15-744 Computer Networking

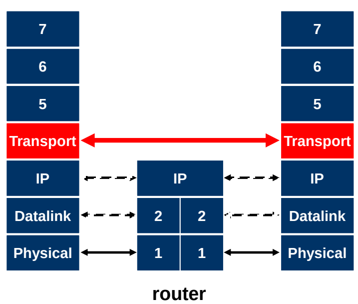
Review 2 – Transport Protocols

Outline

- Transport introduction
- Error recovery & flow control
- TCP flow control/connection setup/data transfer
- TCP reliability
- Congestion sources and collapse
- Congestion control basics

Transport Protocols

- Lowest level end-to-end protocol.
  - Header generated by sender is interpreted only by the destination
  - Routers view transport header as part of the payload
  - Not always true…
    - Firewalls



Functionality Split

- Network provides best-effort delivery
- End-systems implement many functions
  - Reliability
  - In-order delivery
  - Demultiplexing
  - Message boundaries
  - Connection abstraction
  - Congestion control
  - …

Transport Protocols

- UDP provides just integrity and demux
- TCP adds…
  - Connection-oriented
  - Reliable
  - Ordered
  - Byte-stream
  - Full duplex
  - Flow and congestion controlled
- DCCP, RTP, SCTP -- not widely used.

UDP: User Datagram Protocol [RFC 768]

- “No frills,” “bare bones” Internet transport protocol
- “Best effort” service, UDP segments may be:
  - Lost
  - Delivered out of order to app
- *Connectionless:*
  - No handshaking between UDP sender, receiver
  - Each UDP segment handled independently of others

Why is there a UDP?

- No connection establishment (which can add delay)
- Simple: no connection state at sender, receiver
- Small header
- No congestion control: UDP can blast away as fast as desired

UDP, cont.

- Often used for streaming multimedia apps
  - Loss tolerant
  - Rate sensitive
- Other UDP uses (why?):
  - DNS
- Reliable transfer over UDP
  - Must be at application layer
  - Application-specific error recovery

Length, in bytes of UDP segment, including header

32 bits

Source port #	Dest port #
Length	Checksum
Application data (message)	

UDP segment format

UDP Checksum

Goal: detect “errors” (e.g., flipped bits) in transmitted segment – optional use!

Sender:

- Treat segment contents as sequence of 16-bit integers
- Checksum: addition (1’s complement sum) of segment contents
- Sender puts checksum value into UDP checksum field

Receiver:

- Compute checksum of received segment
- Check if computed checksum equals checksum field value:
  - NO - error detected
  - YES - no error detected

  But maybe errors nonetheless?

High-Level TCP Characteristics

- Protocol implemented entirely at the ends
  - Fate sharing (on IP)
- Protocol has evolved over time and will continue to do so
  - Nearly impossible to change the header
  - Use options to add information to the header
  - Change processing at endpoints
  - Backward compatibility is what makes it TCP

9

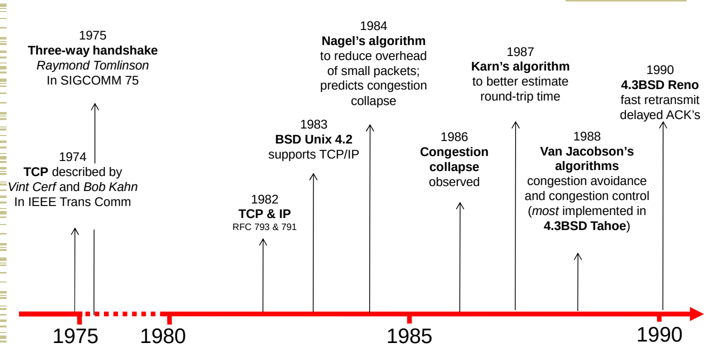
TCP Header

Flags: SYN
FIN
RESET
PUSH
URG
ACK

Source port		Destination port	
Sequence number			
Acknowledgement			
HdrLen	0	Flags	Advertised window
Checksum		Urgent pointer	
Options (variable)			
Data			

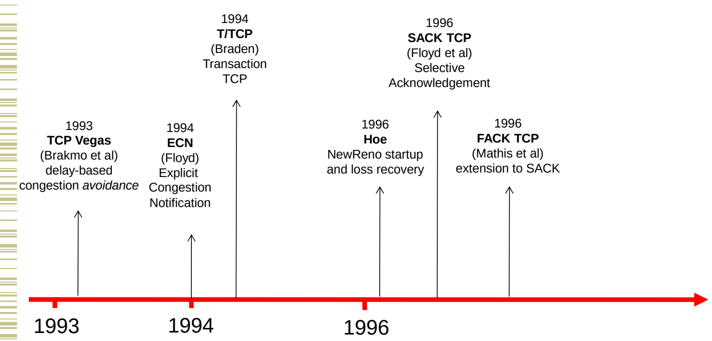
10

Evolution of TCP



11

TCP Through the 1990s



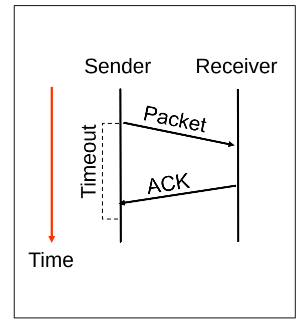
12

Outline

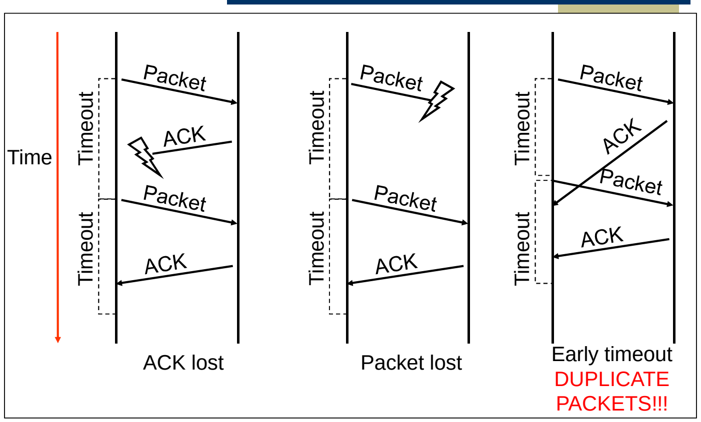
- Transport introduction
- Error recovery & flow control
- TCP flow control/connection setup/data transfer
- TCP reliability
- Congestion sources and collapse
- Congestion control basics

Stop and Wait

- ARQ
  - Receiver sends acknowledgement (ACK) when it receives packet
  - Sender waits for ACK and timeouts if it does not arrive within some time period
- Simplest ARQ protocol
- Send a packet, stop and wait until ACK arrives
- Performance
  - Can only send one packet per round trip

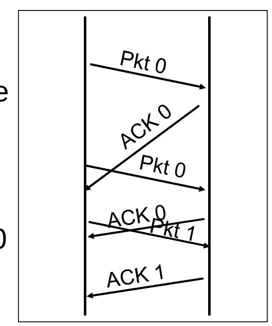


Recovering from Error



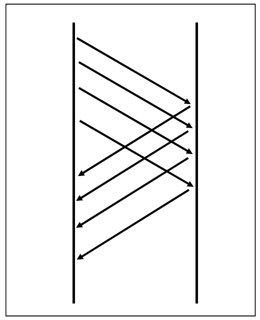
How to Recognize Resends?

- Use sequence numbers
  - both packets and acks
- Sequence # in packet is finite → How big should it be?
  - For stop and wait?
- One bit – won't send seq #1 until received ACK for seq #0



How to Keep the Pipe Full?

- Send multiple packets without waiting for first to be acked
  - Number of pkts in flight = window: Flow control
- Reliable, unordered delivery
  - Several parallel stop & waits
  - Send new packet after each ack
  - Sender keeps list of unack'ed packets; resends after timeout
  - Receiver same as stop & wait
- How large a window is needed?
  - Suppose 10Mbps link, 4ms delay, 500byte pkts
    - 1? 10? 20?
  - Round trip delay * bandwidth = capacity of pipe



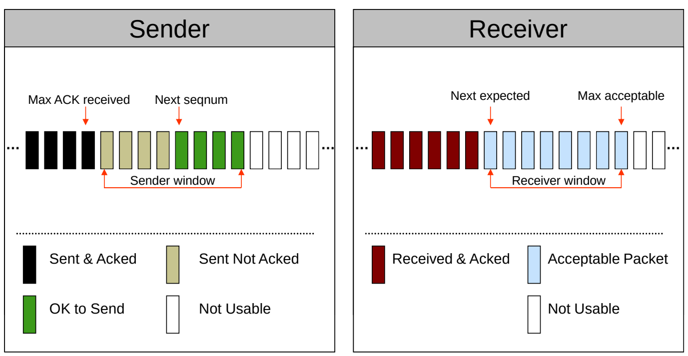
17

Sliding Window

- Reliable, ordered delivery
- Receiver has to hold onto a packet until all prior packets have arrived
  - Why might this be difficult for just parallel stop & wait?
  - Sender must prevent buffer overflow at receiver
- Circular buffer at sender and receiver
  - Packets in transit ≤ buffer size
  - Advance when sender and receiver agree packets at beginning have been received

18

Sender/Receiver State



19

Sequence Numbers

- How large do sequence numbers need to be?
  - Must be able to detect wrap-around
  - Depends on sender/receiver window size
- E.g.
  - Max seq = 7, send win=recv win=7
  - If pkts 0..6 are sent succesfully and all acks lost
    - Receiver expects 7,0..5, sender retransmits old 0..6!!!
- Max sequence must be ≥ send window + recv window

20

Window Sliding – Common Case

- On reception of new ACK (i.e. ACK for something that was not acked earlier)
  - Increase sequence of max ACK received
  - Send next packet
- On reception of new in-order data packet (next expected)
  - Hand packet to application
  - Send cumulative ACK – acknowledges reception of all packets up to sequence number
  - Increase sequence of max acceptable packet

21

Loss Recovery

- On reception of out-of-order packet
  - Send nothing (wait for source to timeout)
  - Cumulative ACK (helps source identify loss)
- Timeout (Go-Back-N recovery)
  - Set timer upon transmission of packet
  - Retransmit all unacknowledged packets
- Performance during loss recovery
  - No longer have an entire window in transit
  - Can have much more clever loss recovery

22

Important Lessons

- Transport service
  - UDP → mostly just IP service
  - TCP → congestion controlled, reliable, byte stream
- Types of ARQ protocols
  - Stop-and-wait → slow, simple
  - Go-back-n → can keep link utilized (except w/ losses)
  - Selective repeat → efficient loss recovery -- used in SACK
- Sliding window flow control
  - Addresses buffering issues and keeps link utilized

23

Good Ideas So Far…

- Flow control
  - Stop & wait
  - Parallel stop & wait
  - Sliding window
- Loss recovery
  - Timeouts
  - Acknowledgement-driven recovery (selective repeat or cumulative acknowledgement)

24

Outline

- Transport introduction
- Error recovery & flow control
- TCP flow control/connection setup/data transfer
- TCP reliability
- Congestion sources and collapse
- Congestion control basics

25

More on Sequence Numbers

- 32 Bits, Unsigned → for bytes not packets!
- Why So Big?
  - For sliding window, must have
  - |Sequence Space| > |Sending Window| + |Receiving Window|
    - No problem
  - Also, want to guard against stray packets
    - With IP, packets have maximum lifetime of 120s
    - Sequence number would wrap around in this time at 286Mbps

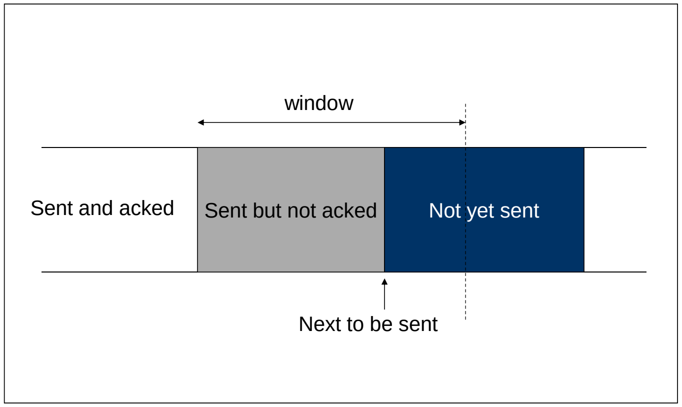
26

TCP Flow Control

- TCP is a sliding window protocol
  - For window size *n*, can send up to *n* bytes without receiving an acknowledgement
  - When the data is acknowledged then the window slides forward
- Each packet advertises a window size
  - Indicates number of bytes the receiver has space for
- Original TCP always sent entire window
  - Congestion control now limits this

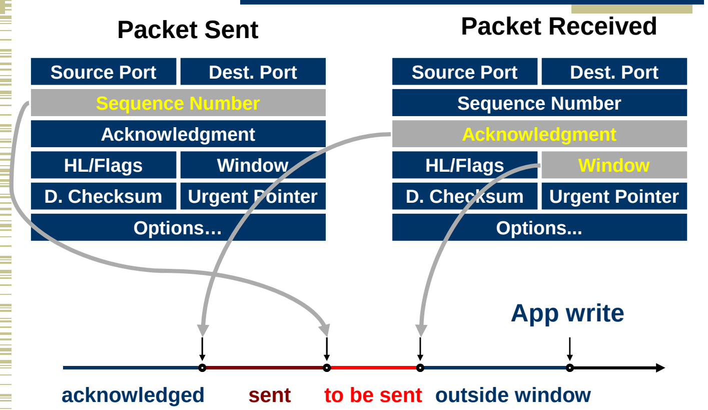
27

Window Flow Control: Send Side



28

Window Flow Control: Send Side

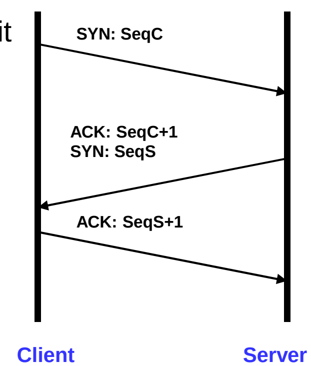


Performance Considerations

- The window size can be controlled by receiving application
  - Can change the socket buffer size from a default (e.g. 8Kbytes) to a maximum value (e.g. 64 Kbytes)
- The window size field in the TCP header limits the window that the receiver can advertise
  - 16 bits → 64 KBytes
  - 10 msec RTT → 51 Mbit/second
  - 100 msec RTT → 5 Mbit/second
  - TCP options to get around 64KB limit → scales window size

Establishing Connection: Three-Way handshake

- Each side notifies other of starting sequence number it will use for sending
  - Why not simply chose 0?
    - Must avoid overlap with earlier incarnation
    - Security issues
- Each side acknowledges other's sequence number
  - SYN-ACK: Acknowledge sequence number + 1
- Can combine second SYN with first ACK



Outline

- Transport introduction
- Error recovery & flow control
- TCP flow control/connection setup/data transfer
- TCP reliability
- Congestion sources and collapse
- Congestion control basics

Reliability Challenges

- Congestion related losses
- Variable packet delays
  - What should the timeout be?
- Reordering of packets
  - How to tell the difference between a delayed packet and a lost one?

TCP = Go-Back-N Variant

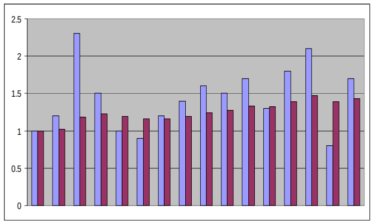
- Sliding window with cumulative acks
  - Receiver can only return a single "ack" sequence number to the sender.
  - Acknowledges all bytes with a lower sequence number
  - Starting point for retransmission
  - Duplicate acks sent when out-of-order packet received
- But: sender only retransmits a single packet.
  - Reason???
    - Only one that it knows is lost
    - Network is congested → shouldn't overload it
- Error control is based on byte sequences, not packets.
  - Retransmitted packet can be different from the original lost packet – Why?

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!
- Spurious timeouts
  - "Conservation of packets" principle – never more than a window worth of packets in flight

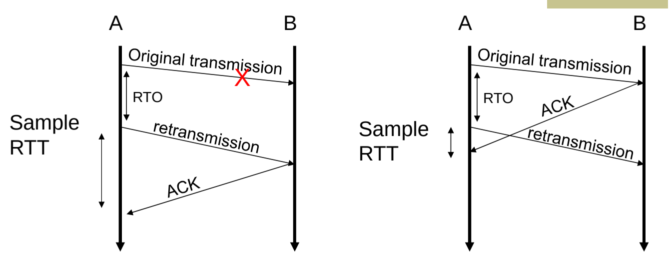
Original TCP Round-trip Estimator

- 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 premature timeouts

RTT Sample Ambiguity



- Karn's RTT Estimator
  - If a segment has been retransmitted:
    - Don't count RTT sample on ACKs for this segment
    - Keep backed off time-out for next packet
    - Reuse RTT estimate only after one successful transmission

37

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

38

Timestamp Extension

- Used to improve timeout mechanism by more accurate measurement of RTT
- When sending a packet, insert current time into option
  - 4 bytes for time, 4 bytes for echo a received timestamp
- Receiver echoes timestamp in ACK
  - Actually will echo whatever is in timestamp
- Removes retransmission ambiguity
  - Can get RTT sample on any packet

39

Timer Granularity

- Many TCP implementations set RTO in multiples of 200,500,1000ms
- Why?
  - Avoid spurious timeouts – RTTs can vary quickly due to cross traffic
  - Reduce timer expensive timer interrupts on hosts
- What happens for the first couple of packets?
  - Pick a very conservative value (seconds)

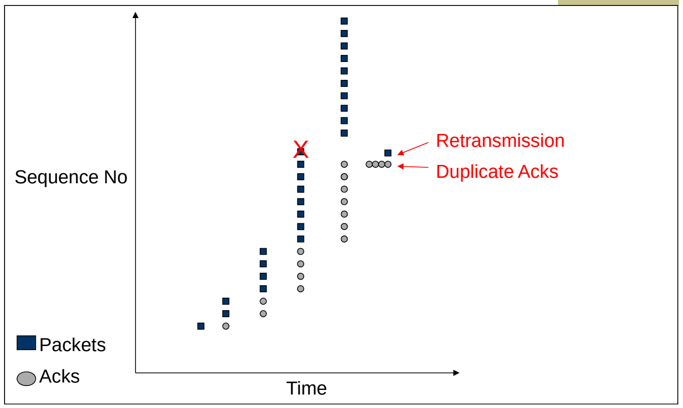
40

Fast Retransmit -- Avoiding Timeouts

- What are duplicate acks (dupacks)?
  - Repeated acks for the same sequence
- When can duplicate acks occur?
  - Loss
  - Packet re-ordering
  - Window update – advertisement of new flow control window
- Assume re-ordering is infrequent and not of large magnitude
  - Use receipt of 3 or more duplicate acks as indication of loss
  - Don't wait for timeout to retransmit packet

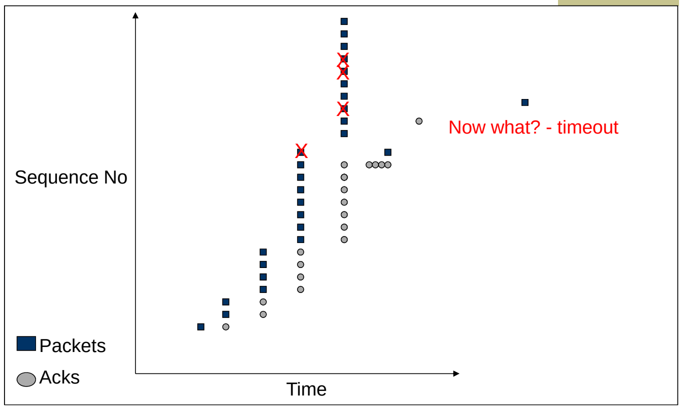
41

Fast Retransmit



42

TCP (Reno variant)



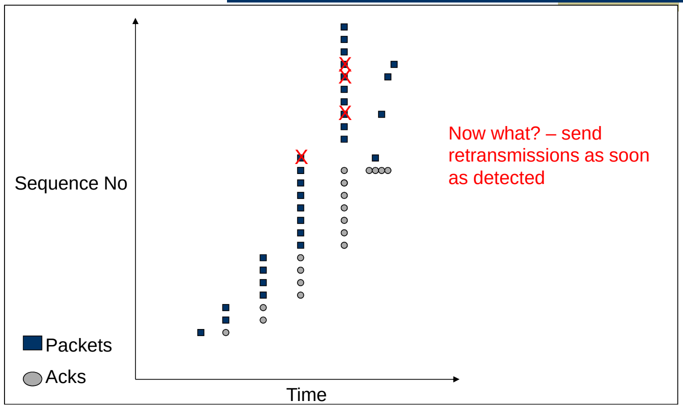
43

SACK

- Basic problem is that cumulative acks provide little information
- 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

44

SACK



Performance Issues

- Timeout >> fast rexmit
- Need 3 dupacks/sacks
- Not great for small transfers
  - Don't have 3 packets outstanding
- What are real loss patterns like?

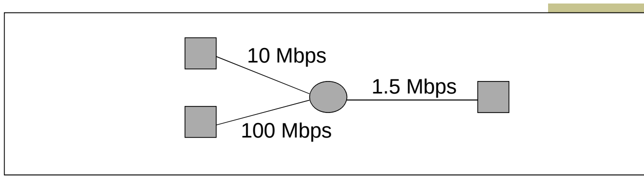
Important Lessons

- Three-way TCP Handshake
- TCP timeout calculation → how is RTT estimated
- Modern TCP loss recovery
  - Why are timeouts bad?
  - How to avoid them? → e.g. fast retransmit

Outline

- Transport introduction
- Error recovery & flow control
- TCP flow control/connection setup/data transfer
- TCP reliability
- Congestion sources and collapse
- Congestion control basics

Congestion

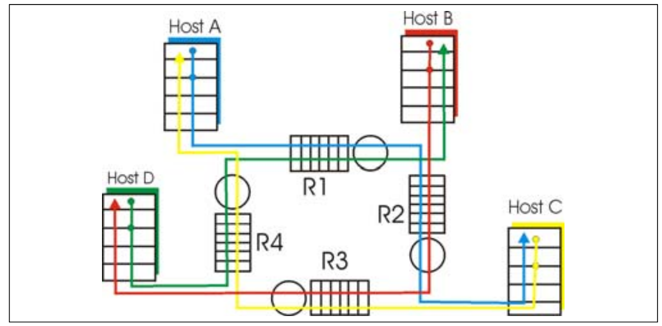


- Different sources compete for resources inside network
- Why is it a problem?
  - Sources are unaware of current state of resource
  - Sources are unaware of each other
  - In many situations will result in < 1.5 Mbps of throughput (congestion collapse)

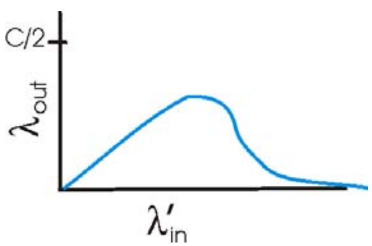
Causes & Costs of Congestion

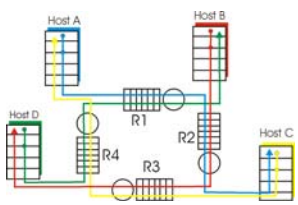
- Four senders – multihop paths
- Timeout/retransmit

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
    - Classical congestion collapse
    - 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
  - Etc..

Other Congestion Collapse Causes

- Fragments
  - Mismatch of transmission and retransmission units
  - Solutions
    - Make network drop all fragments of a packet (early packet discard in ATM)
    - Do path MTU discovery
- Control traffic
  - Large percentage of traffic is for control
    - Headers, routing messages, DNS, etc.
- Stale or unwanted packets
  - Packets that are delayed on long queues
  - "Push" data that is never used

53

Where to Prevent Collapse?

- Can end hosts prevent problem?
  - Yes, but must trust end hosts to do right thing
  - E.g., sending host must adjust amount of data it puts in the network based on detected congestion
- Can routers prevent collapse?
  - No, not all forms of collapse
  - Doesn't mean they can't help
    - Sending accurate congestion signals
    - Isolating well-behaved from ill-behaved sources

54

Congestion Control and Avoidance

- A mechanism which:
  - Uses network resources efficiently
  - Preserves fair network resource allocation
  - Prevents or avoids collapse
- Congestion collapse is not just a theory
  - Has been frequently observed in many networks

55

Approaches For Congestion Control

- Two broad approaches towards congestion control:

End-to-end

- No explicit feedback from network
- Congestion inferred from end-sys   tem observed loss, delay
- Approach taken by TCP

Network-assisted

- Routers provide feedback to end systems
  - Explicit rate sender should send at
  - Single bit indicating congestion (SNA, DEC bit, TCP/IP ECN, ATM)
- Problem: makes routers complicated

56

Example: TCP Congestion Control

- Very simple mechanisms in network
  - FIFO scheduling with shared buffer pool
  - Feedback through packet drops
- TCP interprets packet drops as signs of congestion and slows down
  - This is an assumption: packet drops are not a sign of congestion in all networks
    - E.g. wireless networks
- Periodically probes the network to check whether more bandwidth has become available.

Outline

- Transport introduction
- Error recovery & flow control
- TCP flow control/connection setup/data transfer
- TCP reliability
- Congestion sources and collapse
- Congestion control basics

Objectives

- Simple router behavior
- Distributedness
- Efficiency: $X_{knee} = \Sigma x_i(t)$
- Fairness: $(\Sigma x_i)^2/n(\Sigma x_i^2)$
- Power: $(\text{throughput}^\alpha/\text{delay})$
- Convergence: control system must be stable

Basic Control Model

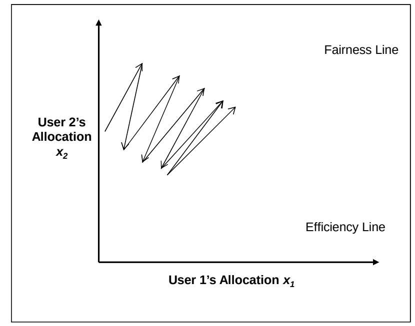
- Let's assume window-based control
- Reduce window when congestion is perceived
  - How is congestion signaled?
    - Either mark or drop packets
  - When is a router congested?
    - Drop tail queues – when queue is full
    - Average queue length – at some threshold
- Increase window otherwise
  - Probe for available bandwidth – how?

Linear Control

- Many different possibilities for reaction to congestion and probing
  - 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?

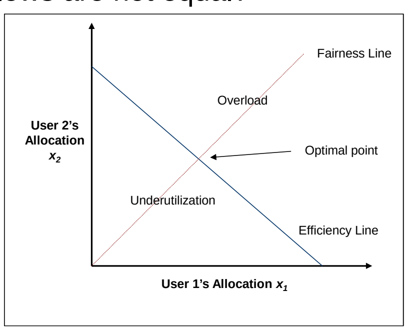
Phase plots

- Simple way to visualize behavior of competing connections over time



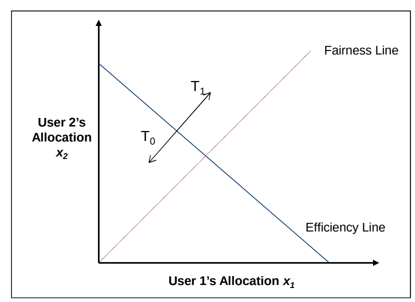
Phase plots

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



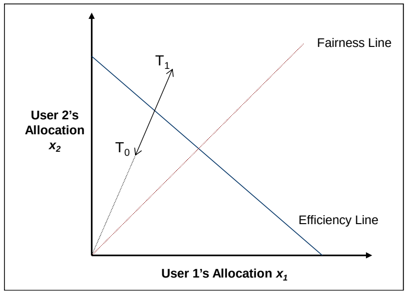
Additive Increase/Decrease

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



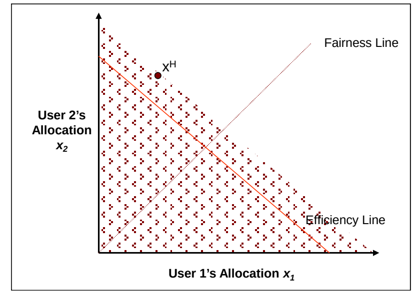
Multiplicative Increase/Decrease

- Both X_1 and X_2 increase by the same factor over time
  - Extension from origin – constant fairness



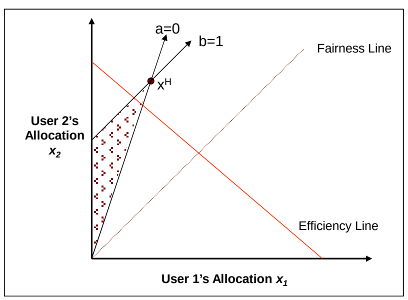
65

Convergence to Efficiency



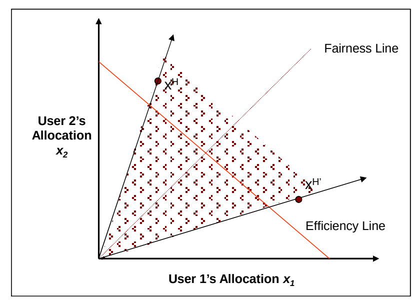
66

Distributed Convergence to Efficiency



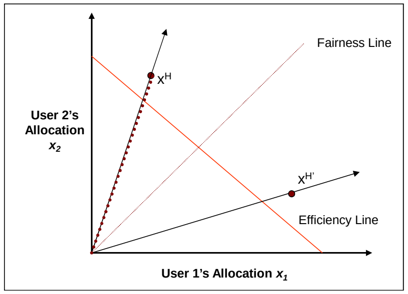
67

Convergence to Fairness

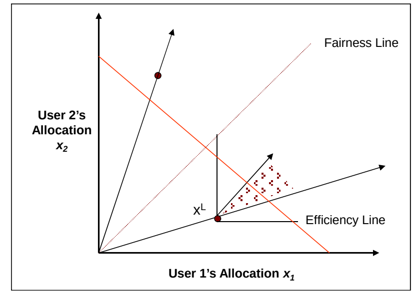


68

Convergence to Efficiency & Fairness



Increase

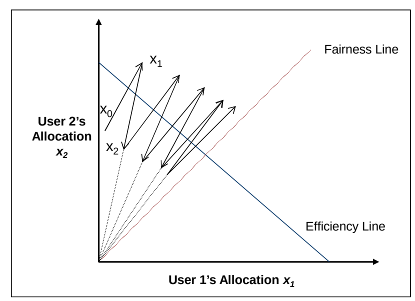


Constraints

- Distributed efficiency
  - I.e., Σ Window(t+1) > Σ Window(t) during increase
    - $a_i > 0$ & $b_i \geq 1$
    - Similarly, $a_d < 0$ & $b_d \leq 1$
- Must never decrease fairness
  - a & b's must be ≥ 0
  - $a_i/b_i > 0$ and $a_d/b_d \geq 0$
- Full constraints
  - $a_d = 0$, $0 \leq b_d < 1$, $a_i > 0$ and $b_i \geq 1$

What is the Right Choice?

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



Questions

- Fairness – why not support skew → AIMD/GAIMD analysis
- More bits of feedback → DECbit, XCP, Vegas
- Guess # of users → hard in async system, look at loss rate?
- Stateless vs. stateful design
- Wired vs. wireless
- Non-linear controls → Bionomial

73

TCP Congestion Control

- Congestion Control
- RED

- Assigned Reading
  - [FJ93] Random Early Detection Gateways for Congestion Avoidance
  - [TFRC] Equation-Based Congestion Control for Unicast Applications (two sections)

74