15-441 15-641 Computer Networking

Lecture 11: Congestion Control

Justine Sherry
Peter Steenkiste

Fall 2017

www.cs.cmu.edu/~prs/15-441-F17

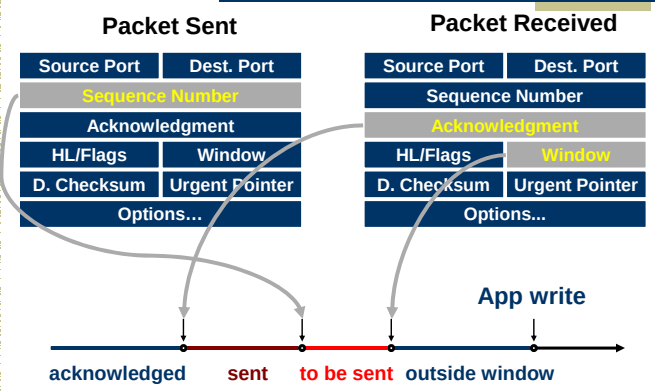
Outline

- Transport introduction
- TCP connection establishment
- Error recovery and flow control
- Making things work in TCP
- Congestion control
- Transport optimization and futures

TCP = Go-Back-N Variant

- TCP uses a sliding window with cumulative acks
  - Receiver can only return a single "ack" sequence number
  - Acknowledges all bytes with a lower sequence number
  - Starting point for retransmission
- But: sender only retransmits one packet after timeout
  - Reason: only knows that that specific packet is lost
    - Network is congested → should not overload it with questionable retransmits
- Receiver stores out of order packets
  - Can be used after the sender "fills the gap"
- Error control is based on byte sequences, not packets

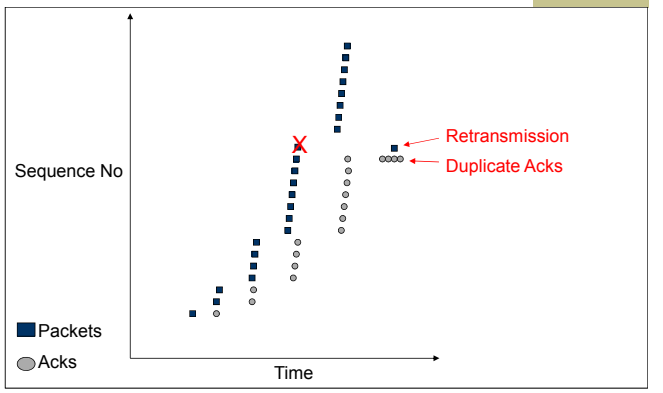
Window Flow Control: Send Side



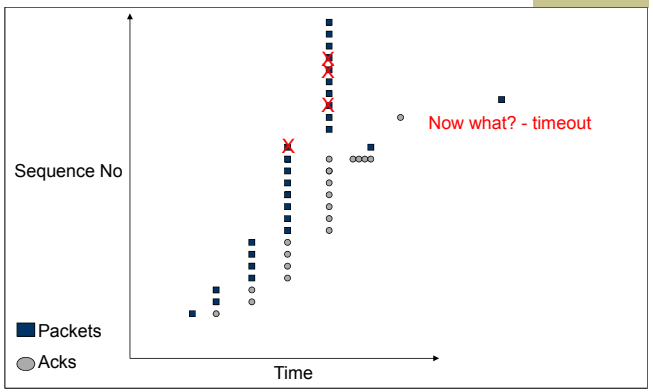
Duplicate ACKs (Fast Retransmit)

- Basic Go-Back-N incurs timeout for every loss
  - Can we do better? How about a NACK?
- Receiver sends “duplicate ack” for out of order packets
  - Repeated acks for the same sequence
  - Serves as a NACK – no room in header for real NACK!
- When can duplicate acks occur?
  - Loss
  - Packet re-ordering – oops! Unnecessary retransmit
- Solution - assume re-ordering is infrequent :
  - Receipt of 3 or more duplicate acks is indication of loss
  - Sender does not wait for timeout to retransmit packet
  - When does this fail?

Duplicate ACKs (Fast Retransmit)



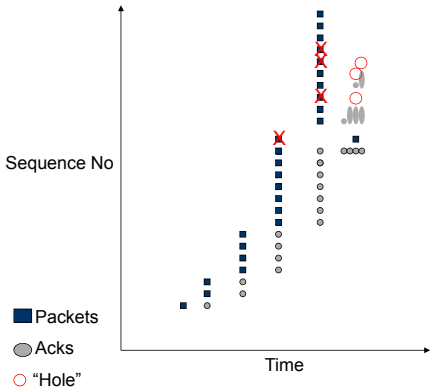
How about Multiple Losses?



SACK

- Cumulative acks provide little information
  - How many packets were really lost?
  - Becomes a problem as windows get bigger
- Selective acknowledgement (SACK) essentially adds a bitmask of packets received
  - Implemented as a TCP option
  - Encoded as a set of received byte ranges (max of 4 ranges/often max of 3)
- When to retransmit?
  - Still need to deal with reordering → wait for out of order by 3pkts

Selective ACK (SACK)

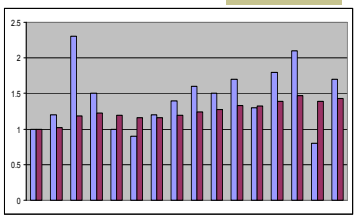


Round-trip Time Estimation

- Wait at least one RTT before retransmitting
- Importance of accurate RTT estimators:
  - Low RTT estimate: unneeded retransmissions
  - High RTT estimate: poor throughput
- RTT estimator must adapt to change in RTT
  - But not too fast, or too slow!
- So how do we estimate RTT?

Original TCP Round-trip Estimator

- Round trip times exponentially averaged:
  - New RTT = α (old RTT) + (1 - α) (new sample)
  - Recommended value for α: 0.8 - 0.9
    - 0.875 for most TCP's



- Retransmit timer set to (b * RTT), where b = 2
  - Every time timer expires, RTO exponentially backed-off
- Not good at preventing spurious timeouts
  - Why?

Jacobson's Retransmission Timeout

- Key observation:
  - At high loads, round trip variance is high
- Solution:
  - Base RTO on RTT and standard deviation
    - RTO = RTT + 4 * rttvar
  - new_rttvar = β * dev + (1- β) old_rttvar
    - Dev = linear deviation
    - Inappropriately named – actually smoothed linear deviation
- In practice: TOs use coarse clock, e.g., 100s of msec

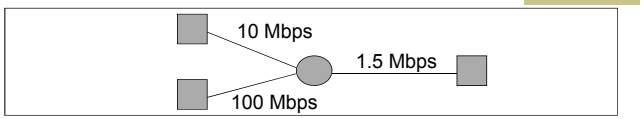
Important Lessons

- Transport service
  - UDP → mostly just IP service + demultiplexing
  - TCP → congestion controlled, reliable, byte stream
- Types of ARQ protocols
  - Sliding window for high throughput
  - Go-back-n → can keep link utilized (except w/ losses)
  - Selective repeat → efficient loss recovery
- TCP uses go-back-n variant
  - Avoid unnecessary retransmission ..
  - … and gaps in the flow (fast retransmit/recovery, SACK)

Outline

- Transport introduction
- TCP connection establishment
- Error recovery and flow control
- Making things work in TCP
- Congestion control
  - Fundamentals
  - TCP congestion control
- Transport optimization and futures

Congestion



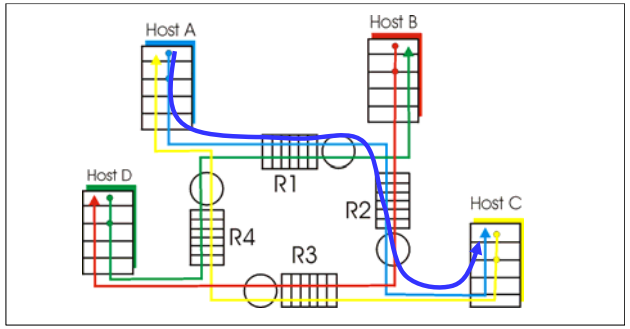
- Many sources "share*" resources inside network
- Problem: demand can exceed capacity of the network
  - Sources are unaware of current state of resource
  - Sources are unaware of each other
- Manifestations:
  - Lost packets (buffer overflow at routers)
  - Long delays (queuing in router buffers)
- Challenge:
  How do we coordinate all nodes in the Internet?

* Share → Compete for?

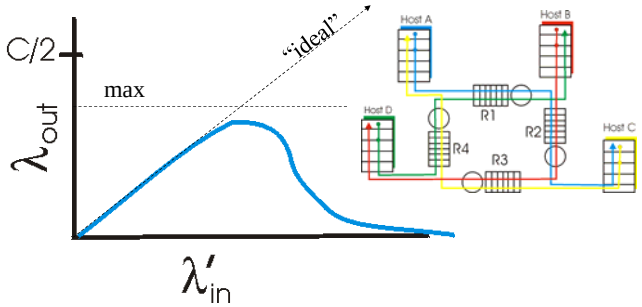
Causes & Costs of Congestion

- Four senders – multihop paths
- Timeout/retransmit

Q: What happens as rate increases?



Causes & Costs of Congestion



- When packet dropped, any "upstream transmission capacity used for that packet was wasted!

Congestion Collapse

- Definition: *Increase in network load results in decrease of useful work done*
- Many possible causes
  - Spurious retransmissions of packets still in flight
    - How can this happen with packet conservation
    - Solution: better timers and TCP congestion control
  - Undelivered packets
    - Packets consume resources and are dropped elsewhere in network
    - Solution: congestion control for ALL traffic

Plan for Today

- So far we considered two networks
  - Network 1: 1 router, 3 links
  - Network 2: 4 routers, 8 links
- Next step: how do we deal with congestion in the Internet
  - Millions of routers
  - Even more links
  - 100s of millions of senders

Congestion Control Goals

- A mechanism that:
- Uses network resources efficiently:

  High $X = \Sigma x_i(t)$
- Prevents collapse
  - Congestion collapse is not just a theory
  - Has been frequently observed in many networks
- Preserves fair network resource allocation

  For example: $(\Sigma x_i)^2/n(\Sigma x_i^2)$

Two Approaches Towards Congestion Control

End-to-end congestion control:

- No explicit feedback from network
- End-systems infer congestion status from observed loss, delay, …
- Approach taken by TCP
- Problem: making it work
  - Avoid significant packet loss
  - Maintain high utilization

Network-assisted congestion control:

- Routers provide feedback to end systems
  - Single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - Explicit rate sender should send at (ATM)
- Problem: makes routers more complicated
  - Per-flow state → poor scalability
  - Can sometimes be avoided

Congestion Control with Binary Feedback (TCP)

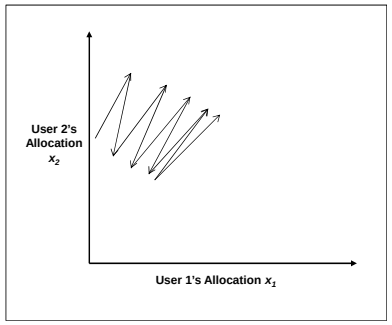
- Very simple mechanisms in network
  - FIFO scheduling with shared buffer pool
  - Feedback through packet drops (or binary feedback)
- TCP interprets packet drops as signs of congestion and sender slows down
  - This is an assumption: packet drops are not a sign of congestion in all networks, e.g., wireless networks
- Sender periodically probes the network to check whether more bandwidth has become available
- Key questions: how much to reduce (after a drop) and increase (when probing) rate

Linear Control

- Many different possibilities for reaction to congestion and probing for bandwidth
  - Examine simple linear controls
    - Window(t + 1) = a + b Window(t)
    - Different a_i/b_i for increase and a_d/b_d for decrease
- Supports various reaction to signals
  - Increase/decrease additively
  - Increased/decrease multiplicatively
  - Which of the four combinations is optimal?
- Example of closed loop control: system must converge!
  - In addition to efficiency, fairness, …. goals

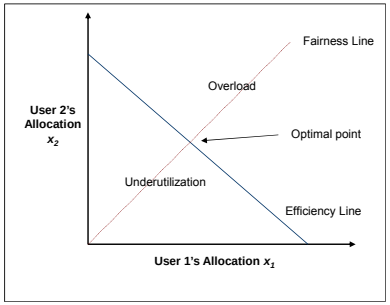
Phase Plots

- Simple way to visualize behavior of competing connections over time
- Sequence of steps with 2 synchronized senders



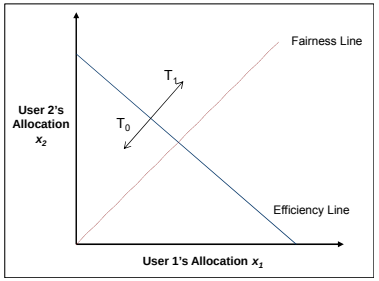
Phase Plots

- What are desirable properties?
- What if flows are not equal?



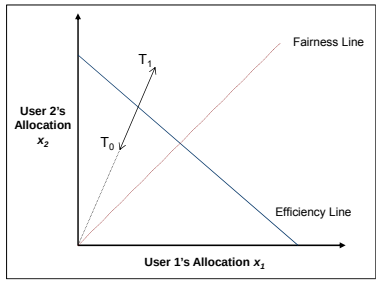
Additive Increase/Decrease

- Both X_1 and X_2 increase/ decrease by the same amount over time
- Additive increase improves fairness and additive decrease reduces fairness

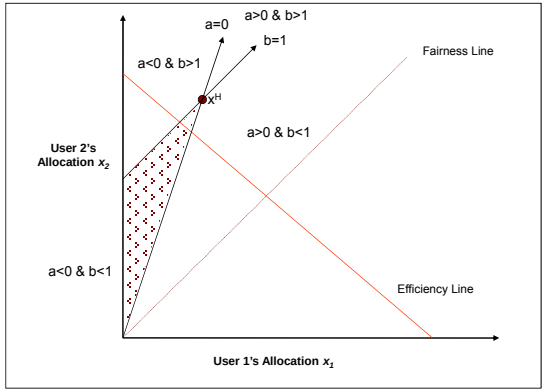


Muliplicative Increase/Decrease

- Both X_1 and X_2 increase by the same factor over time
  - Extension along line through origin
- Constant fairness

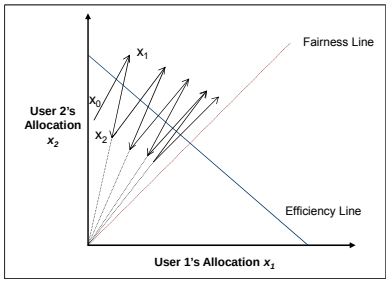


Achieving Fairness AND Efficiency



What is the Right Choice?

- Constraints limit us to AIMD
  - Can have multiplicative term in increase (MAIMD)
  - AIMD moves towards optimal point

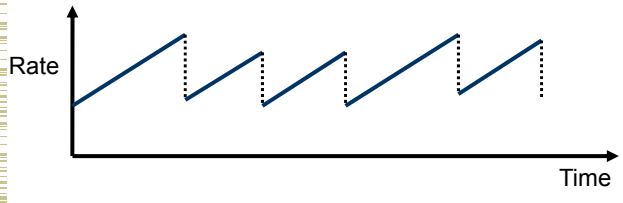


Outline

- Transport introduction
- TCP connection establishment
- Error recovery and flow control
- Making things work in TCP
- TCP congestion control
  - AIMD, packet packing
  - Fast recovery, slow start
- Transport optimization and futures

TCP Congestion Control: Implicit Feedback and AIMD

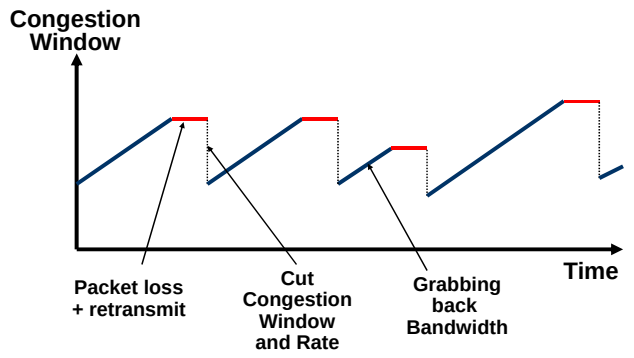
- Distributed, fair and efficient
- Packet loss is seen as sign of congestion and results in a multiplicative rate decrease: factor of 2
- TCP periodically probes for available bandwidth by increasing its rate: by one packet per RTT



Implementation Issue: How to Implement AIMD efficiently

- Operating systems have coarse grain timers – how do control the transmit rate?
  - 100 Mbs → 1500 Byte packet every ~120 μsec
- Solution: uses a congestion window to implement AIMD
  - This is the same strategy that is used for flow control
  - Rate = window / RTT, with RTT more or less constant
- If loss occurs, cut congestion window W in half
  - Set cwnd to 0.5W (multiplicative decrease)
- Upon receiving ACK, increase cwnd by (1 packet)/cwnd
  - What is 1 packet? → 1 MSS worth of bytes
  - After cwnd packets have passed by → increased cwnd by 1 MSS
  - Corresponds to an increase of 1 MSS every roundtrip time

Congestion Avoidance Behavior

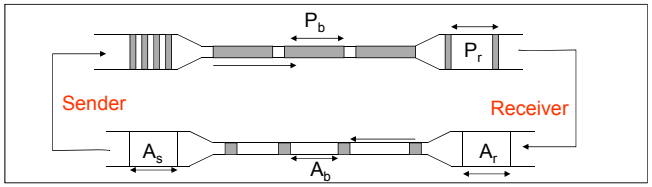


Implementation Issue: Putting the Pieces Together

- Both congestion and flow control want to control when packets can be transmitted – who is really in charge?
- Solution: using a single window to control transmission
  - Sender's maximum window = Min (advertised window, cwnd)
  - In English: can send packets if it does not flood receiver AND it does not congest the network
- The two windows are updated independently
  - Both windows are decreased when a packet is send
  - Advertised window: increased when the receiver sends window update, meaning it freed up a buffer
  - Cwnd: increased when the receiver ACKs the reception of data, meaning data left the network
  - Either event can trigger a send

Implementation Issue: How to Send Packets Smoothly

- Networks do not like very bursty traffic
  - Leads to queue overflow and increases packet loss
- Solution: congestion window helps to "pace" the transmission of data packets – "packet pacing"
- In steady state, a packet is sent when an ack is received
  - Self-clocking behavior: flow remains smooth, once it is smooth



Congestion Avoidance Sequence Plot Pacing and Additive Increase