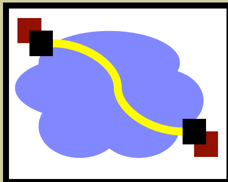
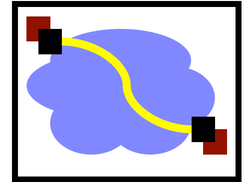


CE693 Advanced Computer Networks

Review 2 – Transport Protocols

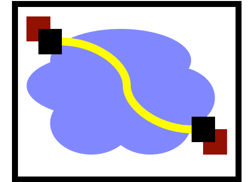
Acknowledgments: Lecture slides are from the graduate level Computer Networks course thought by Srinivasan Seshan at CMU. When slides are obtained from other sources, a a reference will be noted on the bottom of that slide. A full list of references is provided on the last slide.

Outline

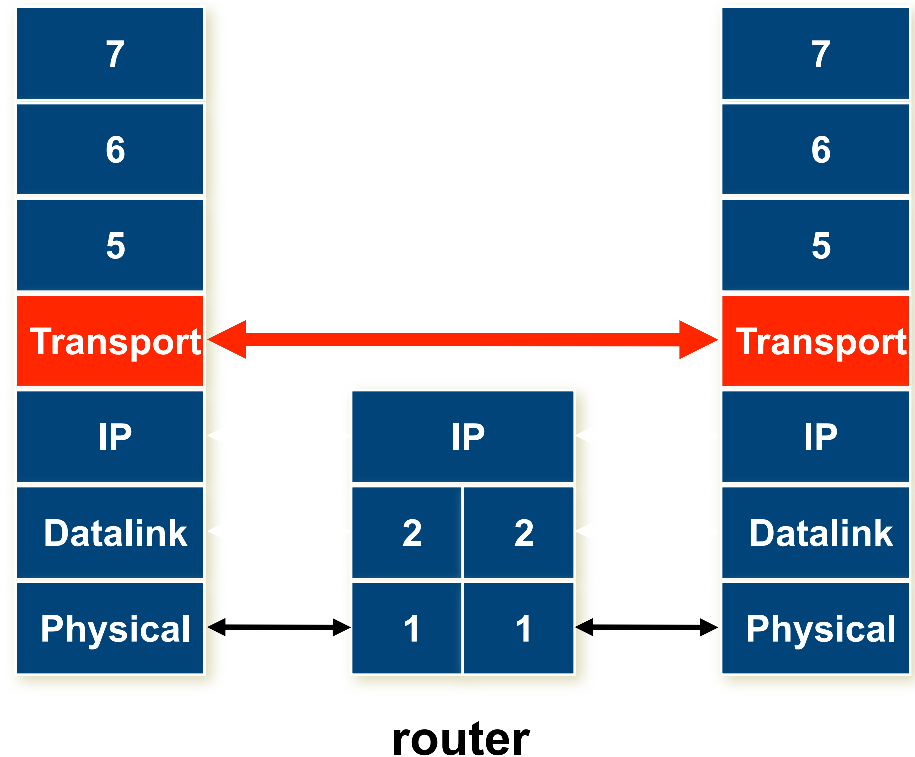


- Transport introduction
- Error recovery & flow control
- TCP flow control/connection setup/data transfer
- TCP reliability
- Congestion sources and collapse
- Congestion control basics

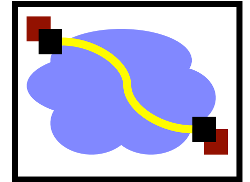
Transport Protocols



- 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

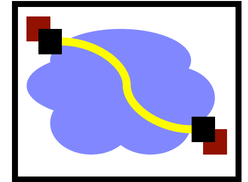


Functionality Split



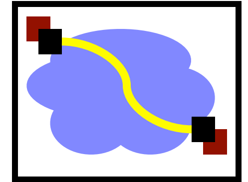
- Network provides best-effort delivery
- End-systems implement many functions
  - Reliability
  - In-order delivery
  - Demultiplexing
  - Message boundaries
  - Connection abstraction
  - Congestion control
  - …

Transport Protocols



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

UDP: User Datagram Protocol [RFC 768]

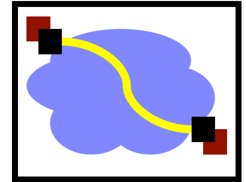


- "No frills," "bare bones" Internet transport protocol
- "Best effort" service, UDP segments may be:
  - Lost
  - Delivered out of order to app
- *Connectionless:*
  - No handshaking between UDP sender, receiver
  - Each UDP segment handled independently of others

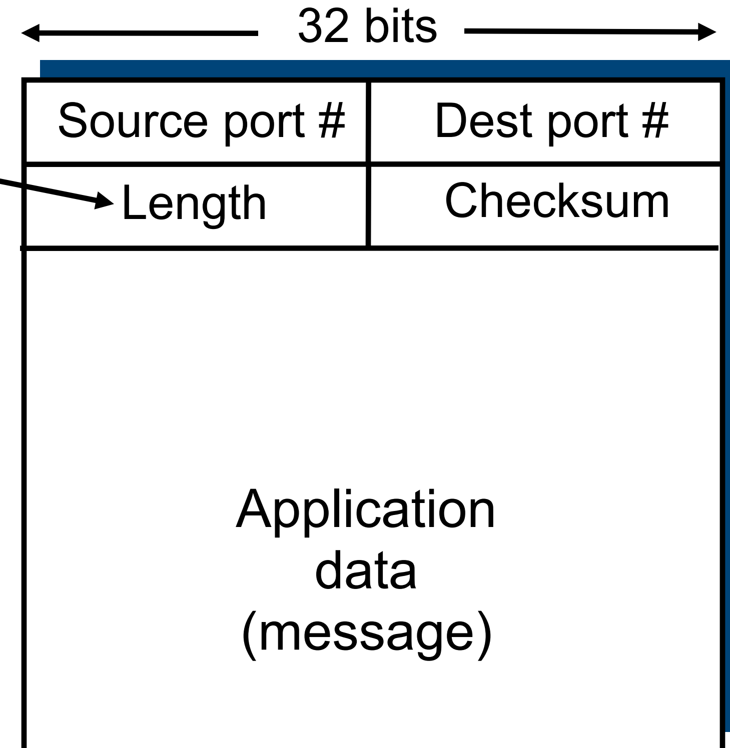
Why is there a UDP?

- 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.

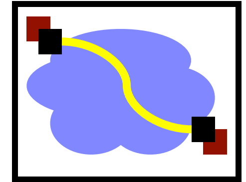


- Often used for streaming multimedia apps
  - Loss tolerant
  - Rate sensitive
- Other UDP uses (why?):
  - DNS
- Reliable transfer over UDP
  - Must be at application layer
  - Application-specific error recovery



UDP segment format

UDP Checksum



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

Sender:

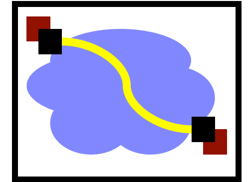
- Treat segment contents as sequence of 16-bit integers
- Checksum: addition (1’s complement sum) of segment contents
- Sender puts checksum value into UDP checksum field

Receiver:

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

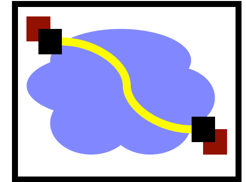
  But maybe errors nonetheless?

High-Level TCP Characteristics

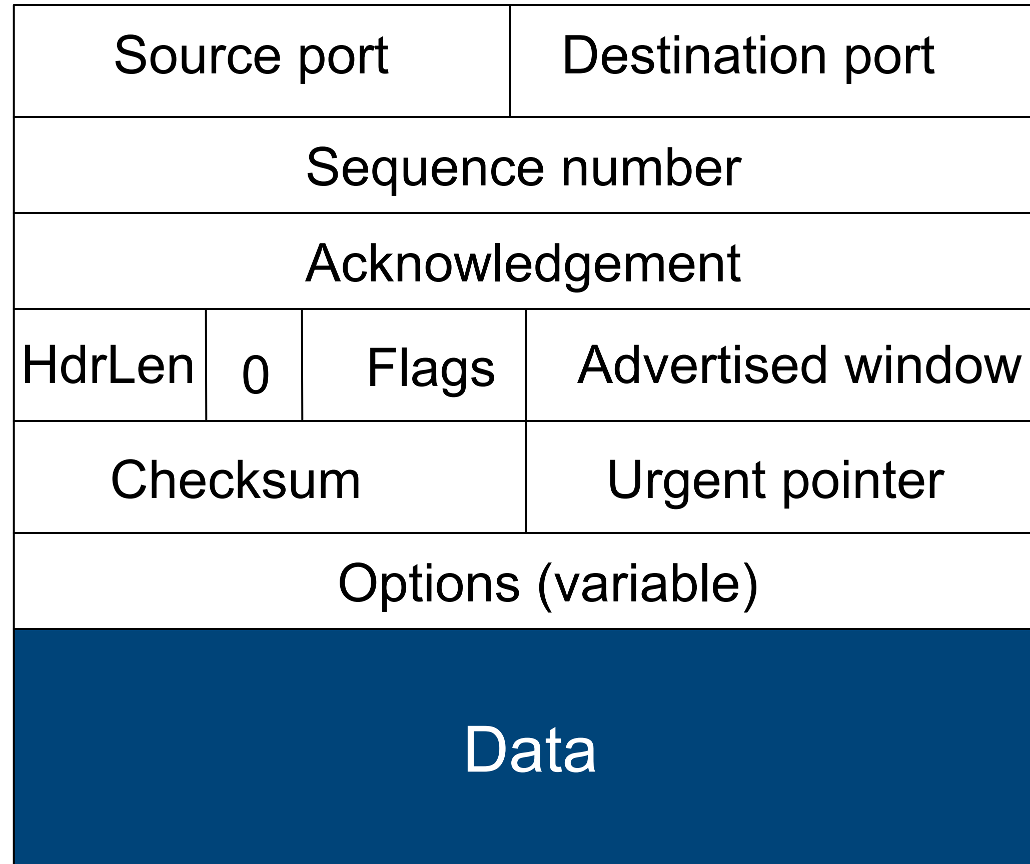


- Protocol implemented entirely at the ends
  - Fate sharing (on IP)
- Protocol has evolved over time and will continue to do so
  - Nearly impossible to change the header
  - Use options to add information to the header
  - Change processing at endpoints
  - Backward compatibility is what makes it TCP

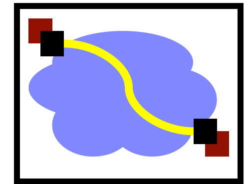
TCP Header



Flags: SYN
FIN
RESET
PUSH
URG
ACK



Evolution of TCP



1975
Three-way handshake
Raymond Tomlinson
In SIGCOMM 75

1974
TCP described by
Vint Cerf and *Bob Kahn*
In IEEE Trans Comm

1982
TCP & IP
RFC 793 & 791

1983
BSD Unix 4.2
supports TCP/IP

1984
Nagel's algorithm
to reduce overhead
of small packets;
predicts congestion
collapse

1986
Congestion
collapse
observed

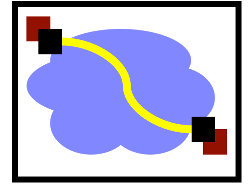
1987
Karn's algorithm
to better estimate
round-trip time

1988
Van Jacobson's
algorithms
congestion avoidance
and congestion control
(*most* implemented in
4.3BSD Tahoe)

1990
4.3BSD Reno
fast retransmit
delayed ACK's

1975 1980 1985 1990

TCP Through the 1990s



1993
TCP Vegas
(Brakmo et al)
delay-based
congestion *avoidance*

1994
ECN
(Floyd)
Explicit
Congestion
Notification

1994
T/TCP
(Braden)
Transaction
TCP

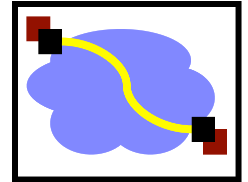
1996
Hoe
NewReno startup
and loss recovery

1996
SACK TCP
(Floyd et al)
Selective
Acknowledgement

1996
FACK TCP
(Mathis et al)
extension to SACK

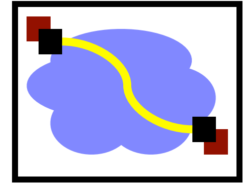
1993 1994 1996

Outline

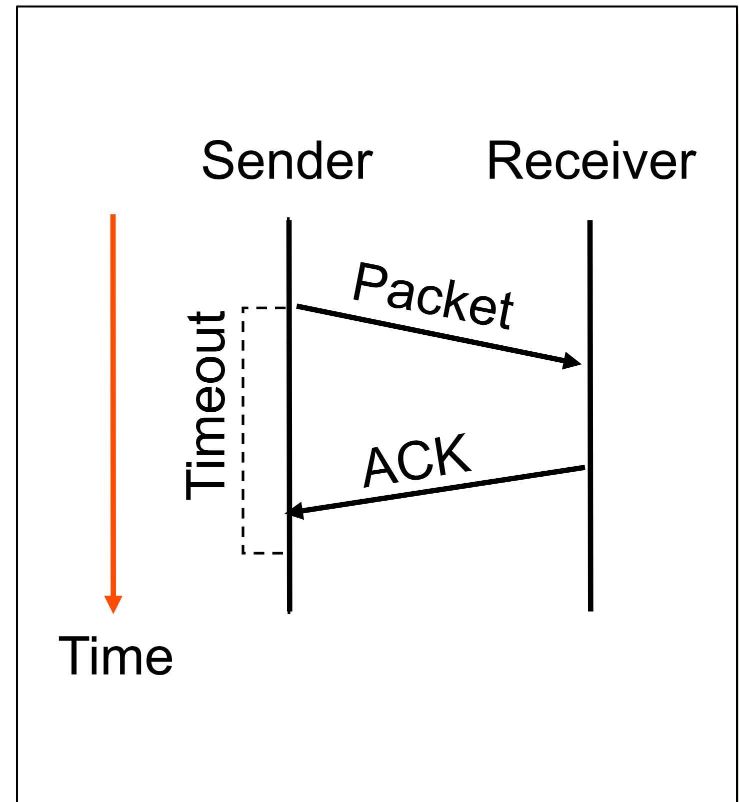


- Transport introduction
- Error recovery & flow control
- TCP flow control/connection setup/data transfer
- TCP reliability
- Congestion sources and collapse
- Congestion control basics

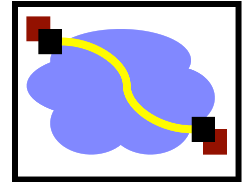
Stop and Wait



- 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



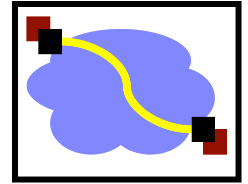
Time

Timeout
Timeout
Packet
ACK
Packet
ACK
ACK lost

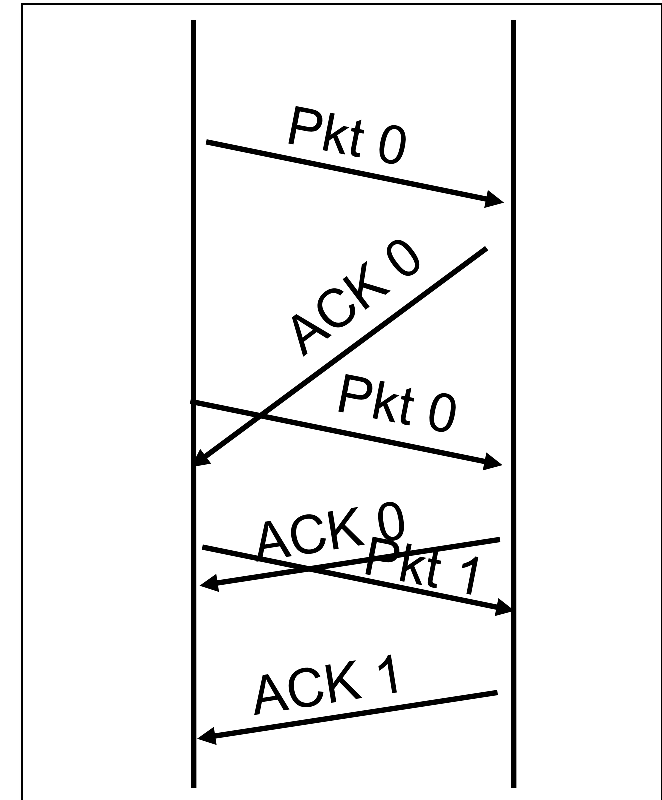
Timeout
Timeout
Packet
Packet
ACK
Packet lost

Timeout
Timeout
Packet
ACK
Packet
ACK
Early timeout
DUPLICATE PACKETS!!!

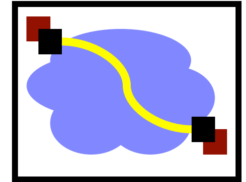
How to Recognize Resends?



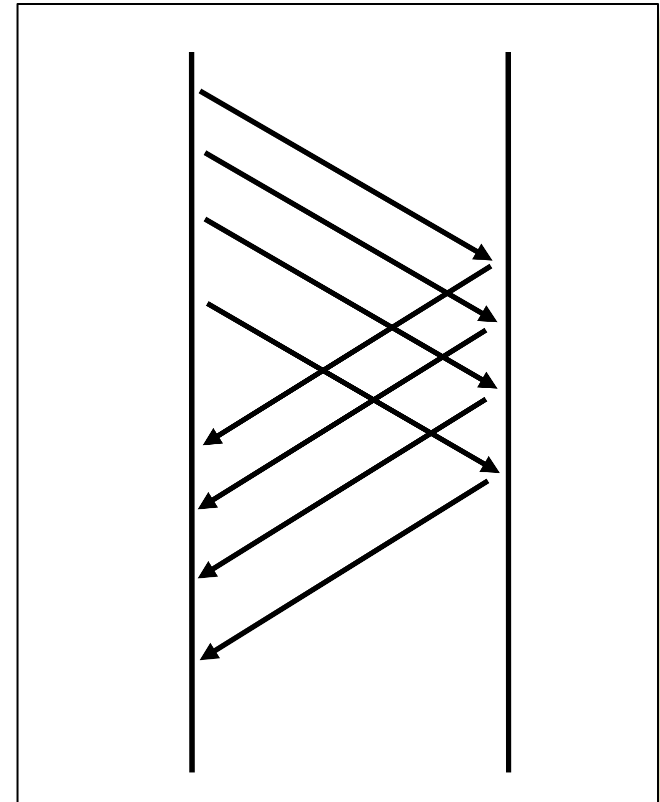
- Use sequence numbers
  - both packets and acks
- Sequence # in packet is finite → How big should it be?
  - For stop and wait?
- One bit – won't send seq #1 until received ACK for seq #0



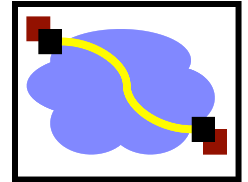
How to Keep the Pipe Full?



- 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?
  - delay * bandwidth = capacity of pipe

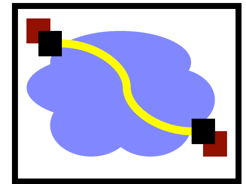


Sliding Window

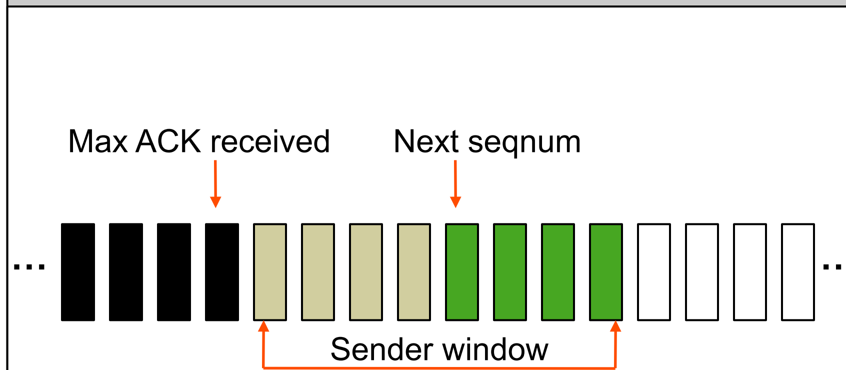


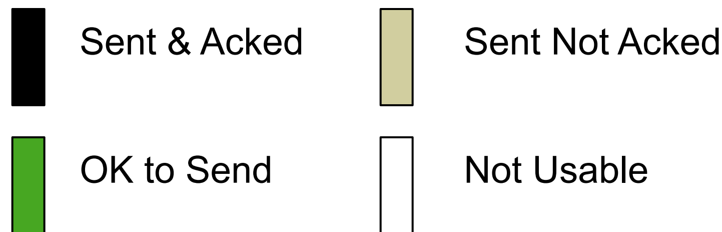
- Reliable, ordered delivery
- Receiver has to hold onto a packet until all prior packets have arrived
  - Why might this be difficult for just parallel stop & wait?
  - Sender must prevent buffer overflow at receiver
- Circular buffer at sender and receiver
  - Packets in transit ≤ buffer size
  - Advance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

Sender/Receiver State

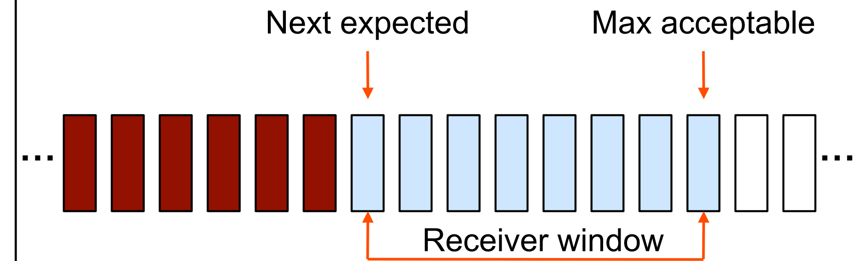


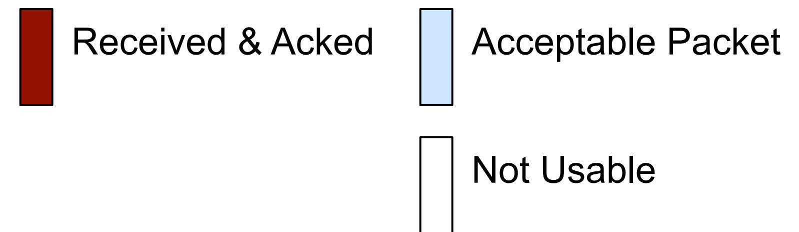
Sender



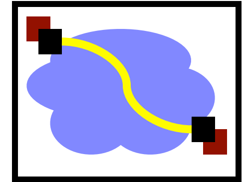


Receiver



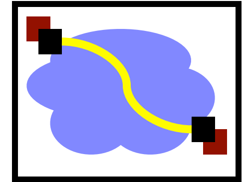


Sequence Numbers



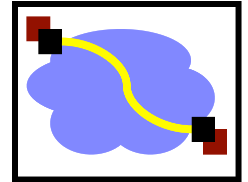
- How large do sequence numbers need to be?
  - Must be able to detect wrap-around
  - Depends on sender/receiver window size
- E.g.
  - Max seq = 7, send win=recv win=7
  - If pkts 0..6 are sent succesfully and all acks lost
    - Receiver expects 7,0..5, sender retransmits old 0..6!!!
- Max sequence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

Window Sliding – Common Case



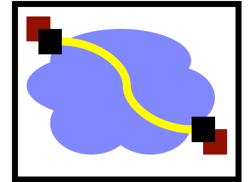
- On reception of new ACK (i.e. ACK for something that was not acked earlier)
  - Increase sequence of max ACK received
  - Send next packet
- On reception of new in-order data packet (next expected)
  - Hand packet to application
  - Send cumulative ACK – acknowledges reception of all packets up to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

Loss Recovery



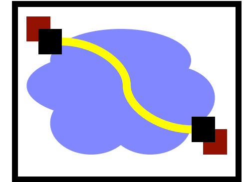
- On reception of out-of-order packet
  - Send nothing (wait for source to timeout)
  - Cumulative ACK (helps source identify loss)
- Timeout (Go-Back-N recovery)
  - Set timer upon transmission of packet
  - Retransmit all unacknowledged packets
- Performance during loss recovery
  - No longer have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

Important Lessons



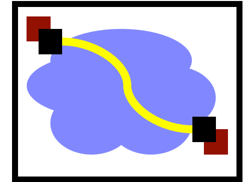
- Transport service
  - UDP → mostly just IP service
  - TCP → congestion controlled, reliable, byte stream
- Types of ARQ protocols
  - Stop-and-wait → slow, simple
  - Go-back-n → can keep link utilized (except w/ losses)
  - Selective repeat → efficient loss recovery -- used in SACK
- Sliding window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

Good Ideas So Far…



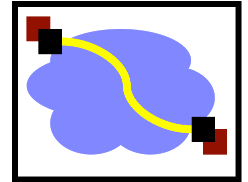
- Flow control
  - Stop & wait
  - Parallel stop & wait
  - Sliding window
- Loss recovery
  - Timeouts
  - Acknowledgement-driven recovery (selective repeat or cumulative acknowledgement)

Outline



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

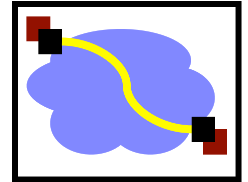
More on Sequence Numbers



- 32 Bits, Unsigned → for bytes not packets!

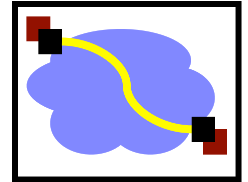
- Why So Big?
  - For sliding window, must have
  - |Sequence Space| > |Sending Window| + |Receiving Window|
    - No problem
  - Also, want to guard against stray packets
    - With IP, packets have maximum lifetime of 120s
    - Sequence number would wrap around in this time at 286Mbps

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

Window Flow Control: Send Side



window

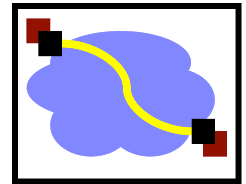
Sent and acked

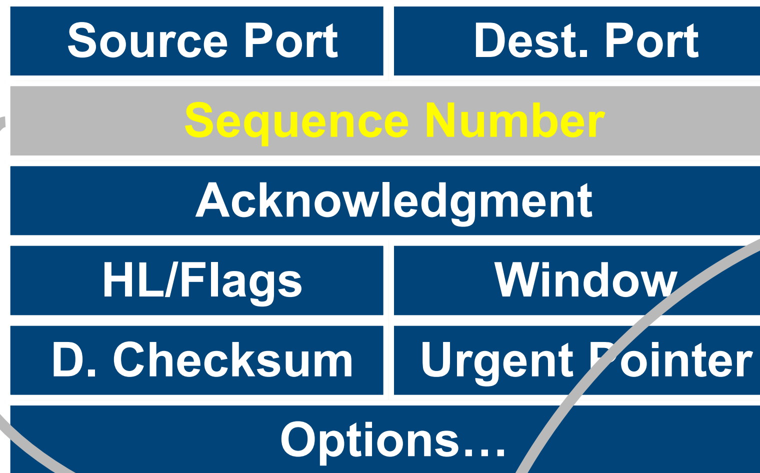
Sent but not acked

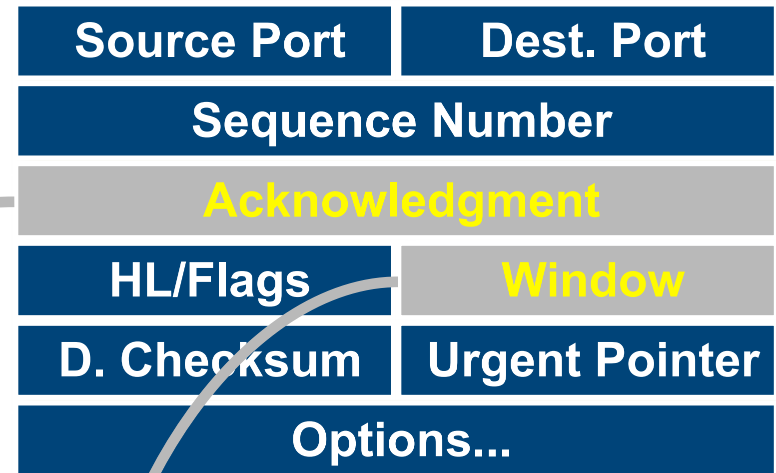
Not yet sent

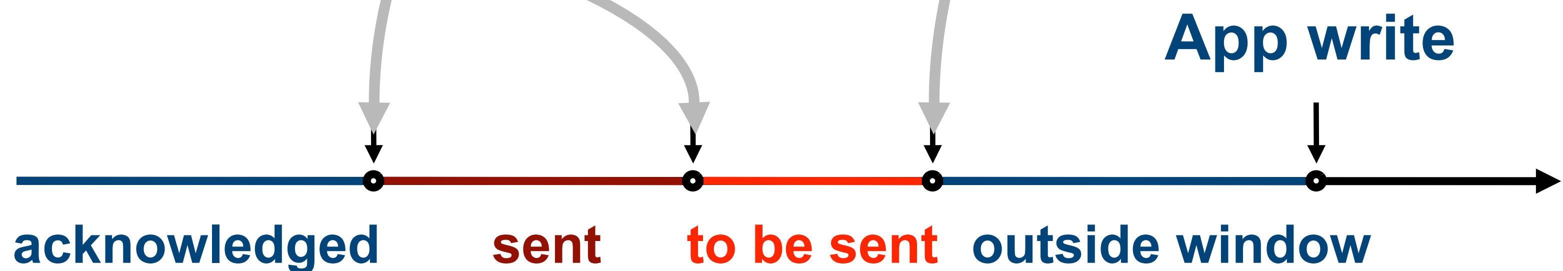
Next to be sent

Window Flow Control: Send Side

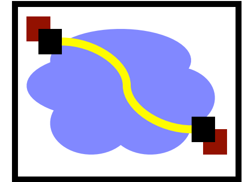






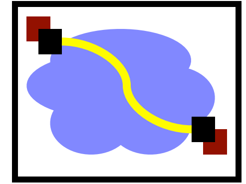


Performance Considerations

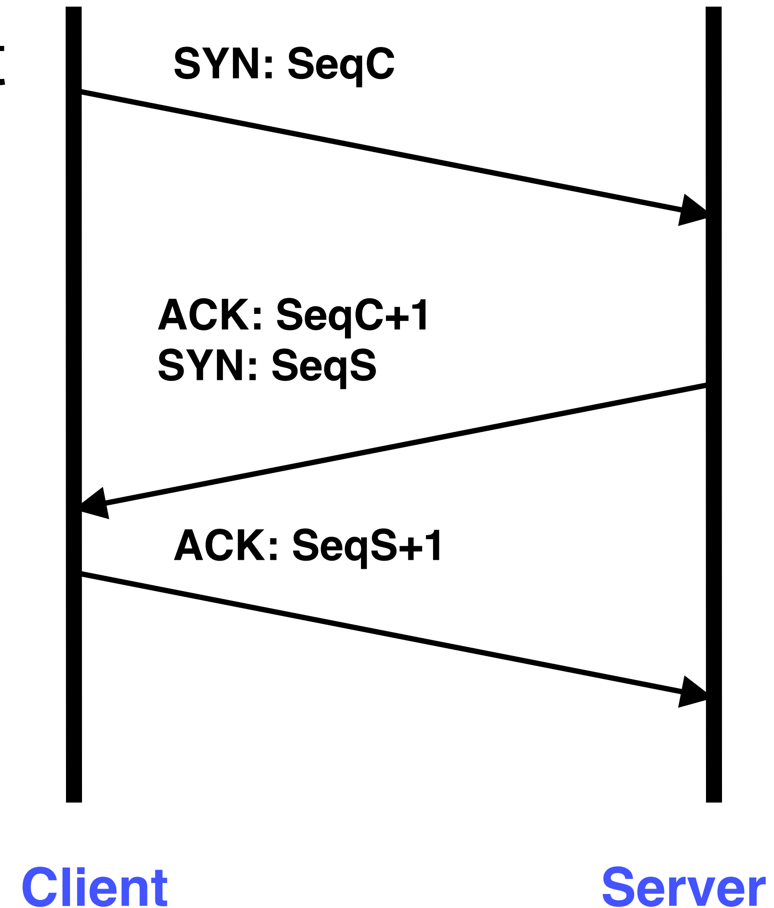


- The window size can be controlled by receiving application
  - Can change the socket buffer size from a default (e.g. 8Kbytes) to a maximum value (e.g. 64 Kbytes)
- The window size field in the TCP header limits the window that the receiver can advertise
  - 16 bits → 64 KBytes
  - TCP options to get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

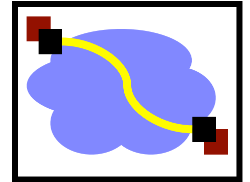
Establishing Connection: Three-Way handshake



- 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

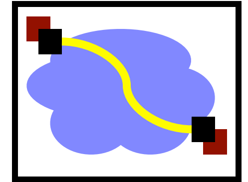


Outline



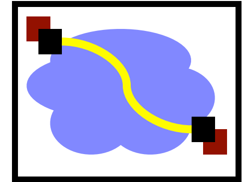
- Transport introduction
- Error recovery & flow control
- TCP flow control/connection setup/data transfer
- TCP reliability
- Congestion sources and collapse
- Congestion control basics

Reliability Challenges



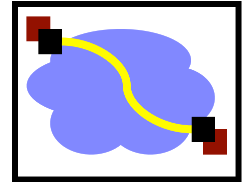
- Congestion related losses
- Variable packet delays
  - What should the timeout be?
- Reordering of packets
  - How to tell the difference between a delayed packet and a lost one?

Round-trip Time Estimation

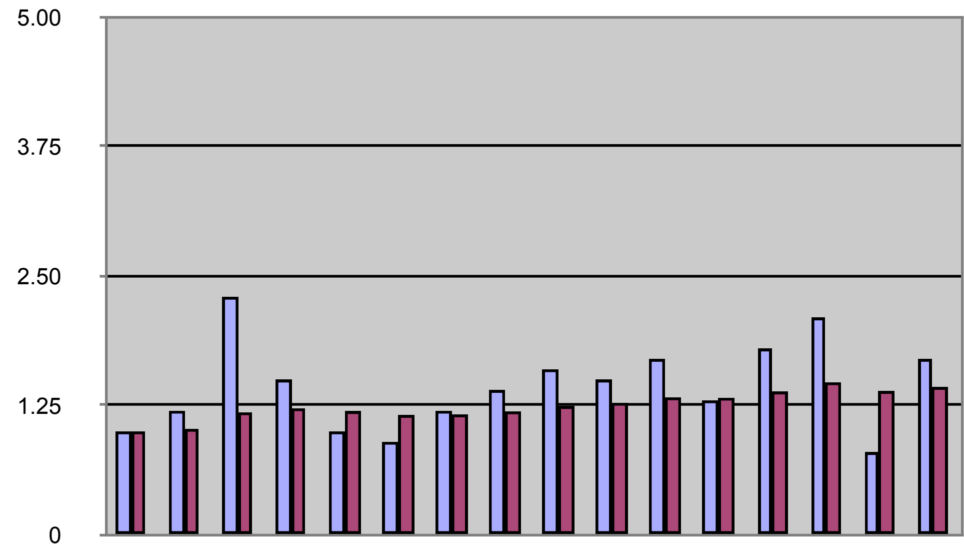


- Wait at least one RTT before retransmitting
- Importance of accurate RTT estimators:
  - Low RTT estimate
    - unneeded retransmissions
  - High RTT estimate
    - poor throughput
- RTT estimator must adapt to change in RTT
  - But not too fast, or too slow!

Original TCP Round-trip Estimator

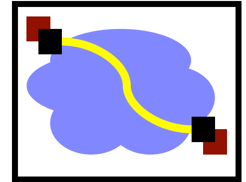


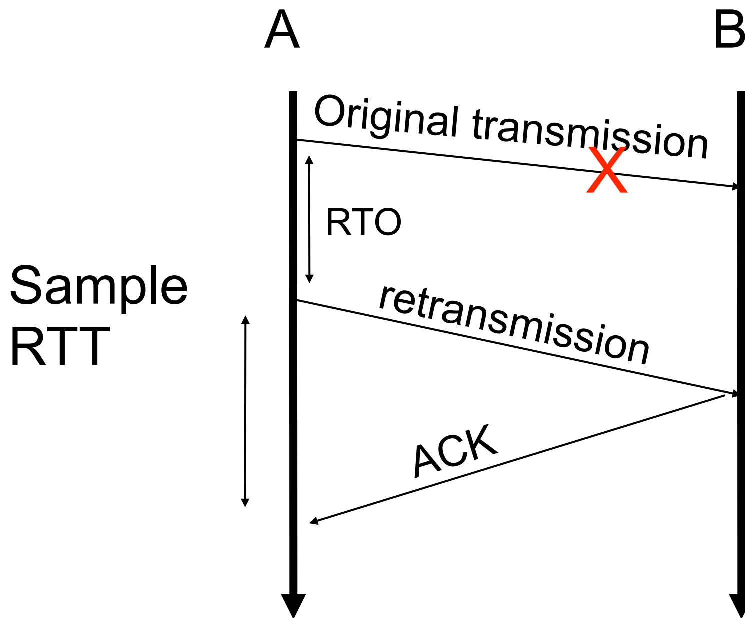
- Round trip times exponentially averaged:
  - New RTT = α (old RTT) + (1 - α) (new sample)
  - Recommended value for α: 0.8 - 0.9
    - 0.875 for most TCP's

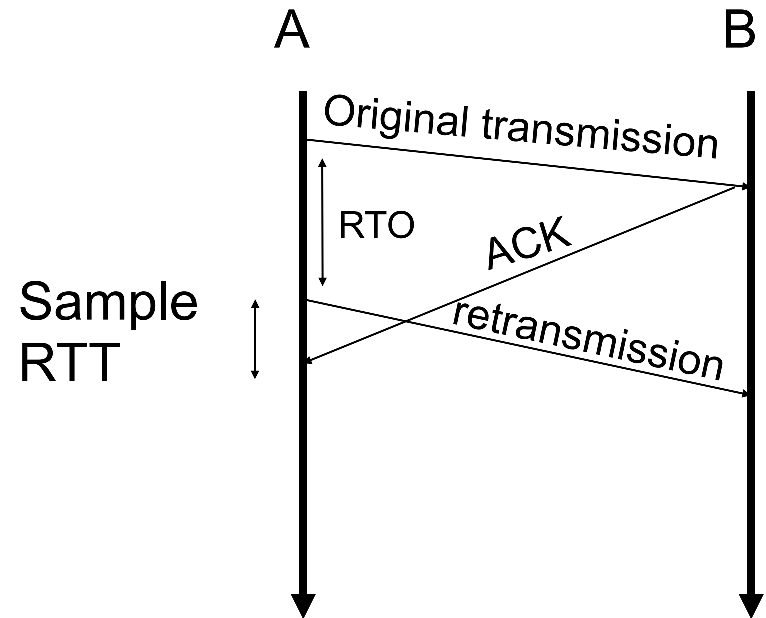


- Retransmit timer set to (b * RTT), where b = 2
  - Every time timer expires, RTO exponentially backed-off

RTT Sample Ambiguity

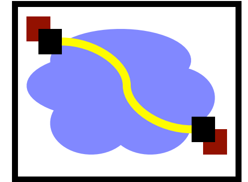






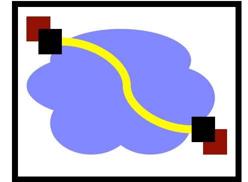
- Karn's RTT Estimator
  - If a segment has been retransmitted:
    - Don't count RTT sample on ACKs for this segment
    - Keep backed off time-out for next packet
    - Reuse RTT estimate only after one successful transmission

Timestamp Extension



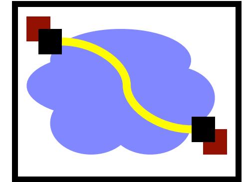
- Used to improve timeout mechanism by more accurate measurement of RTT
- When sending a packet, insert current time into option
  - 4 bytes for time, 4 bytes for echo a received timestamp
- Receiver echoes timestamp in ACK
  - Actually will echo whatever is in timestamp
- Removes retransmission ambiguity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

Timer Granularity



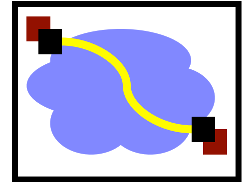
- Many TCP implementations set RTO in multiples of 200,500,1000ms
- Why?
  - Avoid spurious timeouts – RTTs can vary quickly due to cross traffic
  - Reduce timer expensive timer interrupts on hosts
- What happens for the first couple of packets?
  - Pick a very conservative value (seconds)

Fast Retransmit -- Avoiding Timeouts



- What are duplicate acks (dupacks)?
  - Repeated acks for the same sequence
- When can duplicate acks occur?
  - Loss
  - Packet re-ordering
- 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

Fast Retransmit



Sequence No

Retransmission

Duplicate Acks

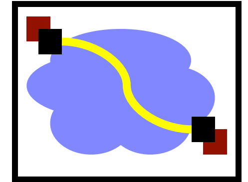
X

Packets

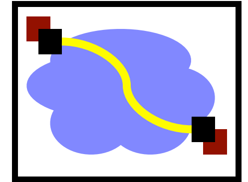
Acks

Time

TCP (Reno variant)



SACK



- Basic problem is that cumulative acks provide little information
- Selective acknowledgement (SACK) 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

SACK

Sequence No

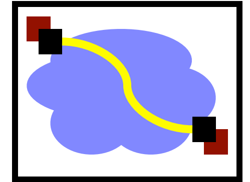
Packets

Acks

Time

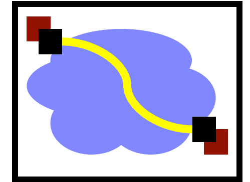
Now what? – send retransmissions as soon as detected

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?

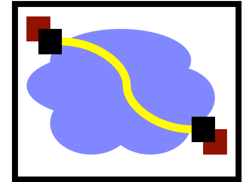
Important Lessons



- Three-way TCP Handshake
- TCP timeout calculation → how is RTT estimated

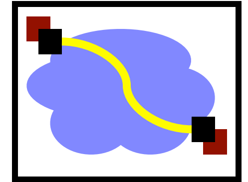
- Modern TCP loss recovery
  - Why are timeouts bad?
  - How to avoid them? → e.g. fast retransmit

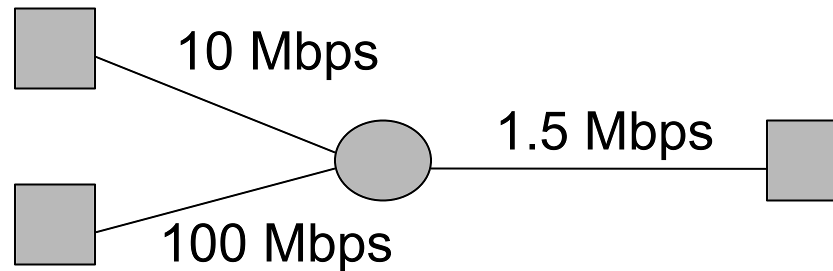
Outline



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

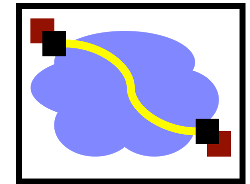
Congestion





- 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)

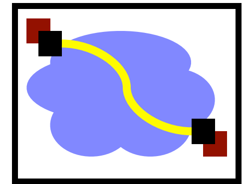
Causes & Costs of Congestion

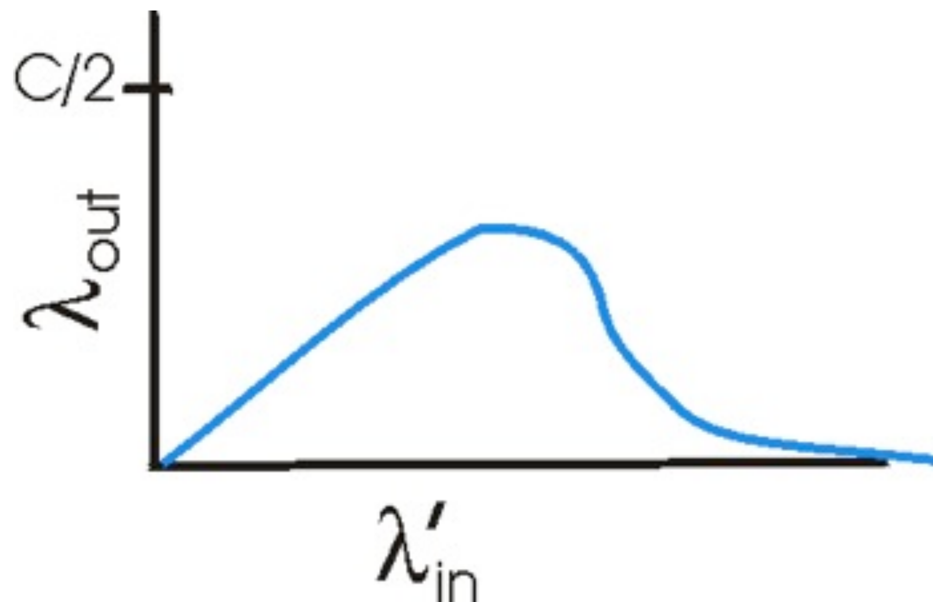


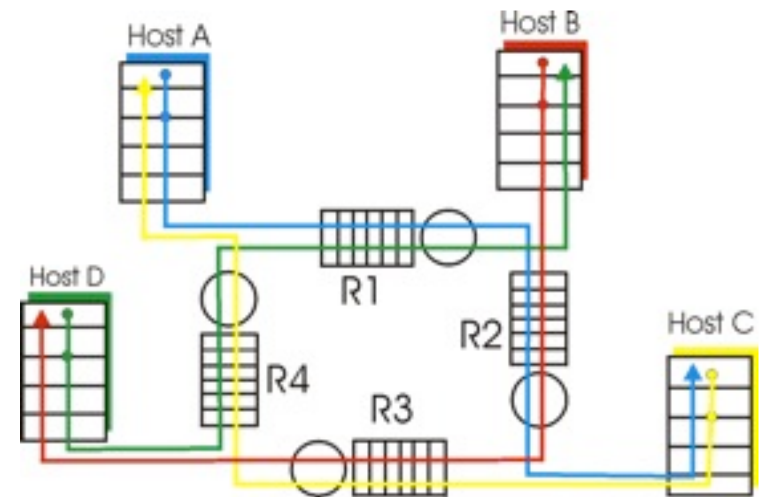
- Four senders – multihop paths
- Timeout/retransmit

Q: What happens as rate increases?

Causes & Costs of Congestion

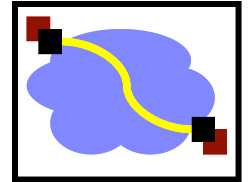






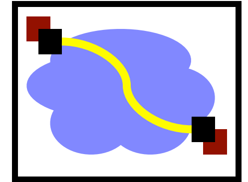
- When packet dropped, any "upstream transmission capacity used for that packet was wasted!

Congestion Collapse



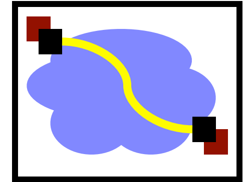
- Definition: Increase in network load results in decrease of useful work done
- Many possible causes
  - Spurious retransmissions of packets still in flight
    - Classical congestion collapse
    - Solution: better timers and TCP congestion control
  - Undelivered packets
    - Packets consume resources and are dropped elsewhere in network
    - Solution: congestion control for ALL traffic
  - Etc..

Where to Prevent Collapse?



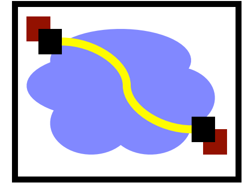
- Can end hosts prevent problem?
  - Yes, but must trust end hosts to do right thing
  - E.g., sending host must adjust amount of data it puts in the network based on detected congestion
- Can routers prevent collapse?
  - No, not all forms of collapse
  - Doesn’t mean they can’t help
    - Sending accurate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

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

Approaches For Congestion Control



- Two broad approaches towards congestion control:

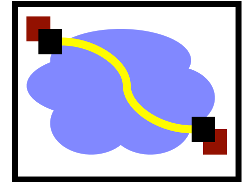
End-to-end

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

Network-assisted

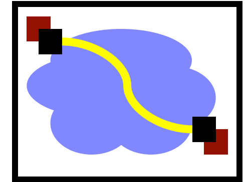
- Routers provide feedback to end systems
  - Explicit rate sender should send at
  - Single bit indicating congestion (SNA, DEC bit, TCP/IP ECN, ATM)
- Problem: makes routers complicated

Example: TCP Congestion Control



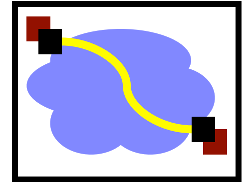
- Very simple mechanisms in network
  - FIFO scheduling with shared buffer pool
  - Feedback through packet drops
- TCP interprets packet drops as signs of congestion and slows down
  - This is an assumption: packet drops are not a sign of congestion in all networks
    - E.g. wireless networks
- Periodically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

Outline



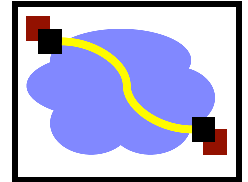
- Transport introduction
- Error recovery & flow control
- TCP flow control/connection setup/data transfer
- TCP reliability
- Congestion sources and collapse
- Congestion control basics

Basic Control Model

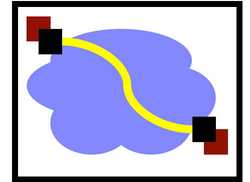


- Let's assume window-based control
- Reduce window when congestion is perceived
  - How is congestion signaled?
    - Either mark or drop packets
  - When is a router congested?
    - Drop tail queues – when queue is full
    - Average queue length – at some threshold
- Increase window otherwise
  -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

Linear Control

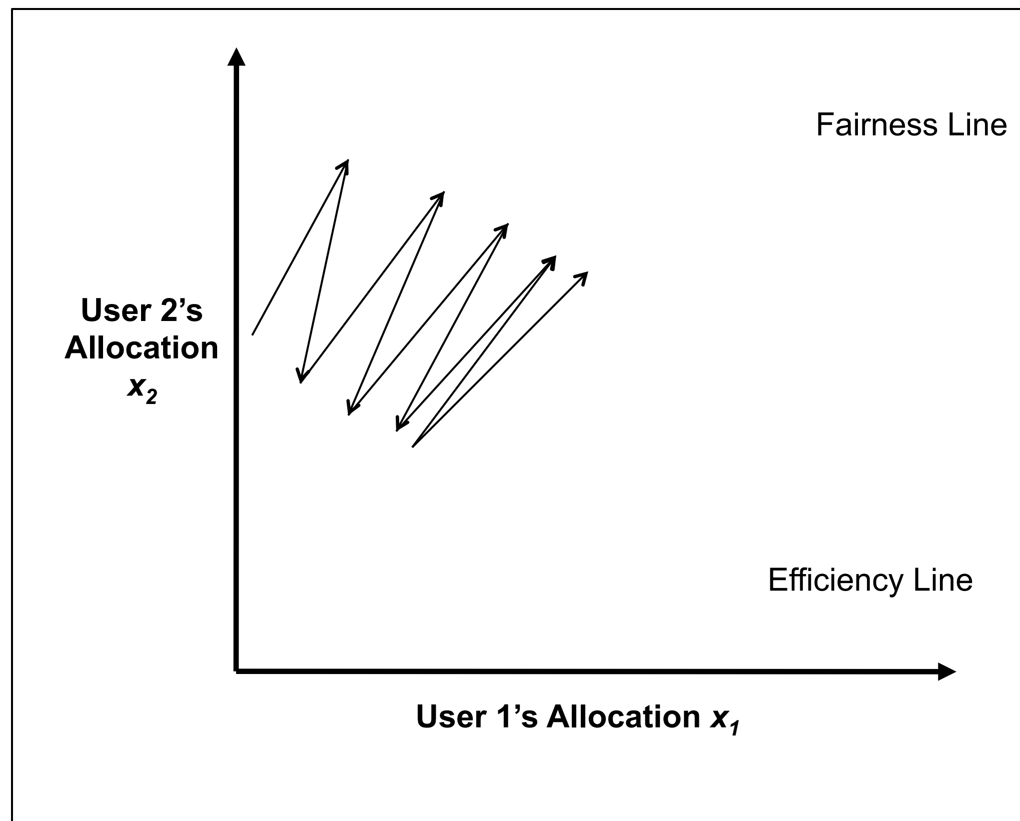


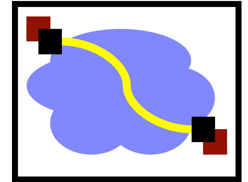
- Many different possibilities for reaction to congestion and probing
  - Examine simple linear controls
  - Window(t + 1) = a + b Window(t)
  - Different a_i/b_i for increase and a_d/b_d for decrease
- Supports various reaction to signals
  - Increase/decrease additively
  - Increased/decrease multiplicatively
  - Which of the four combinations is optimal?



Phase plots

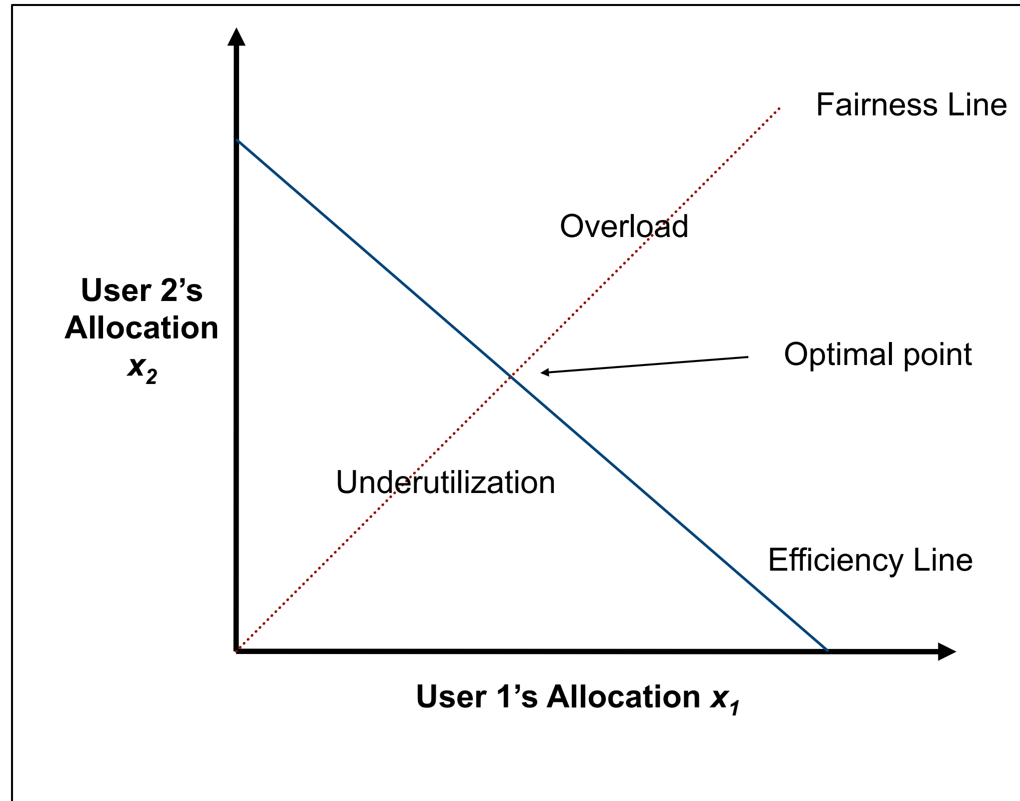
- Simple way to visualize behavior of competing connections over time



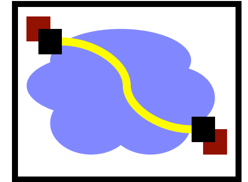


Phase plots

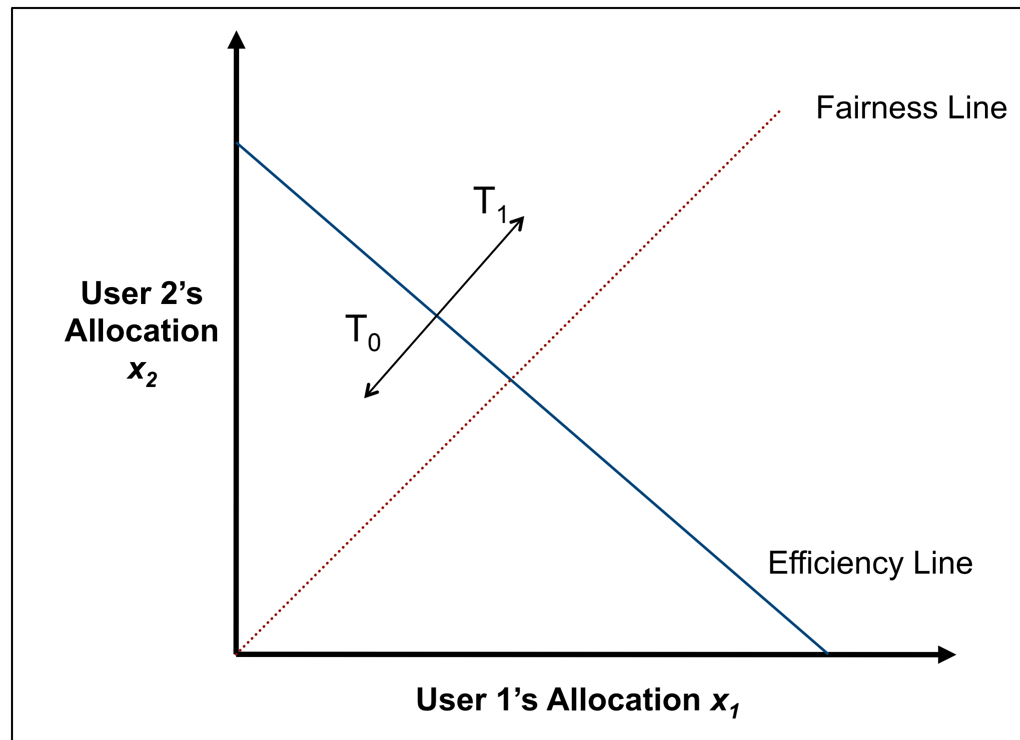
- What are desirable properties?
- What if flows are not equal?



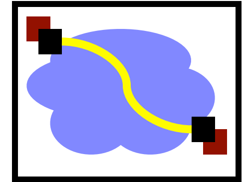
Additive Increase/Decrease



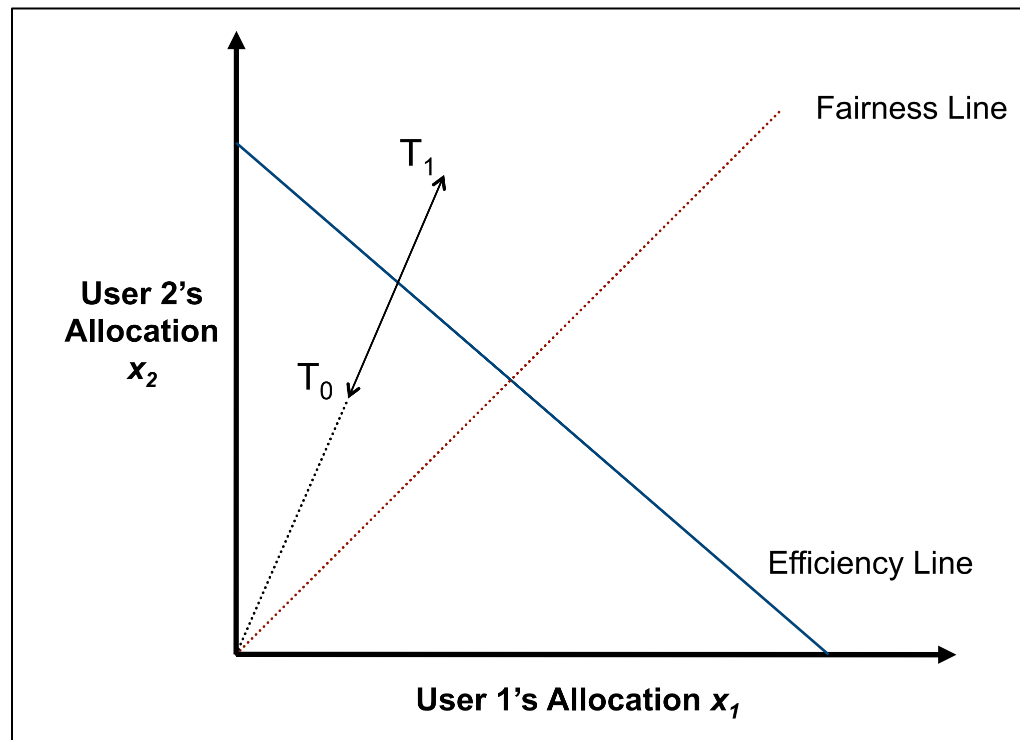
- Both X_1 and X_2 increase/decrease by the same amount over time



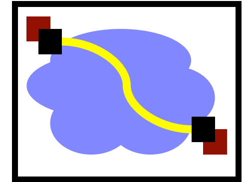
Multiplicative Increase/Decrease

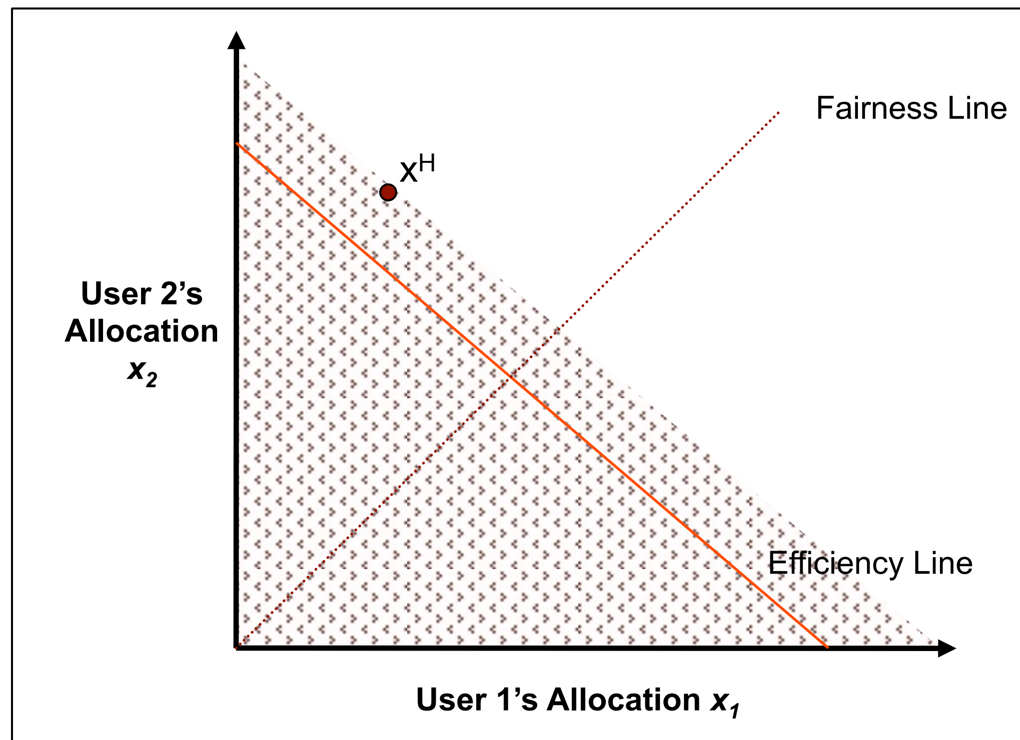


- Both X_1 and X_2 increase by the same factor over time
  - Extension from origin – constant fairness

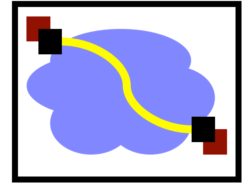


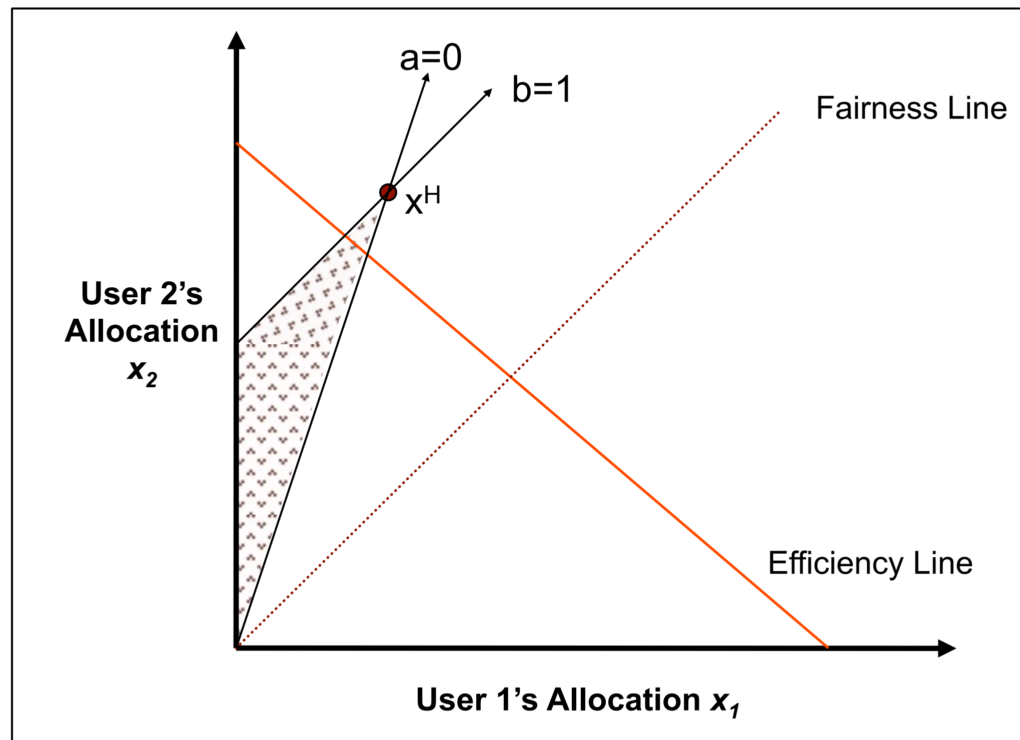
Convergence to Efficiency



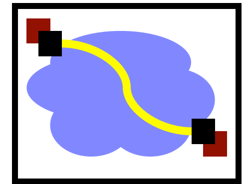


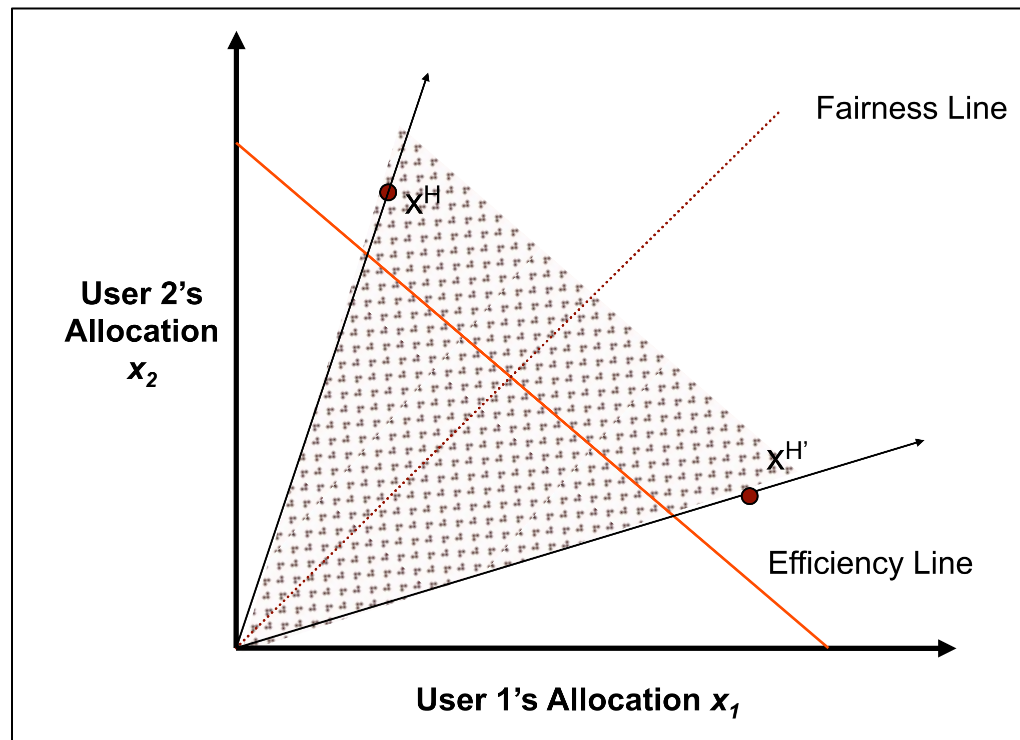
Distributed Convergence to Efficiency



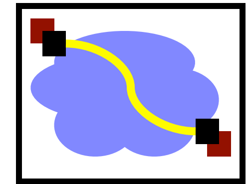


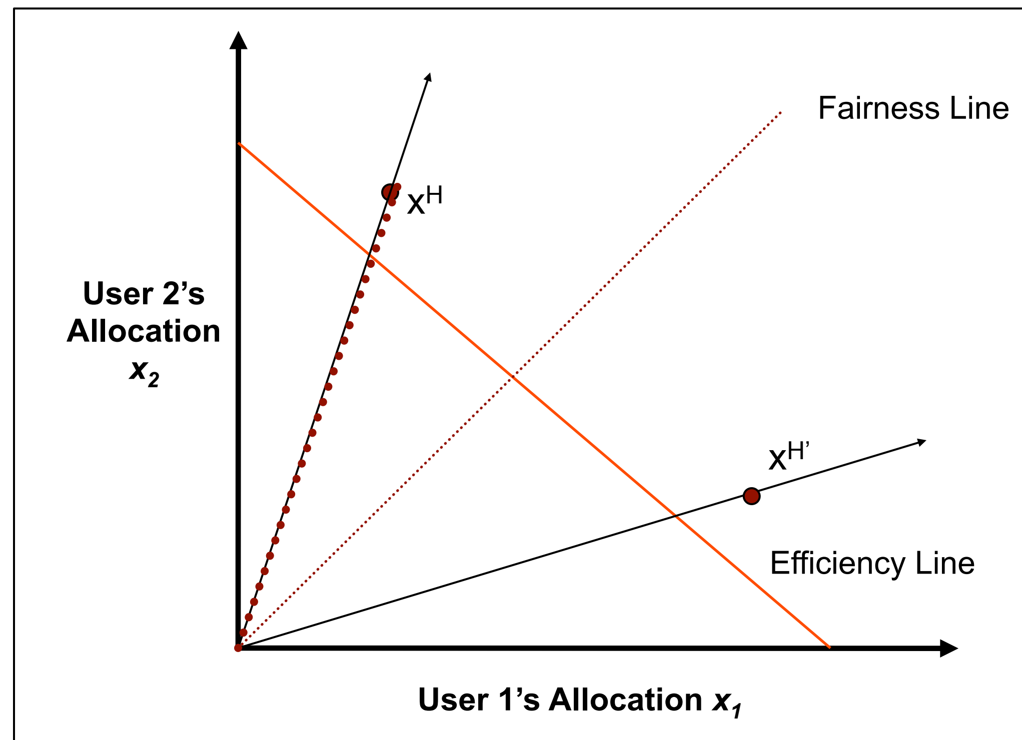
Convergence to Fairness



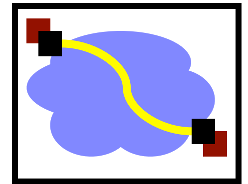


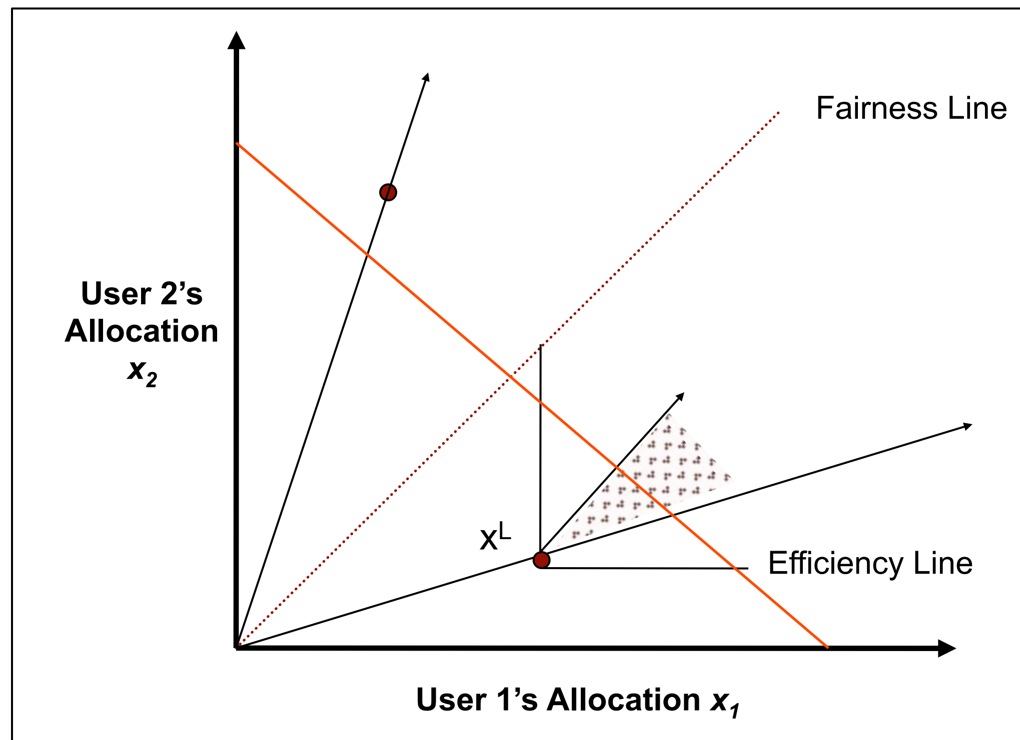
Convergence to Efficiency & Fairness



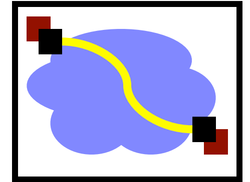


Increase

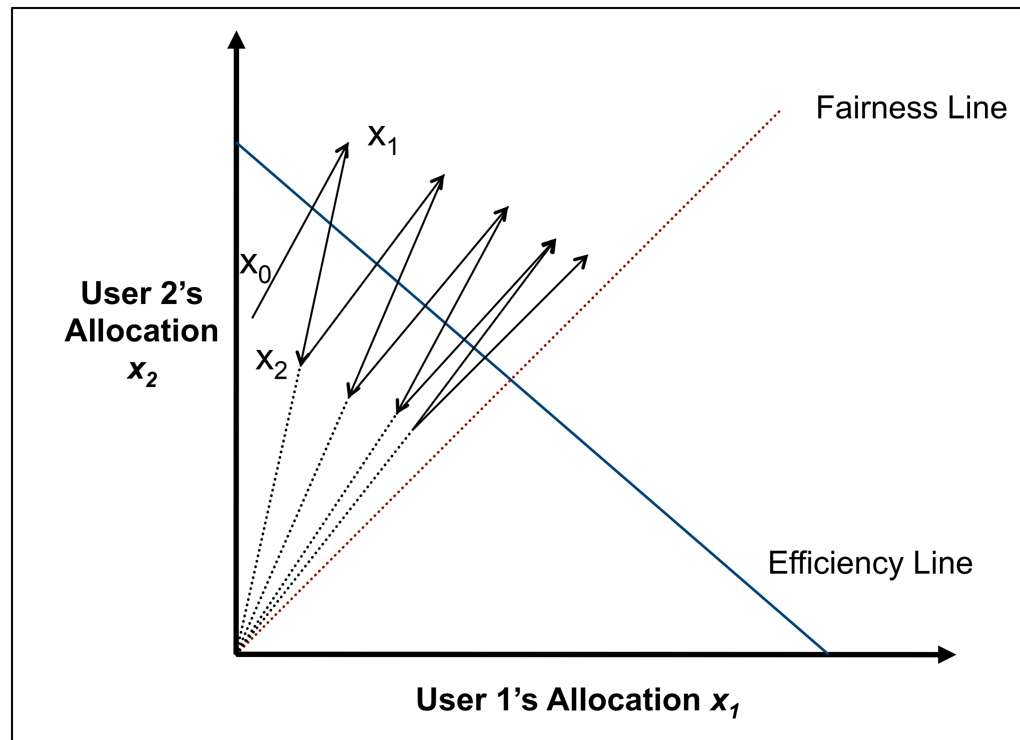




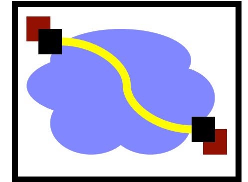
What is the Right Choice?



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



TCP Congestion Control



- Congestion Control
- RED

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