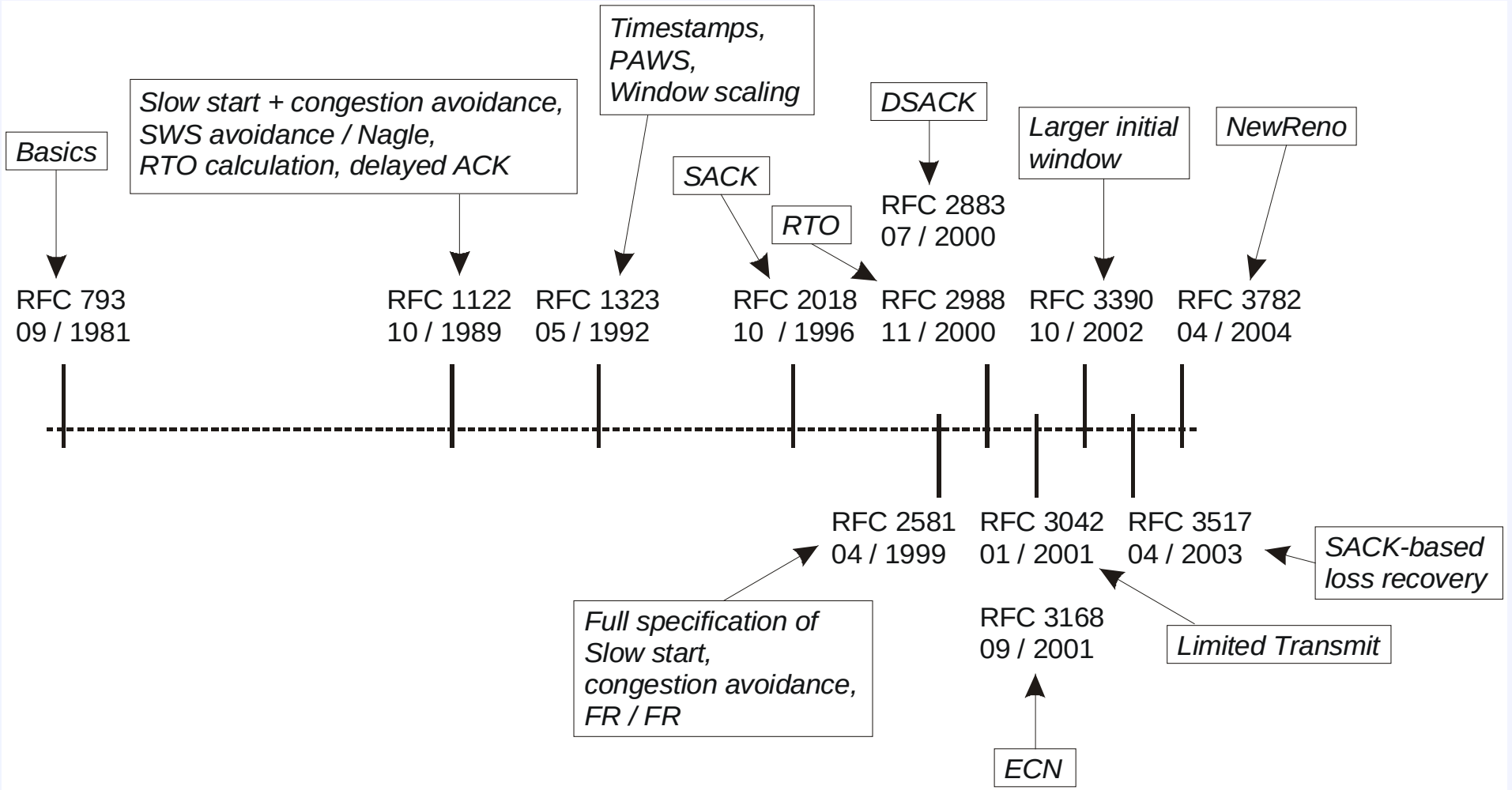
DPS NSG Team http://dps.uibk.ac.at/nsg
Institute of Computer Science
University of Innsbruck, Austria

What TCP does for you (roughly)

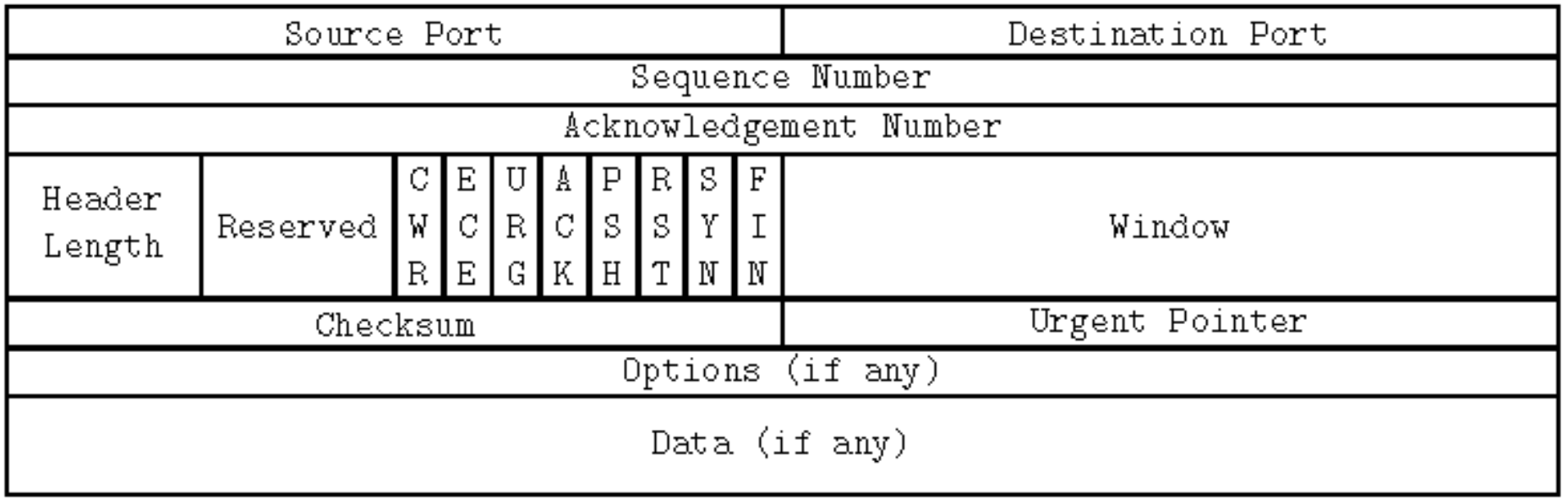
- UDP features: multiplexing + protection against corruption
  - ports, checksum
- stream-based in-order delivery
  - segments are ordered according to sequence numbers
  - only consecutive bytes are delivered
- reliability
  - missing segments are detected (ACK is missing) and retransmitted
- flow control
  - receiver is protected against overload (window based)
- congestion control
  - network is protected against overload (window based)
  - protocol tries to fill available capacity
- connection handling
  - explicit establishment + teardown
- full-duplex communication
  - e.g., an ACK can be a data segment at the same time (piggybacking)

TCP History

Standards track TCP RFCs which influence when a packet is sent (status: early 2005)

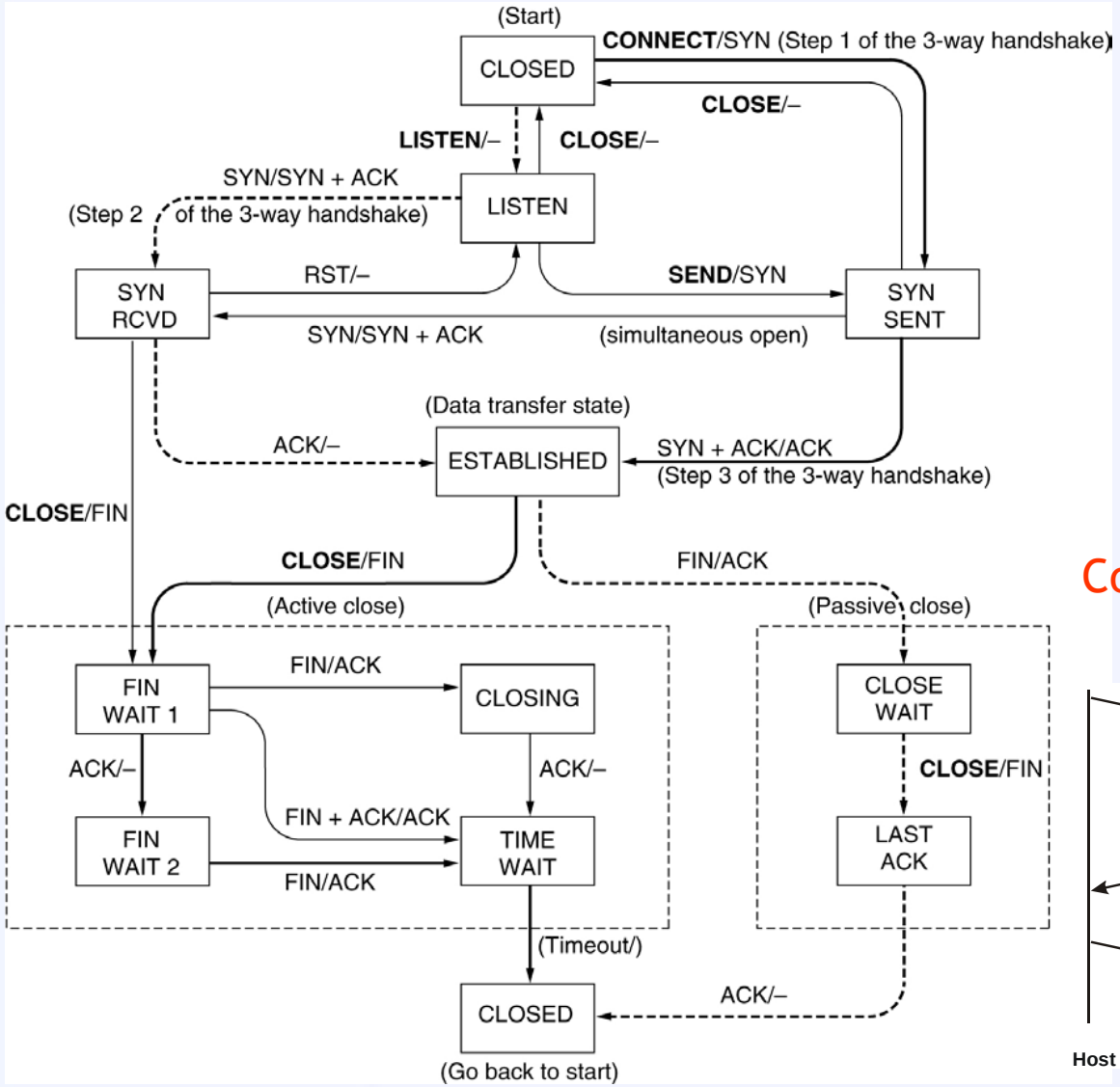


TCP Header



- Flags indicate connection setup/teardown, ACK, ..
- If no data: packet is just an ACK
- Window = advertised window from receiver (flow control)

TCP Connection Management

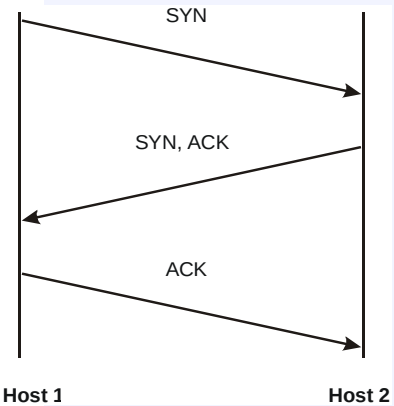


heavy solid line:
normal path for a client

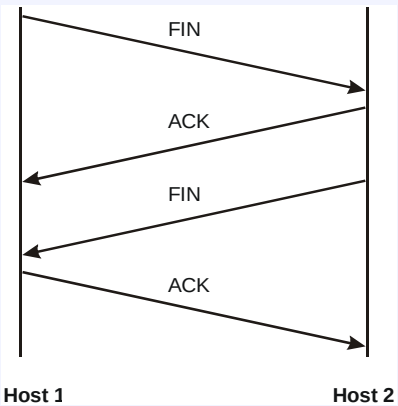
heavy dashed line:
normal path for a server

Light lines:
unusual events

Connection setup

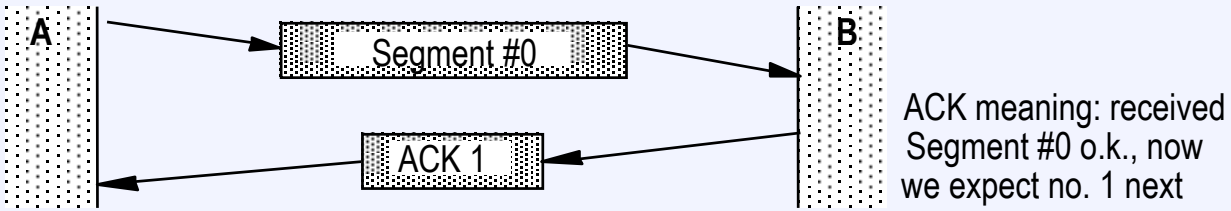


teardown



Error Control: Acknowledgement

ACK ("positive" Acknowledgement)



Purposes:

- sender: throw away copy of segment held for retransmit,
- time-out cancelled
- msg-number can be re-used

TCP counts bytes, not segments; ACK carries "next expected byte" (#+1)

ACKs are cumulative

- ACK *n* acknowledges all bytes "*last one ACKed*" thru *n-1*

ACKs should be delayed

- TCP ACKs are unreliable: dropping one does not cause much harm
- Enough to send only 1 ACK every 2 segments, or at least 1 ACK every 500 ms (often set to 200 ms)

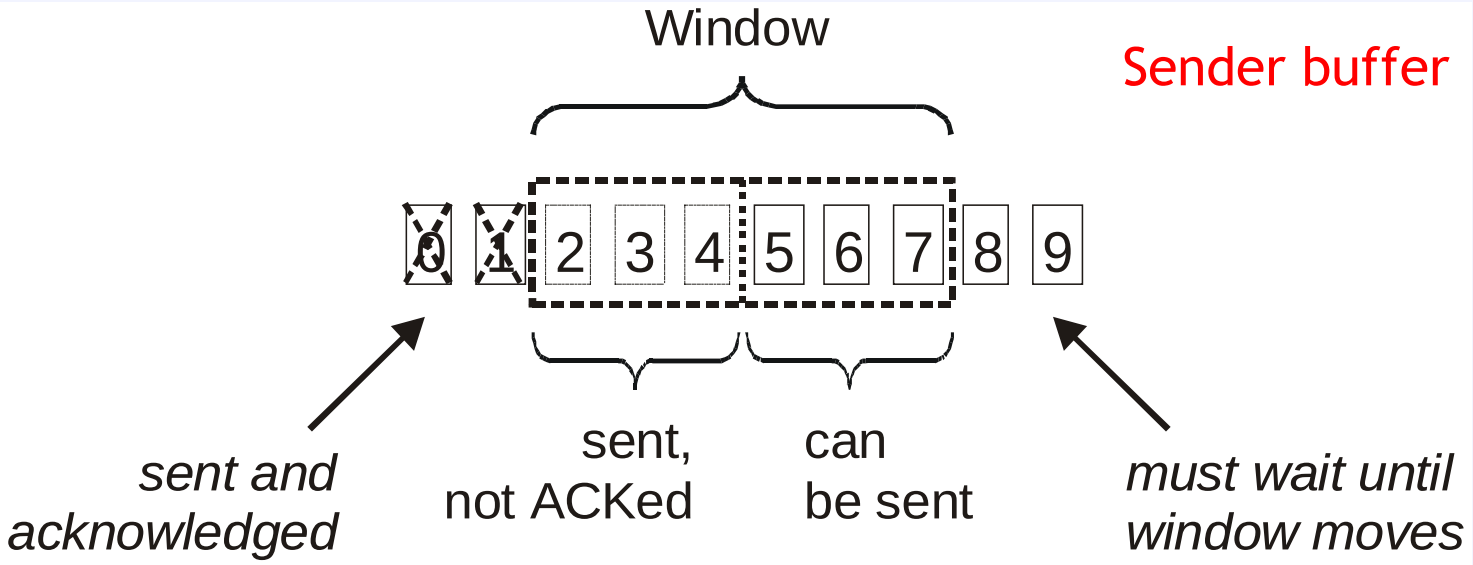
Error Control: Retransmit Timeout (RTO)

- Go-Back-N behavior in response to timeout
- RTO timer value difficult to determine:
  - too long ⇒ bad in case of msg-loss!
  - too short ⇒ risk of false alarms!
  - General consensus: too short is worse than too long; use conservative estimate
- Calculation: measure RTT (Seg# ... ACK#)
- Original suggestion in RFC 793: Exponentially Weighed Moving Average (EWMA)
  - $SRTT = (1-\alpha)\ SRTT + \alpha\ RTT$
  - $RTO = \min(UBOUND, \max(LBOUND, \beta * SRTT))$
- Depending on variation, this RTO may be too small or too large; thus, final algorithm includes variation (approximated via mean deviation)
  - $SRTT = (1-\alpha)\ SRTT + \alpha\ RTT$
  - $\delta = (1 - \beta) * \delta + \beta * [SRTT - RTT]$
  - $RTO = SRTT + 4 * \delta$

RTO calculation

- Problem: retransmission ambiguity
  - Segment #1 sent, no ACK received → segment #1 retransmitted
  - Incoming ACK #2: cannot distinguish whether original or retransmitted segment #1 was ACKed
  - Thus, cannot reliably calculate RTO!
- Solution [Karn/Partridge]: ignore RTT values from retransmits
  - Problem: RTT calculation especially important when loss occurs; sampling theorem suggests that RTT samples should be taken more often
- Solution: Timestamps option
  - Sender writes current time into packet header (option)
  - Receiver reflects value
  - At sender, when ACK arrives, RTT = (current time) - (value carried in option)
  - Problems: additional header space; facilitates NAT detection

Window management

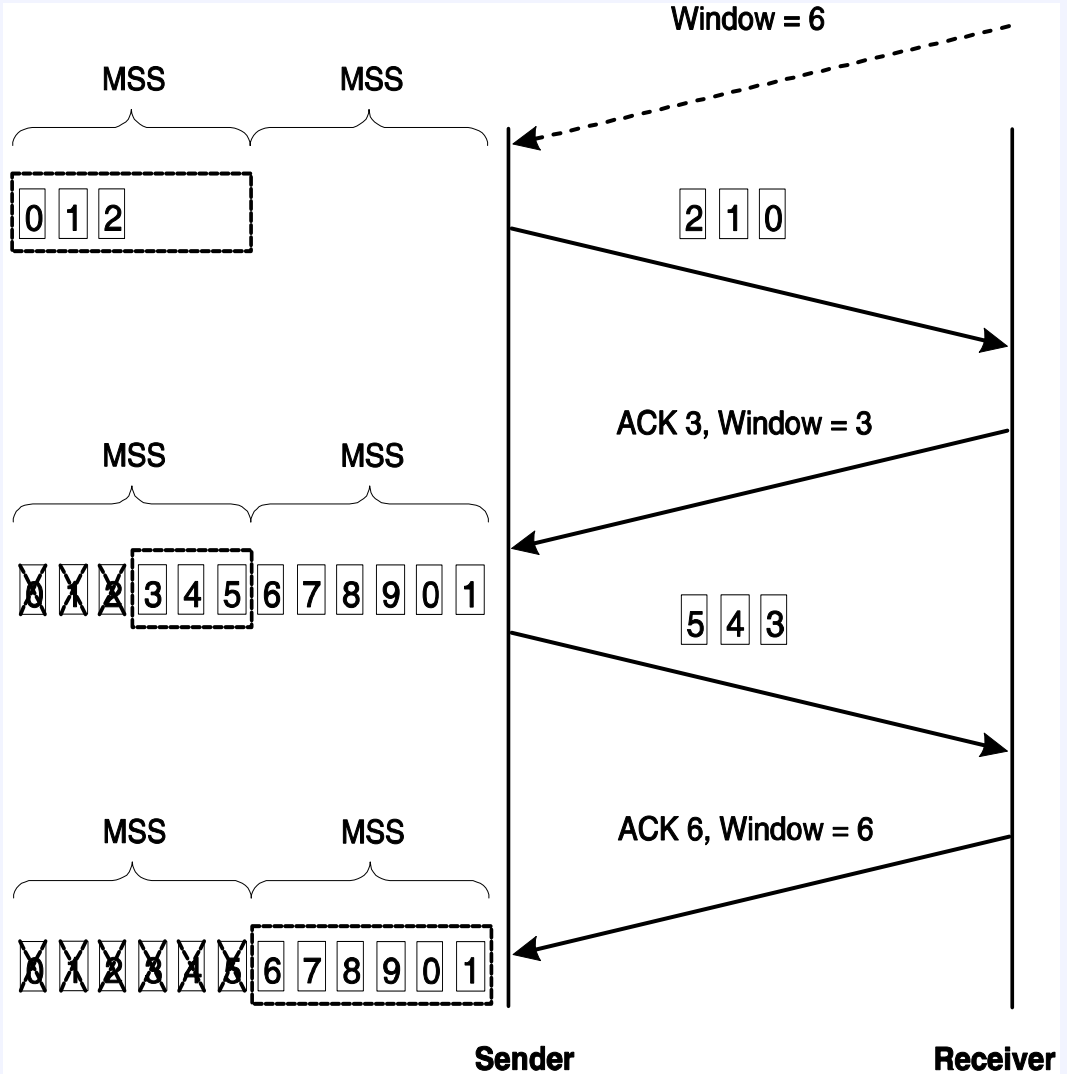


- Receiver "grants" credit (receiver window, rwnd)
  - sender restricts sent data with window
- Receiver buffer not specified
  - i.e. receiver may buffer reordered segments (i.e. with gaps)

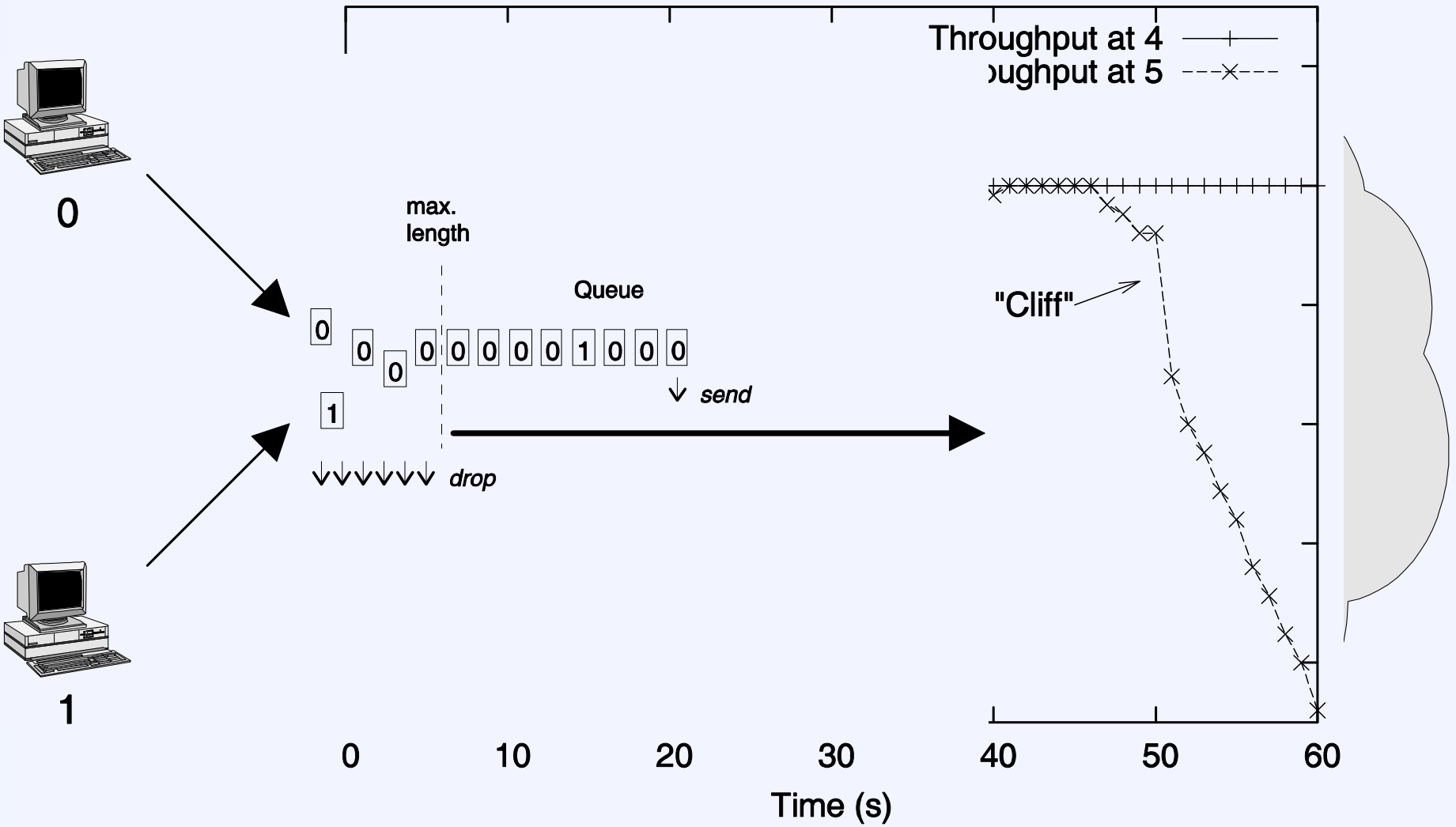
Silly Window Syndrome (SWS)

Called „congestion collapse“ by John Nagle in RFC 896

- Consider telnet: slow typing = large header overhead
  - Solution: wait until segment is filled at the sender (exception: PUSH bit)
  - But what about ls <return>?
- Nagle algorithm: sender waits until SMSS bytes can be sent
  - but 1 small segment /RTT allowed
  - A TCP implementation must support disabling Nagle
- Also, receiver mechanism: slowly reduce rwnd when less than a segment of incoming data until window boundary reached
  - Note that delayed ACKs also help: ACK 3 would not have happened



Congestion collapse



Global congestion collapse in the Internet

Craig Partridge, Research Director for the Internet Research Department at BBN Technologies:

Bits of the network would fade in and out, but usually only for TCP. You could ping. You could get a UDP packet through. Telnet and FTP would fail after a while. And it depended on where you were going (some hosts were just fine, others flaky) and time of day (I did a lot of work on weekends in the late 1980s and the network was wonderfully free then).

Around 1pm was bad (I was on the East Coast of the US and you could tell when those pesky folks on the West Coast decided to start work...).

Another experience was that things broke in unexpected ways - we spent a lot of time making sure applications were bullet-proof against failures. (..)

Finally, I remember being startled when Van Jacobson first described how truly awful network performance was in parts of the Berkeley campus. It was far worse than I was generally seeing. In some sense, I felt we were lucky that the really bad stuff hit just where Van was there to see it.

Internet congestion control: History

- **1968/69:** dawn of the Internet
- **1986:** first congestion collapse
- **1988:** "Congestion Avoidance and Control" (Jacobson)
  Combined congestion/flow control for TCP
  (also: variation change to RTO calculation algorithm)

- Goal: stability - in equilibrum, no packet is sent into the network until an old packet leaves
  - ack clocking, "conservation of packets" principle
  - made possible through window based stop+go - behaviour

- Superposition of stable systems = stable →
  network based on TCP with congestion control = stable

TCP Congestion Control: Tahoe

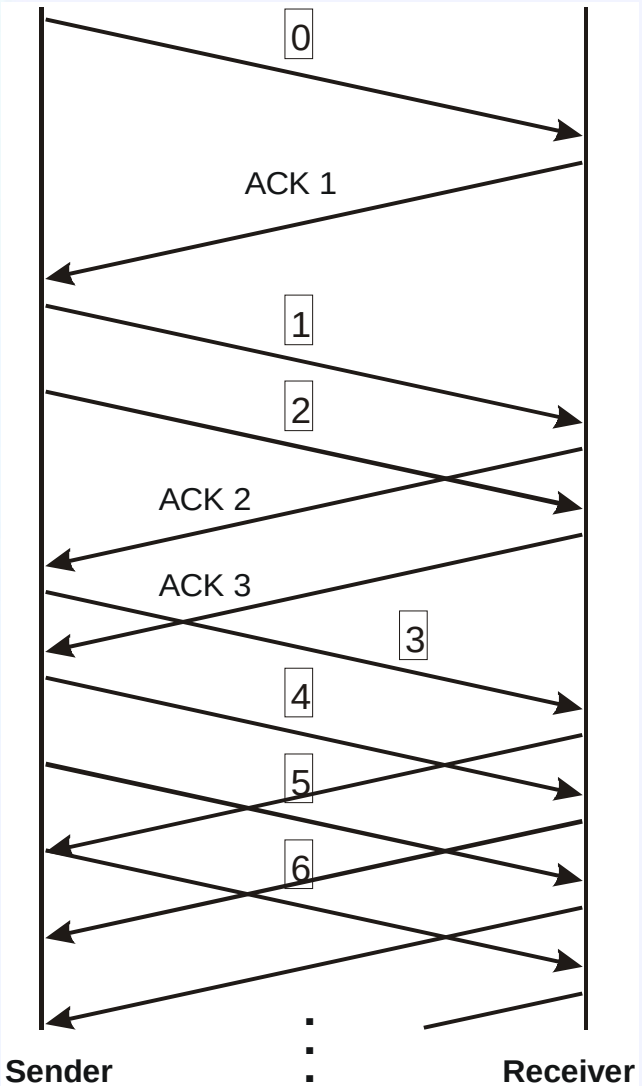
- Distinguish:
  - flow control: protect receiver against overload
    (receiver "grants" a certain amount of data ("receiver window" (rwnd)))
  - congestion control: protect network against overload
    ("congestion window" (cwnd) limits the rate: min(cwnd,rwnd) used!)

- Flow/Congestion Control combined in TCP. Two basic algorithms:
  (window unit: SMSS = Sender Maximum Segment Size, usually adjusted to Path MTU; init cwnd<=2 (*SMSS), ssthresh = usually 64k)

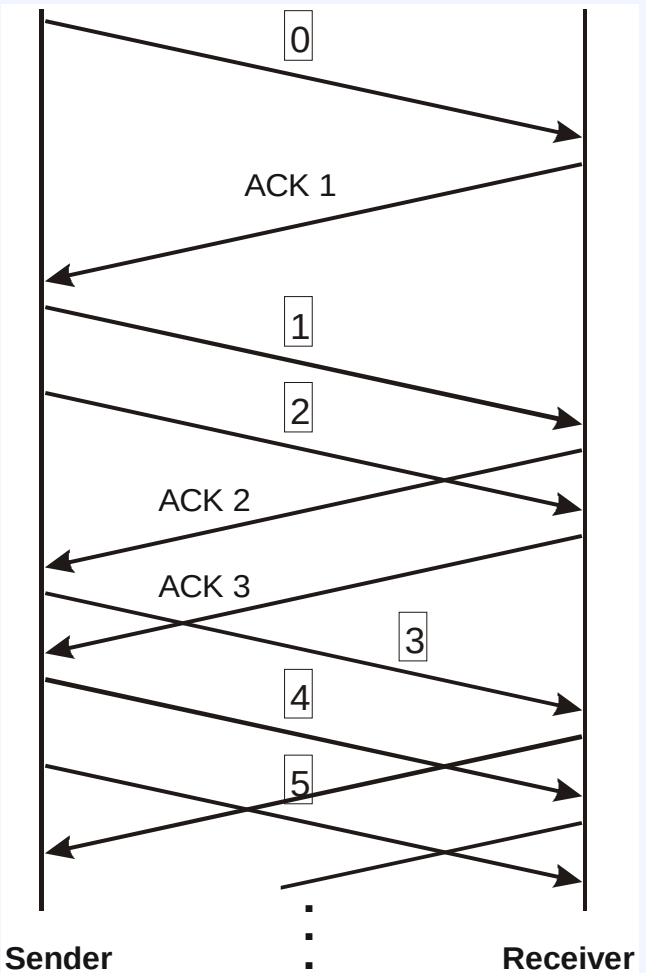
- Slow Start: for each ack received, increase cwnd by 1 (exponential growth) until cwnd >= ssthresh
- Congestion Avoidance: each RTT, increase cwnd by at most one segment (linear growth - "additive increase")

- Timeout: ssthresh = FlightSize/2 (exponential backoff - "multiplicative decrease"), cwnd = 1; FlightSize = bytes in flight (may be less than cwnd)

Slow start and Congestion Avoidance



- Slow start: 3 RTTs for 3 packets = inefficient for very short transfers
- Example: HTTP Requests
- Thus, initial window IW = min(4*MSS, max(2*MSS, 4380 byte))



Fast Retransmit / Fast Recovery (Reno)

Reasoning: slow start = restart; assume that network is empty
But even similar incoming ACKs indicate that packets arrive at the receiver!
Thus, slow start reaction = too conservative.

1. Upon reception of third duplicate ACK (DupACK): ssthresh = FlightSize/2

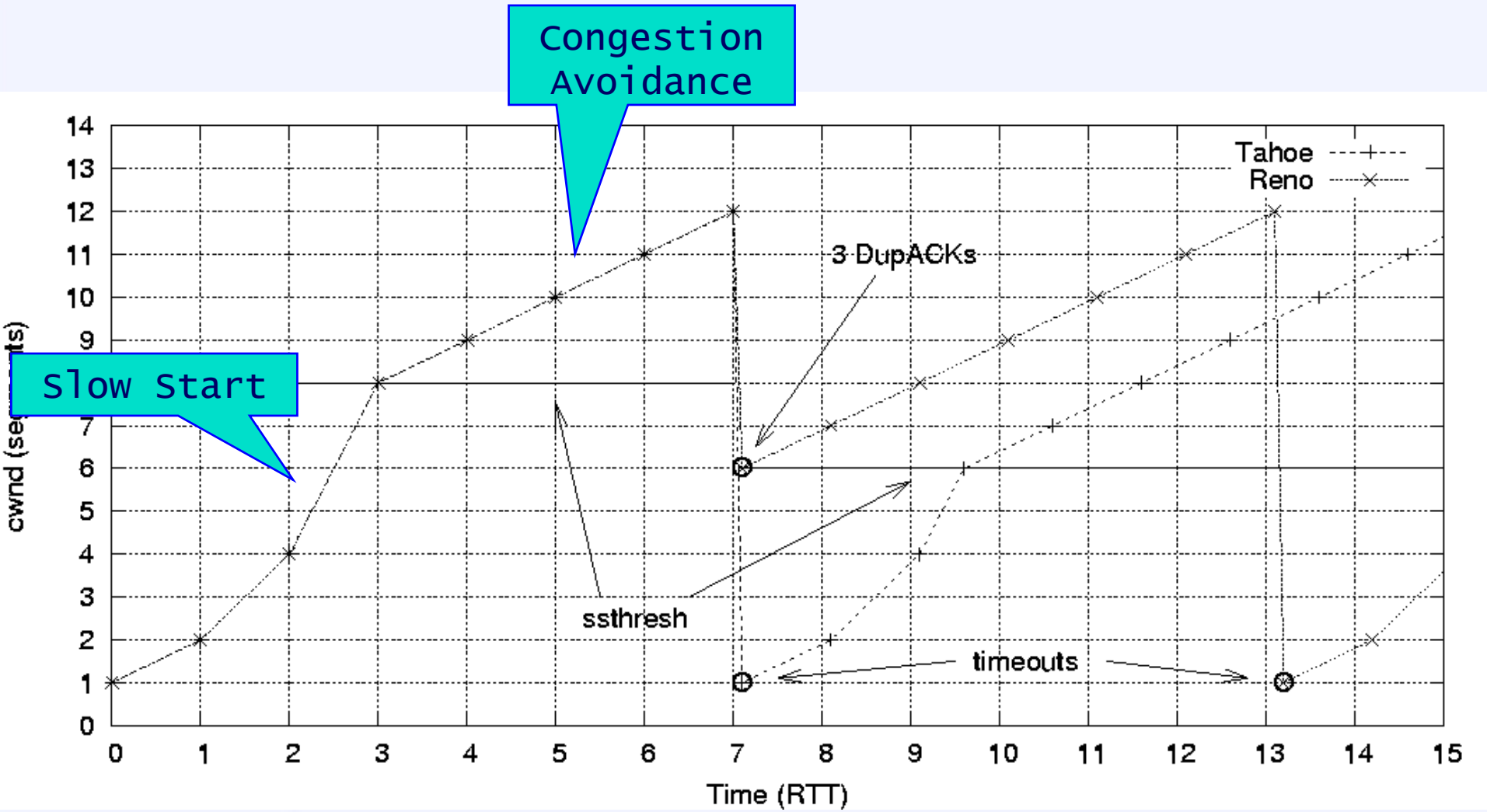
2. Retransmit lost segment (fast retransmit);
   cwnd = ssthresh + 3*SMSS
   ("inflates" cwnd by the number of segments (three) that have left the network and which the receiver has buffered)

3. For each additional DupACK received: cwnd += SMSS
   (inflates cwnd to reflect the additional segment that has left the network)

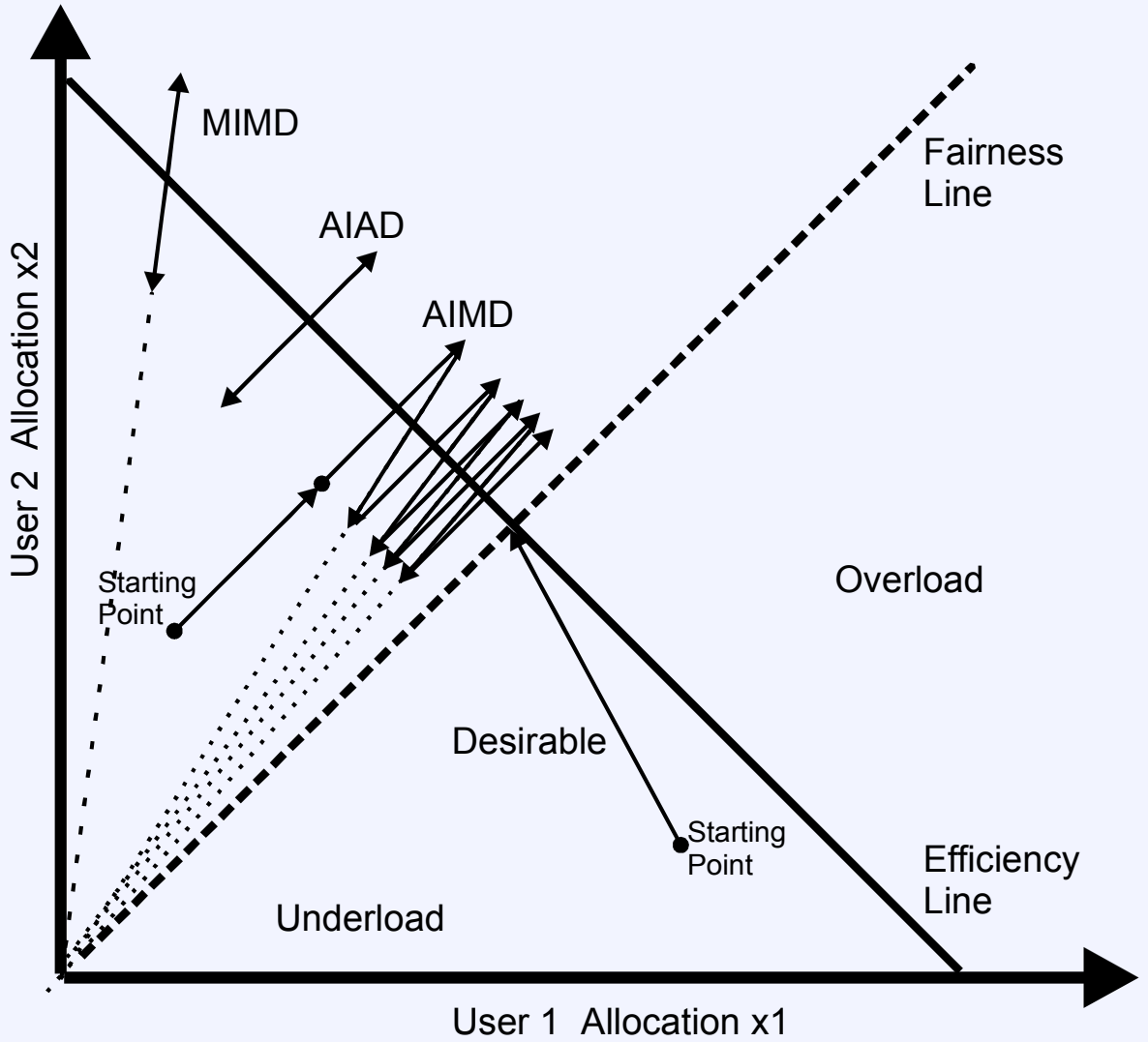
4. Transmit a segment, if allowed by the new value of cwnd and rwnd

5. Upon reception of ACK that acknowledges new data ("full ACK"):
   "deflate" window: cwnd = ssthresh (the value set in step 1)

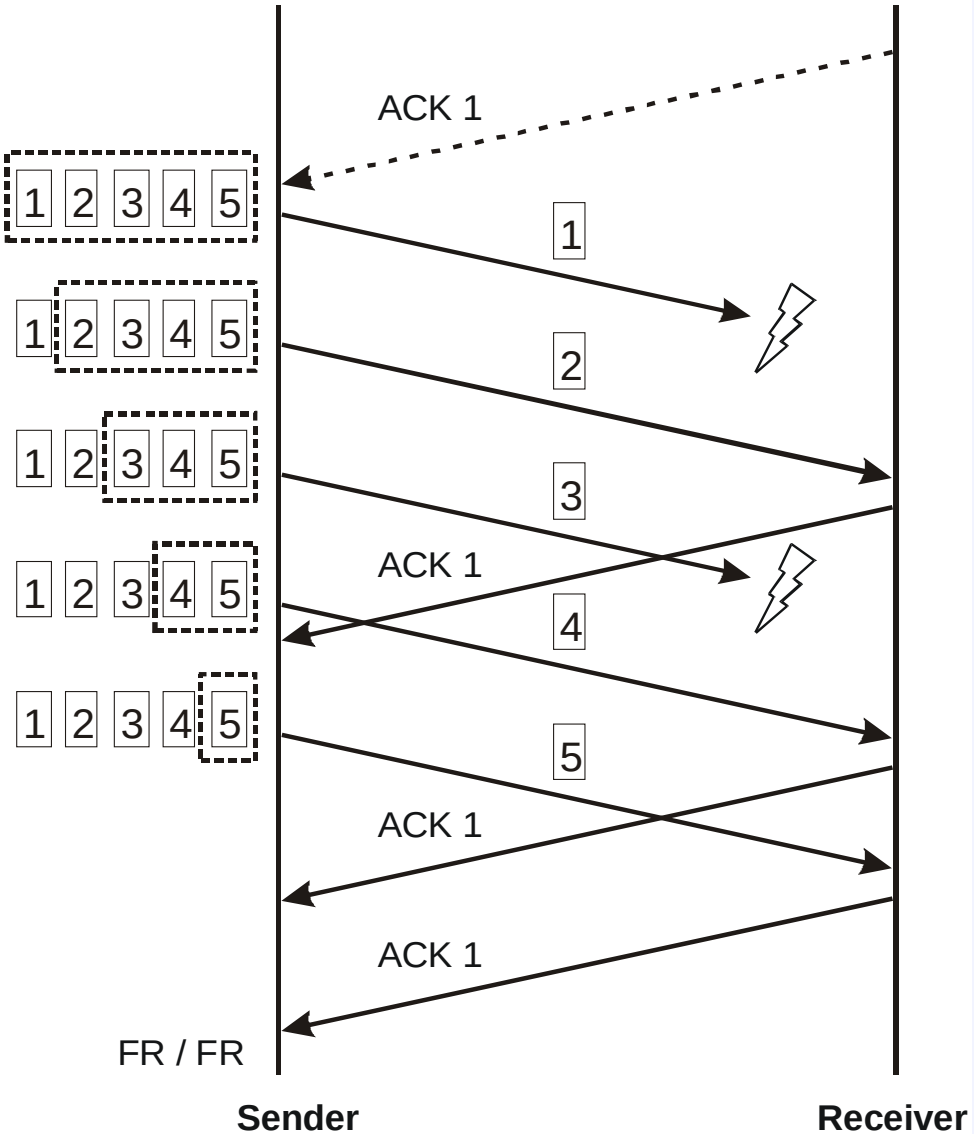
Tahoe vs. Reno



Background: AIMD



One window, multiple dropped segments

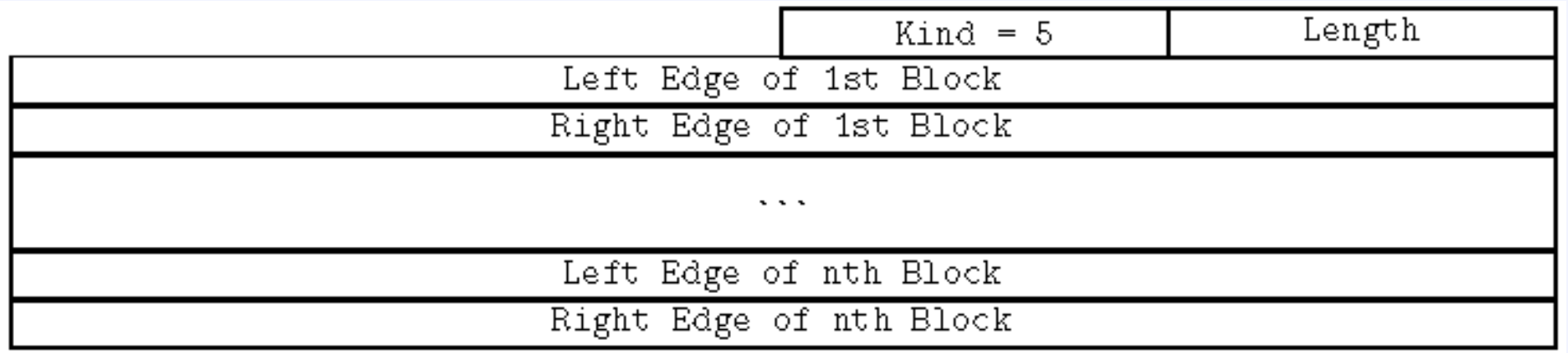


- Sender cannot detect loss of multiple segments from a single window
- Insufficient information in DupACKs
- NewReno:
  - stay in FR/FR when partial ACK arrives after DupACKs
  - retransmit single segment
  - only full ACK ends process
- Important to obtain enough ACKs to avoid timeout
  - Limited transmit: also send new segment for first two DupACKs

Example: ACK 3

Example: ACK 6

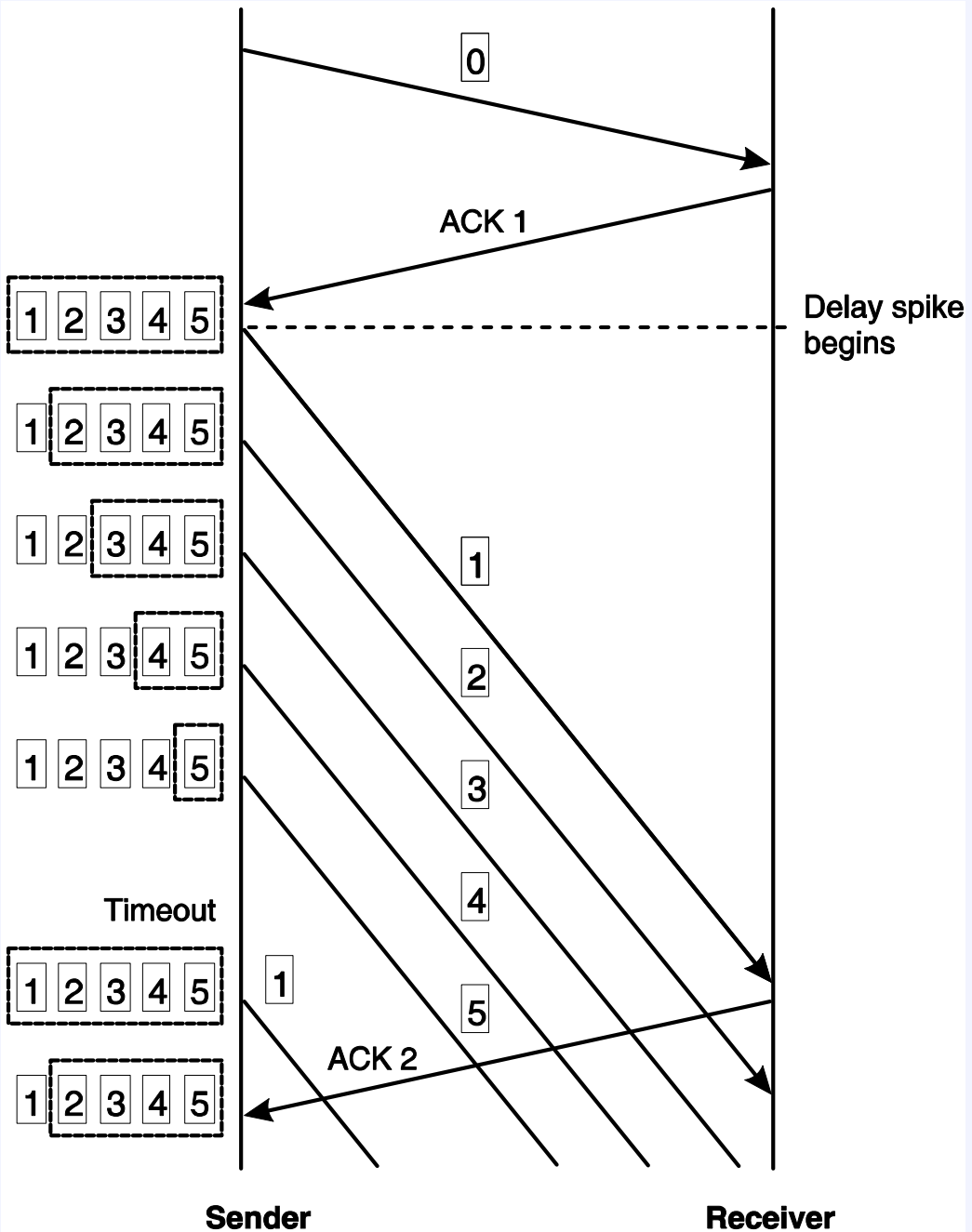
Selective ACKnowledgements (SACK)



- Example on previous slide: send ACK 1, SACK 3, SACK 5 in response to segment #4
- Better sender reaction possible
  - Reno and NewReno can only retransmit a single segment per window
  - SACK can retransmit more (RFC 3517 – maintain scoreboard, pipe variable)
  - Particularly advantageous when window is large (long fat pipes)
- but: requires receiver code change
- Extension: DSACK informs the sender of duplicate arrivals

Spurious timeouts

- Common occurrence in wireless scenarios (handover): sudden delay spike
- Can lead to timeout
  → slow start
  - But: underlying assumption: "pipe empty" is wrong! ("spurious timeout")
  - Old incoming ACK after timeout should be used to undo the error
- Several methods proposed
  Examples:
  - Eifel Algorithm: use timestamps option to check: timestamp in ACK < time of timeout?
  - DSACK: duplicate arrived
  - F-RTO: check for ACKs that shouldn't arrive after Slow Start



Appropriate Byte Counting

- Increasing in Congestion Avoidance mode: common implementation (e.g. Jan'05 FreeBSD code): cwnd += SMSS*SMSS/cwnd for every ACK (same as cwnd += 1/cwnd if we count segments)
  - Problem: e.g. cwnd = 2: 2 + 1/2 + 1/ (2+1/2)) = 2+0.5+0.4 = 2.9 thus, cannot send a new packet after 1 RTT
  - Worse with delayed ACKs (cwnd = 2.5)
  - Even worse with ACKs for less than 1 segment (consider 1000 1-byte ACKs) → too aggressive!

- Solution: Appropriate Byte Counting (ABC)
  - Maintain bytes_acked variable; send segment when threshold exceeded
  - Works in Congestion Avoidance; but what about Slow Start?
    - Here, ABC + delayed ACKs means that the rate increases in 2*SMSS steps
    - If a series of ACKs are dropped, this could be a significant burst ("micro-burstiness"); thus, limit of 2*SMSS per ACK recommended

Limited Slow Start and cwnd Validation

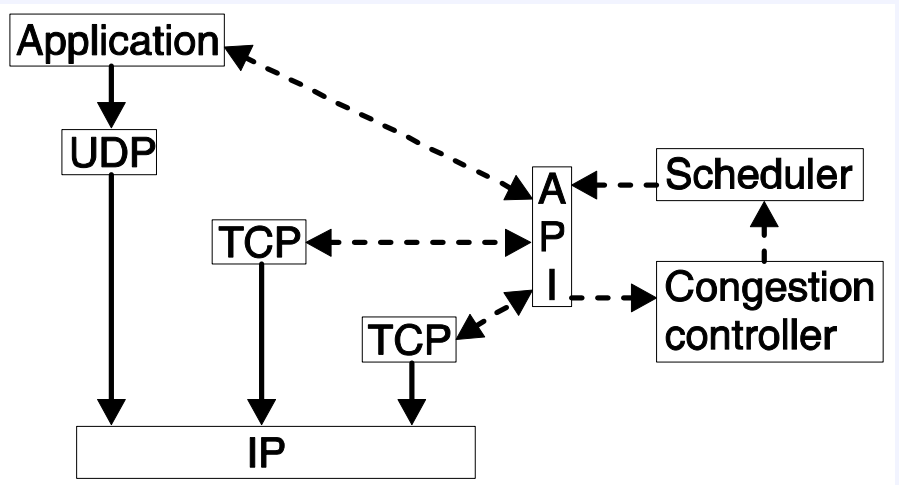
- Slow start problems:
  - initial ssthresh = constant, not related to real network
    this is especially severe when cwnd and ssthresh are very large
    - Proposals to initially adjust ssthresh failed: must be quick and precise
  - Assume: cwnd and ssthresh are large, and avail.bw. = current window + 1 SMSS/RTT ?
    - Next updates (cwnd++ for every ACK) will cause many packet drops

- Solution: Limited Slow Start
  - cwnd <= max_ssthresh: normal operation; recommend. max_ssthresh=100 SMSS
  - else K = int(cwnd/(0.5*max_ssthresh), cwnd += int(MSS/K)
  - More conservative than Slow Start:
    for a while cwnd+=MSS/2, then cwnd+=MSS/3, etc.

- Cwnd validation
  - What if sender stops, or does not send as much as it could?
    - maintain cwnd = wrong if break is long (not related to real network anymore)
    - reset = too conservative if break is short
    - Solution: slowly decay TCP parameters - cwnd /= 2 every RTT,
      ssthresh = between previous and new cwnd

Maintaining congestion state

- TCP Control Block (TCB): information such as RTO, scoreboard, cwnd, ..
- Related to network path, yet separately stored per TCP connection
  - Compare: layering problem of PMTU storage
- TCB interdependence: affects initialization phase
  - Temporal sharing: learn from previous connection (e.g. for consecutive HTTP requests)
  - Ensemble sharing: learn from existing connections here, some information should change - e.g. cwnd should be cwnd/n, n = number of connections; but less aggressive than "old" implementation
- Congestion Manager
  - One entity in the OS maintains all the
  - congestion control related state
  - Used by TCP's and UDP based applications
  - Hard to implement, not really used



Explicit Congestion Notification (ECN)

- Active Queue Management
  - monitor queue, do not just drop upon overflow ⇒ more intelligent decisions
  - maintain low average queue length, alleviate phase effects, enforce fairness

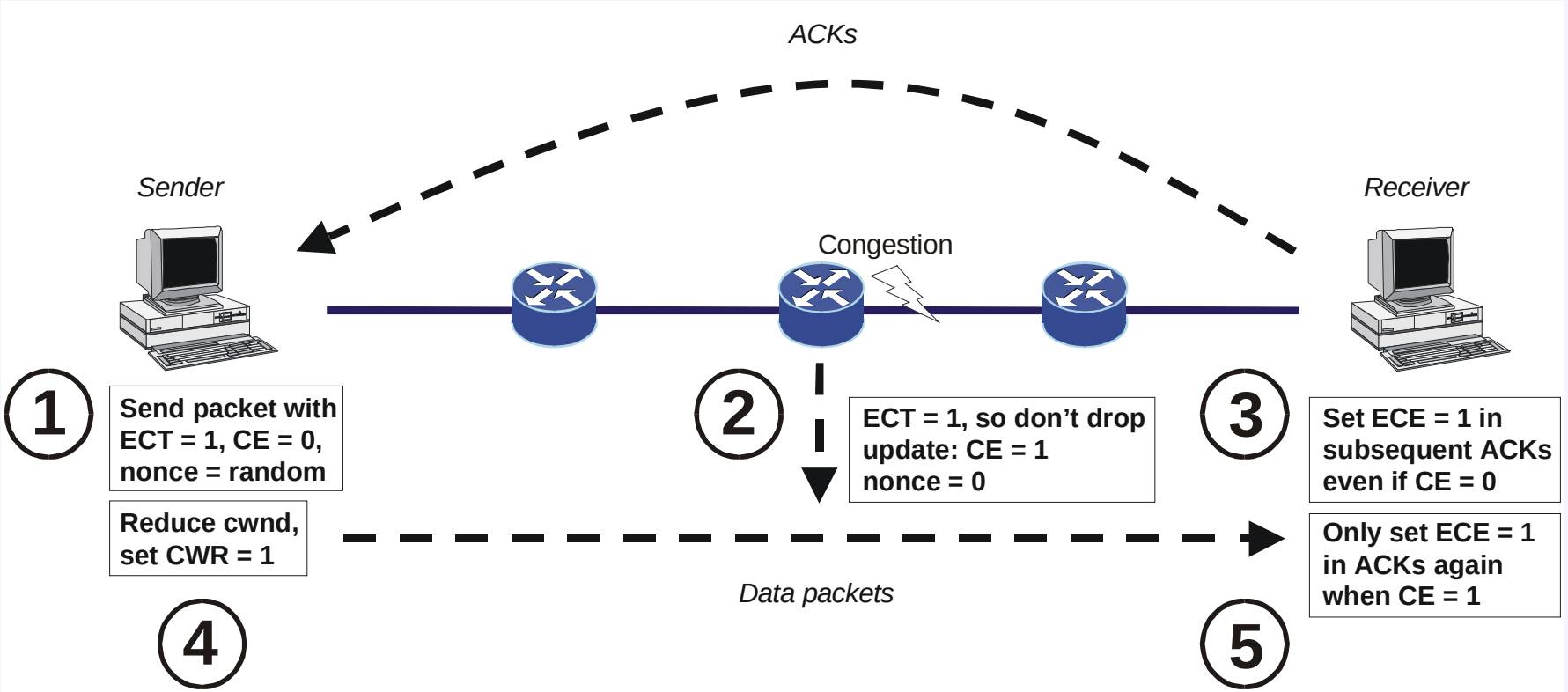
- Explicit Congestion Notification (ECN)
  - Instead of dropping, set a bit; reduced loss ⇒ major benefit!

- Receiver informs sender about bit; sender behaves as if a packet was dropped
  ⇒ actual communication between end nodes and the network

- Typical incentives:
  - sender = server; efficiently use connection, fairly distribute bandwidth
    - use ECN as it was designed
  - receiver = client; goal = high throughput, does not care about others
    - ignore ECN flag, do not inform sender about it

- Need to make it impossible for receiver to lie about ECN flag when it was set
  - Solution: nonce = random number from sender, deleted by router when setting ECN
  - Sender believes „no congestion" iff correct nonce is sent back

ECN in action



- Nonce provided by bit combination:
  - ECT(0): ECT=1, CE=0
  - ECT(1): ECT=0, CE=1
- Nonce usage specification still experimental

Fighting TCP SYN attacks

- TCP SYN attack
  - DoS attack - flood a server until it's down, ideally with packets that cause work
  - Note: per-flow state not scalable
  - TCP needs per-flow state (connection state, address, port numbers, ..)
  - 1 SYN packet: search through existing connections + allocate memory
  - TCP SYN attack exploits TCP scalability problem!

- Solution
  - Sequence number negotiated at connection setup
  - Idea:
    - do not maintain state after SYN at server
    - encode cipher in sequence number from server to client
    - Client must reflect it ⇒ check integrity; if okay, generate state from ACK
  - Only requires changes at the server
  - Not specified in RFC - no specification change needed
  - See http://cr.yp.to/syncookies.html for details (how to activate in Linux, ..)

Known issues with TCP

Current IETF concern: TCP security

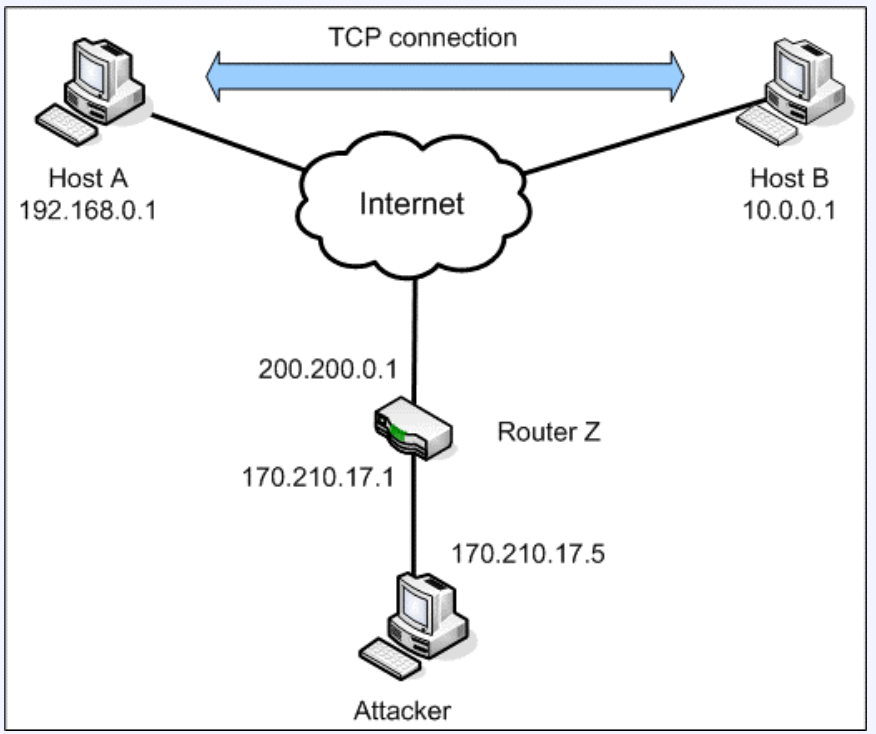
- Historic viewpoint: can an attacker blindly disturb a TCP connection?
  - Hardly: would have to know 4-tuple (src/dst addr, src/dst port and seqno)
  - Thus, no countermeasures in TCP

- Assumption no longer correct!
  [Paul Watson: "Slipping in the Window" (cansecwest/core04 conference)]
  - Window size larger for high speed links (RFC 1323) ⇒ larger number of working seqnos
  - Some applications use long lived connections; e.g. H.323, BGP (major concern!)
    ⇒ longer time available for attacker
  - Also, such long lived connections may have predictable IP addresses / ports
    ⇒ better chances of guessing correct 4-tuple
  - RST attack
    - cause connection to be torn down; works because any RST in current window accepted
    - Mitigation: only accept RST with next expected seqno
  - SYN attack
    - in old spec, SYN with acceptable seqno is answered with RST
    - Mitigation: answer with ACK, which is answered with RST (where new rule applies)
  - DATA attack
    - can lead to "ACK war" (sender / receiver negotiation fails) or corruption
    - Mitigation: always check range of ACK

TCP security /2

- Note: BGP problem long known; awareness issue!
  - RFC 2385 (Proposed Standard, 1998) specifies a MD5 message digest for TCP
  - IPSec authentication can also solve the problem
  - So can authentication based on Timestamps option

- Recent discussion: what about ICMP?
  - Messages can indicate reachability problems, but also source quench and MTU (still beneficial for convergence with new PMTUD, but a security problem)
  - Many pro's and con's to ICMP processing
  - Consider figure: should router Z accept ICMP packets from 170.210.17.1 which tell Host A that Host B is unreachable?



Source: http://www.gont.com.ar/papers/icmp-errors/

Some reasons for TCP CC. stability

"Congestion Avoidance and Control", Van Jacobson, SIGCOMM'88:

- Exponential backoff:
  "For a transport endpoint embedded in a network of unknown topology and with an unknown, unknowable and constantly changing population of competing conversations, only one scheme has any hope of working - exponential backoff - but a proof of this is beyond the scope of this paper."

- Conservation of packets:
  "The physics of flow predicts that systems with this property should be robust in the face of congestion."

- Additive Increase, Multiplicative Decrease:
  Not explicitely cited as a stability reason **in the paper**!
  - ...but in 1000's of other papers!

"Proofs" of TCP stability

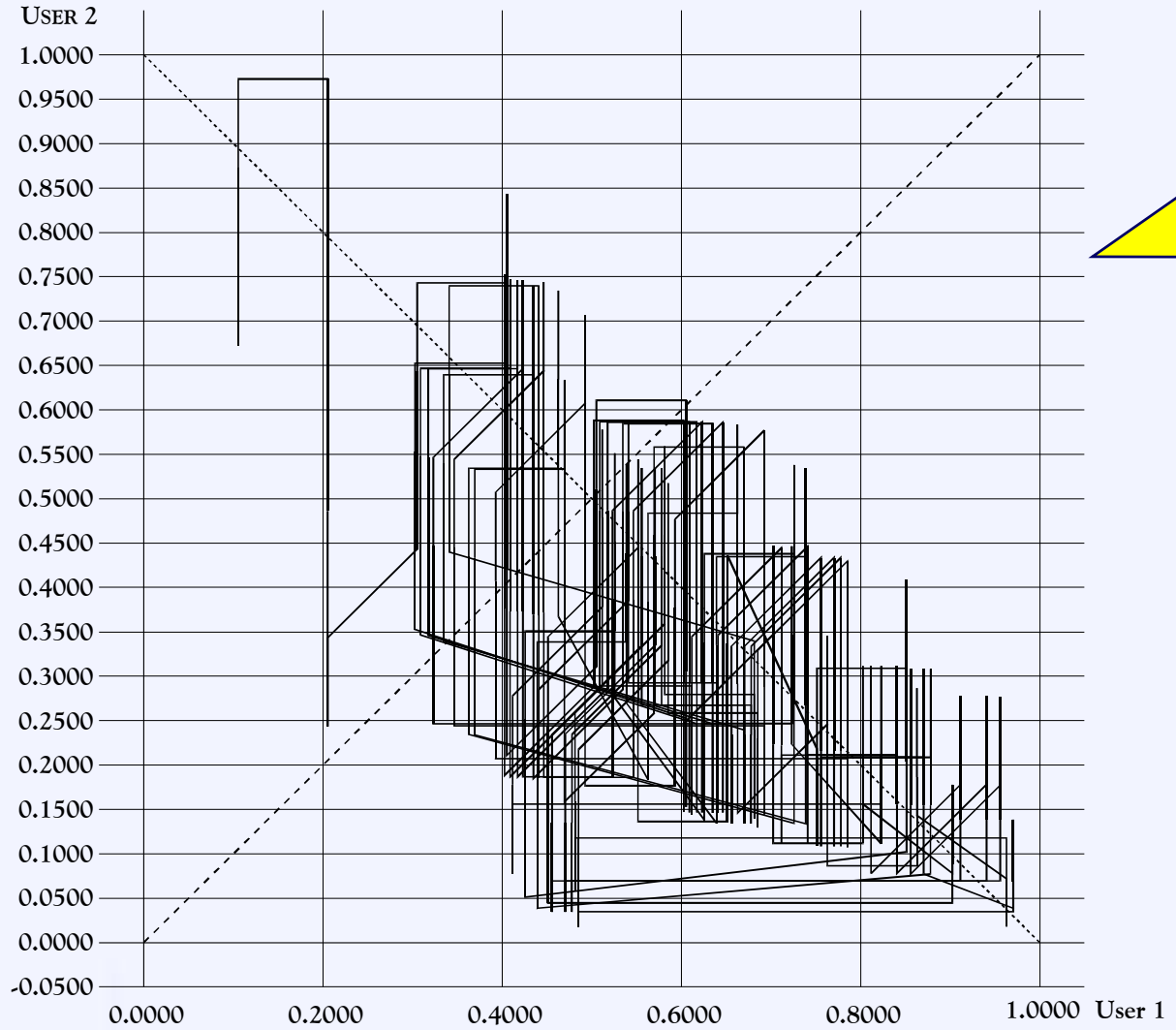
- AIMD:
  Chiu/Jain: diagram + algebraic proof of homogeneous RTT case

- steady-state TCP model: window size ~ 1/sqrt(p)
  (p = packet loss)

- Johari/Tan, Massoulié, ..:
  - local stability, neglect details of TCP behaviour (fluid flow model, ..)
  - assumption:
    "queueing delays will eventually become small relative to propagation delays"

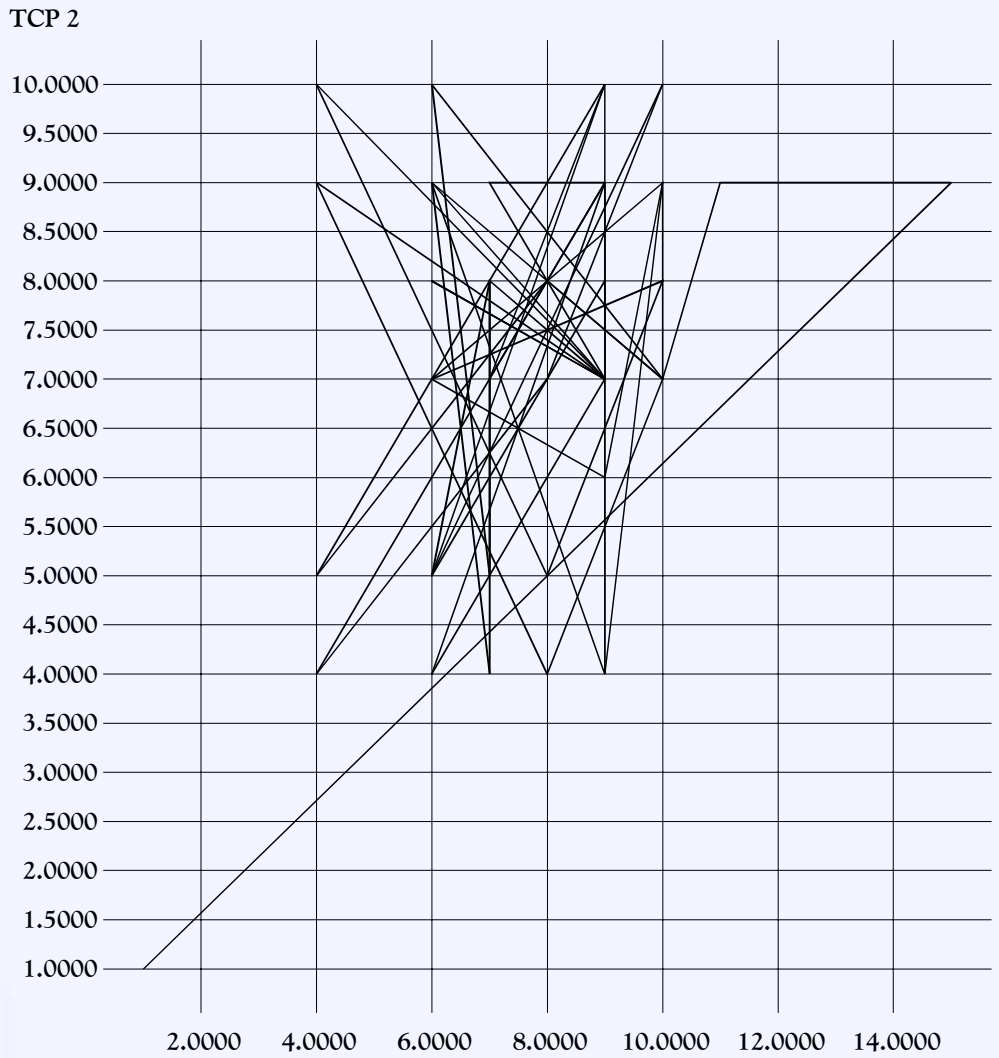
- Steven Low:
  - Duality model (based on utility function / F. Kelly, ..):
    Stability depends on delay, capacity, load and AQM

How Stable is AIMD / async. RTT?



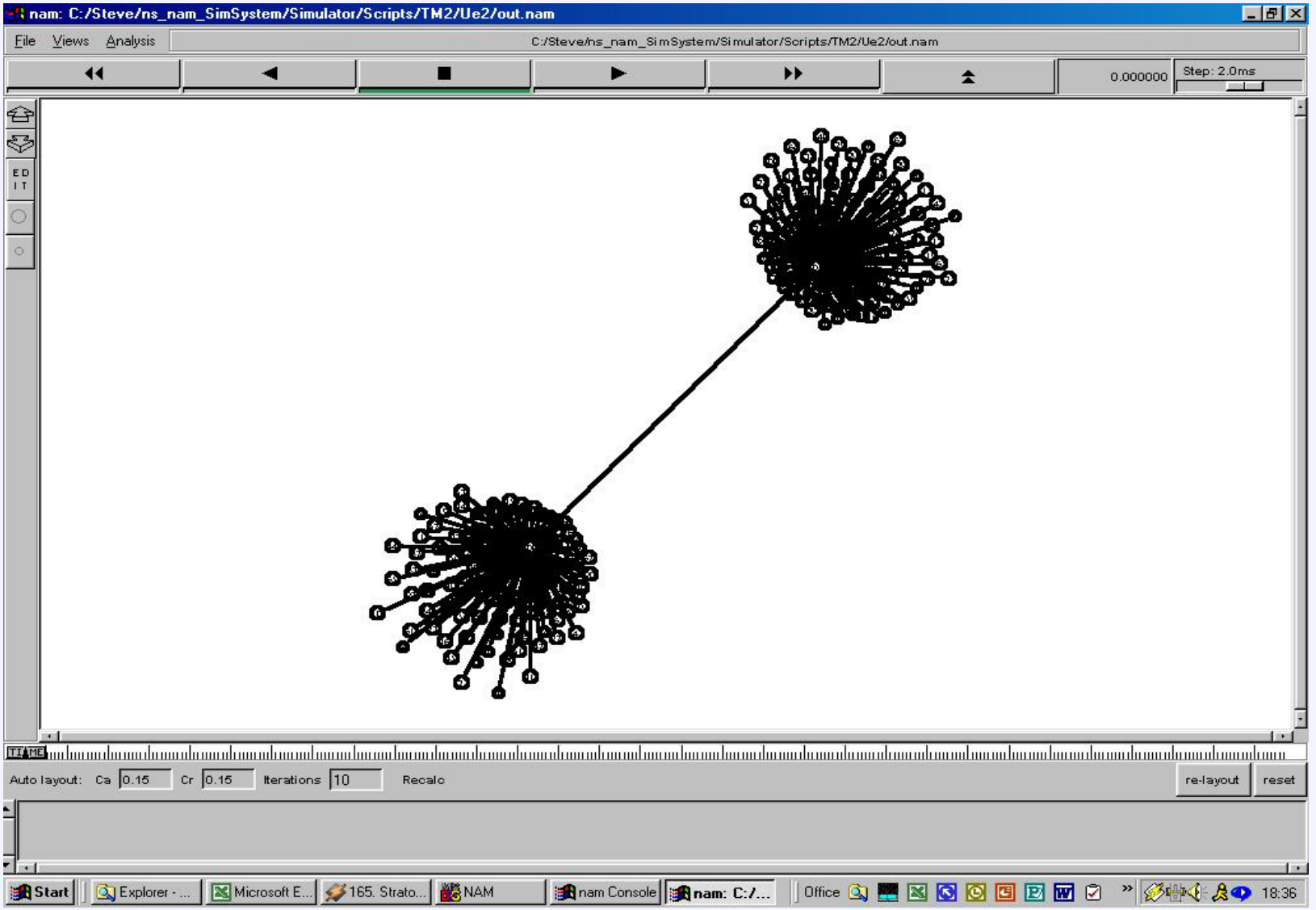
- Simple simulation (no queues, ..)
- RTT: 7 vs. 2
- AI=0.1, MD=0.5
- Simul. time=175

Is AIMD distorted in TCP?

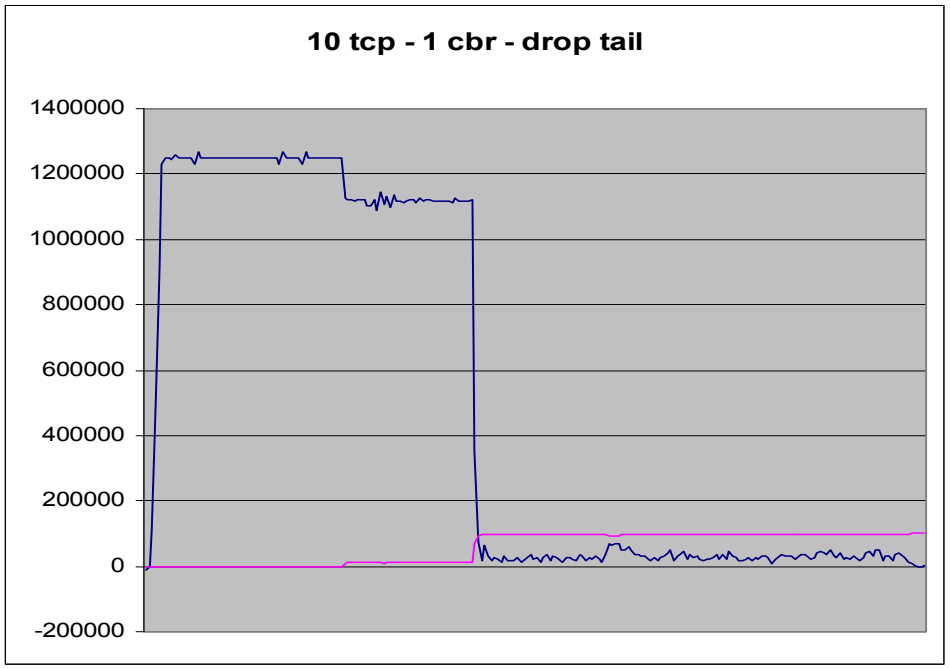


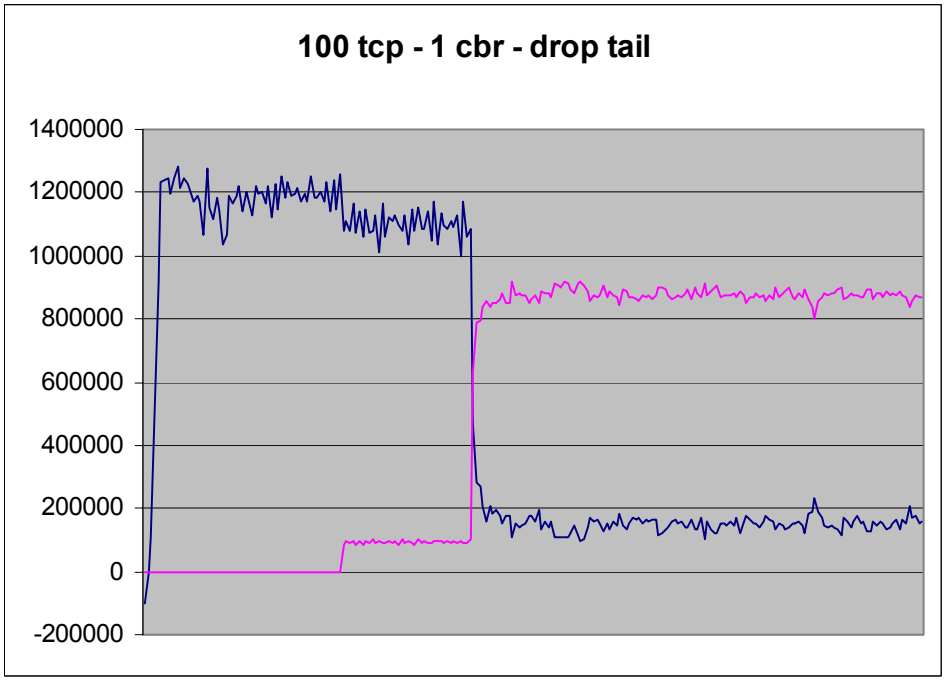
- ns-2 simulator
- TCP Tahoe
- equal RTT
- 1 bottleneck link

TCP vs. UDP: a simple simulation example



It doesn't look good





- For more details, see:
  Promoting the Use of End-to-End Congestion Control in the Internet.
  Floyd, S., and Fall, K..
  IEEE/ACM Transactions on Networking, August 1999.

TCP-friendliness

- TCP dominant - therefore, Internet definition of fairness: TCP-friendliness
  "A flow is TCP-compatible (TCP-friendly) if, in steady state, it uses no more bandwidth than a conformant TCP running under comparable conditions."

- But...
  - TCP regularly increases the queue length and causes loss
    ⇒ detect congestion when it is already (ECN: almost) too late!
    - possible to have more throughput with smaller queues and less loss
      ... but: exceed rate of TCP under similar conditions ⇒ not TCP-friendly!
  - What if I send more than TCP *in the absence* of competing TCP's?
    - can such a mechanism exist?
    - yes! TCP itself, with max. window size = bandwidth * RTT
    - Does this mean that TCP is not TCP-friendly?
  - Details missing from the definition:
    - parameters + version of "conformant TCP"
    - duration! short TCP flows are different than long ones
  - TCP-friendliness = compatibility of new mechanisms with old mechanism
    - there was research since the 80's! e.g. new knowledge about network measurements
  - TCP rate depends on RTT - how does this relate to intuitive "fairness" notion?

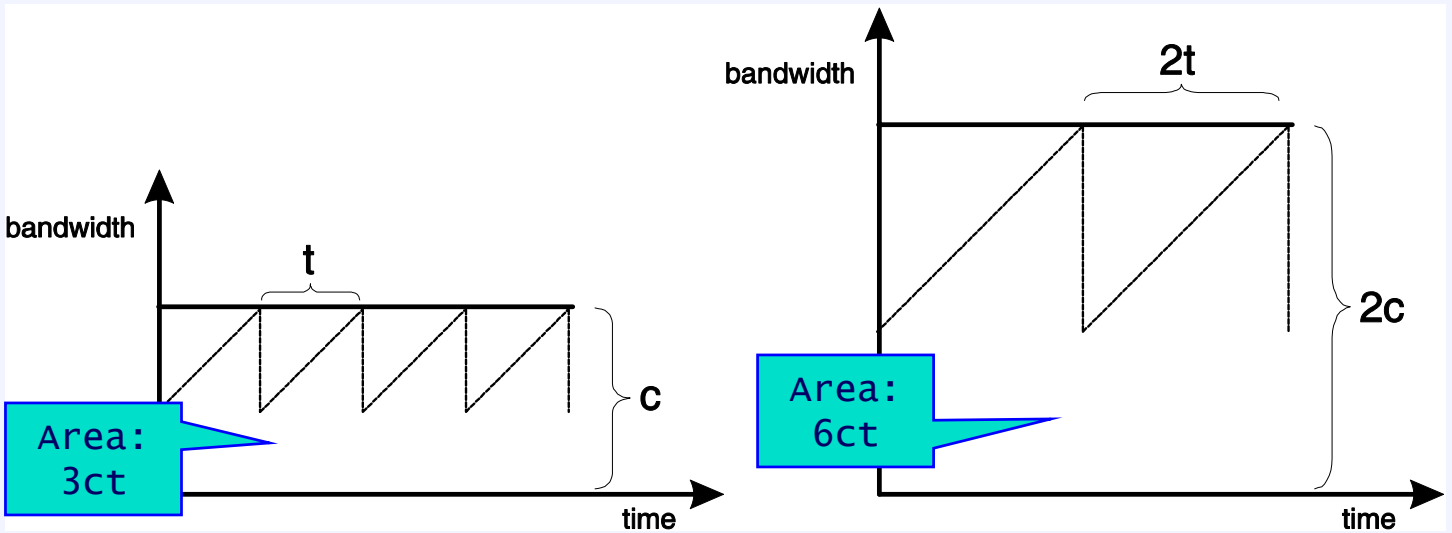
TCP with High Speed links

- TCP over "long fat pipes": large bandwidth*delay product
  - long time to reach equilibrium, MD = problematic!
  - From RFC 3649 (HighSpeed RFC, Experimental):

> For example, for a Standard TCP connection with 1500-byte packets and a 100 ms round-trip time, achieving a steady-state throughput of 10 Gbps would require an average congestion window of 83,333 segments, and a packet drop rate of at most one congestion event every 5,000,000,000 packets (or equivalently, at most one congestion event every 1 2/3 hours). This is widely acknowledged as an unrealistic constraint.

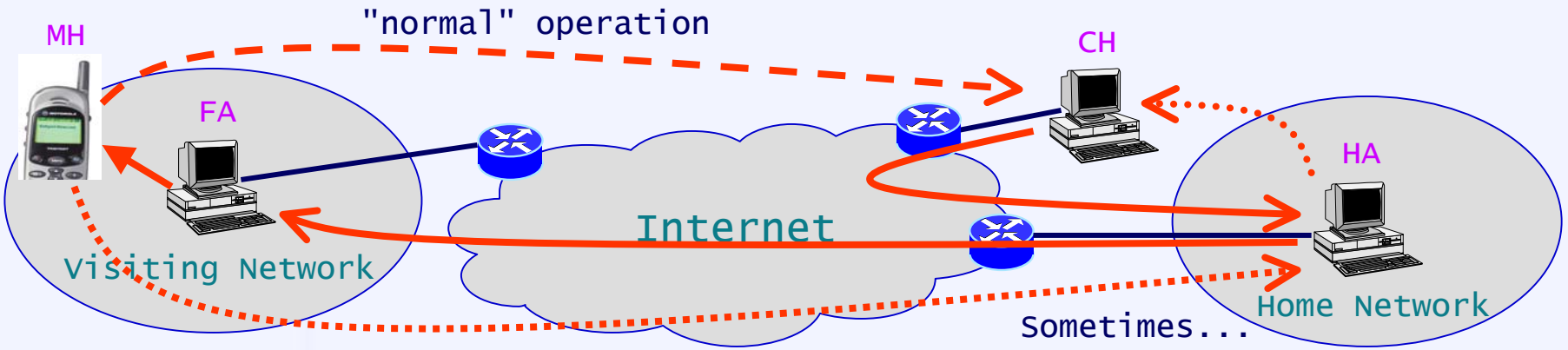
Theoretically, utilization independent of capacity

But: longer convergence time



TCP with asymmetric routing

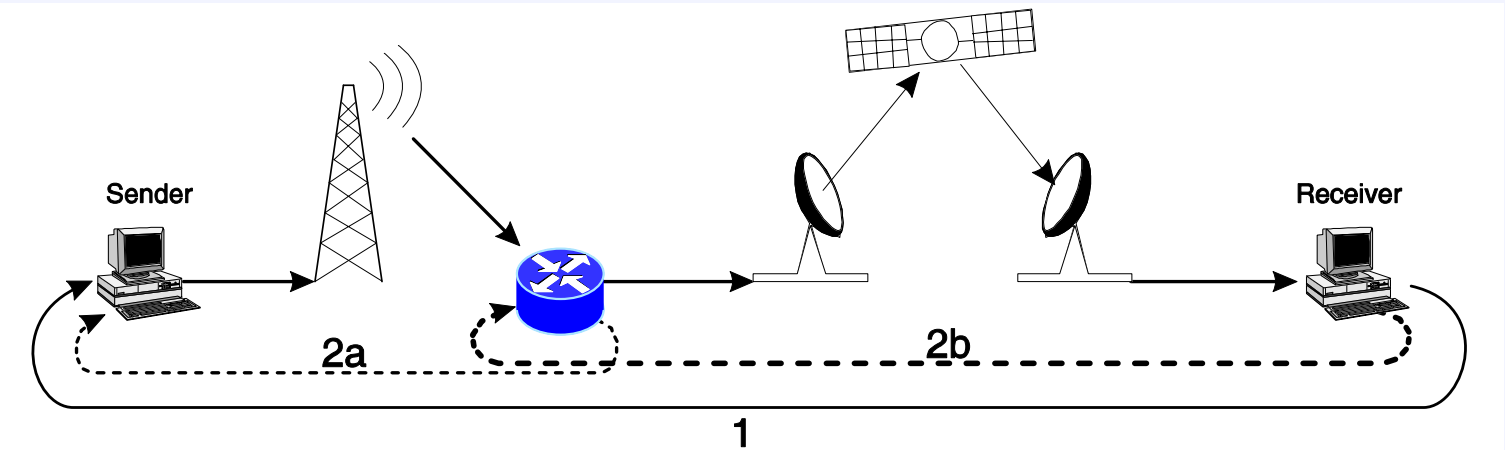
- TCP in asymmetric networks
  - incoming throughput (high capacity link) can be limited by rate of outgoing ACKs (ACK compaction, ACK congestion)
  - Mitigation:
    - Delayed ACKs
    - ACK suppression (selectively drop ACKs)
    - TCP header compression
  - triangular routing with Mobile IP(v4) and FA-Care-of-address can lead to unnecessarily large RTT (and hence large RTT fluctuations)



TCP in noisy environments / over satellite

- TCP over noisy links: problems with "packet loss = congestion"
  - Usually wireless links, where delay fluctuations from link layer ARQ and handover are also issues (mitigation: spurious timeout detection schemes)

- Satellites combine several problems
  - Long delay
  - High capacity
  - Wireless (but usually not noisy (for TCP) because of link layer FEC)
  - Can be asymmetric (e.g. direct satellite downlink, 56k modem uplink)



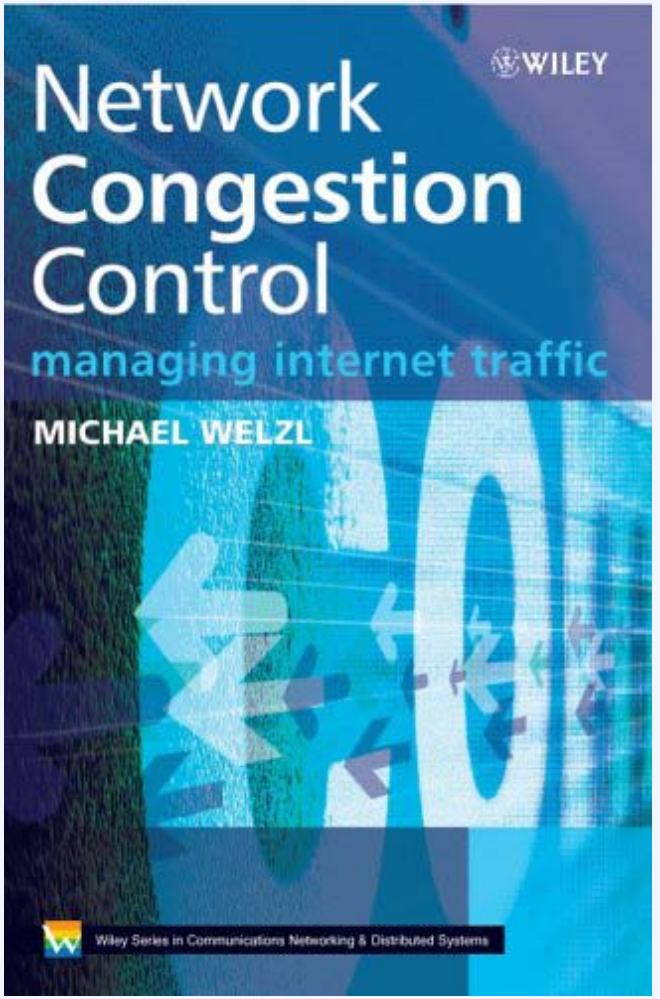
Performance Enhancing Proxy (PEP)

References

- Michael Welzl, "Network Congestion Control: Managing Internet Traffic", John Wiley & Sons, Ltd., August 2005, ISBN: 047002528X

- M. Hassan and R. Jain, "High Performance TCP/IP Networking: Concepts, Issues, and Solutions", Prentice-Hall, 2003, ISBN:0130646342

- M. Duke, R. Braden, W. Eddy, E. Blanton: "A Roadmap for TCP Specification Documents", Internet-draft draft-ietf-tcpm-tcp-roadmap-06.txt, http://www.ietf.org/internet-drafts/draft-ietf-tcpm-tcp-roadmap-06.txt (in RFC Editor Queue)



Thank you!

Questions?