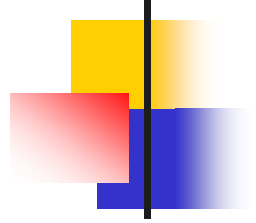


The Transmission Control Protocol



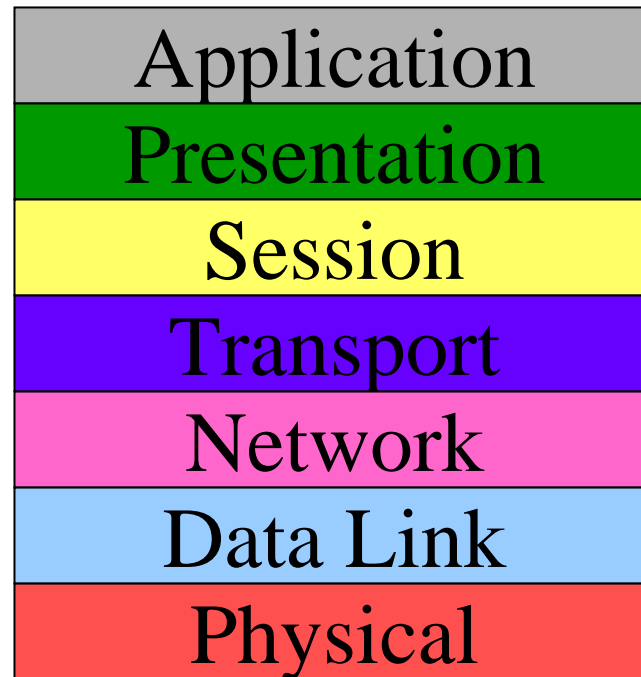
Mostafa Bassiouni
University of Central Florida

August 1, 2005

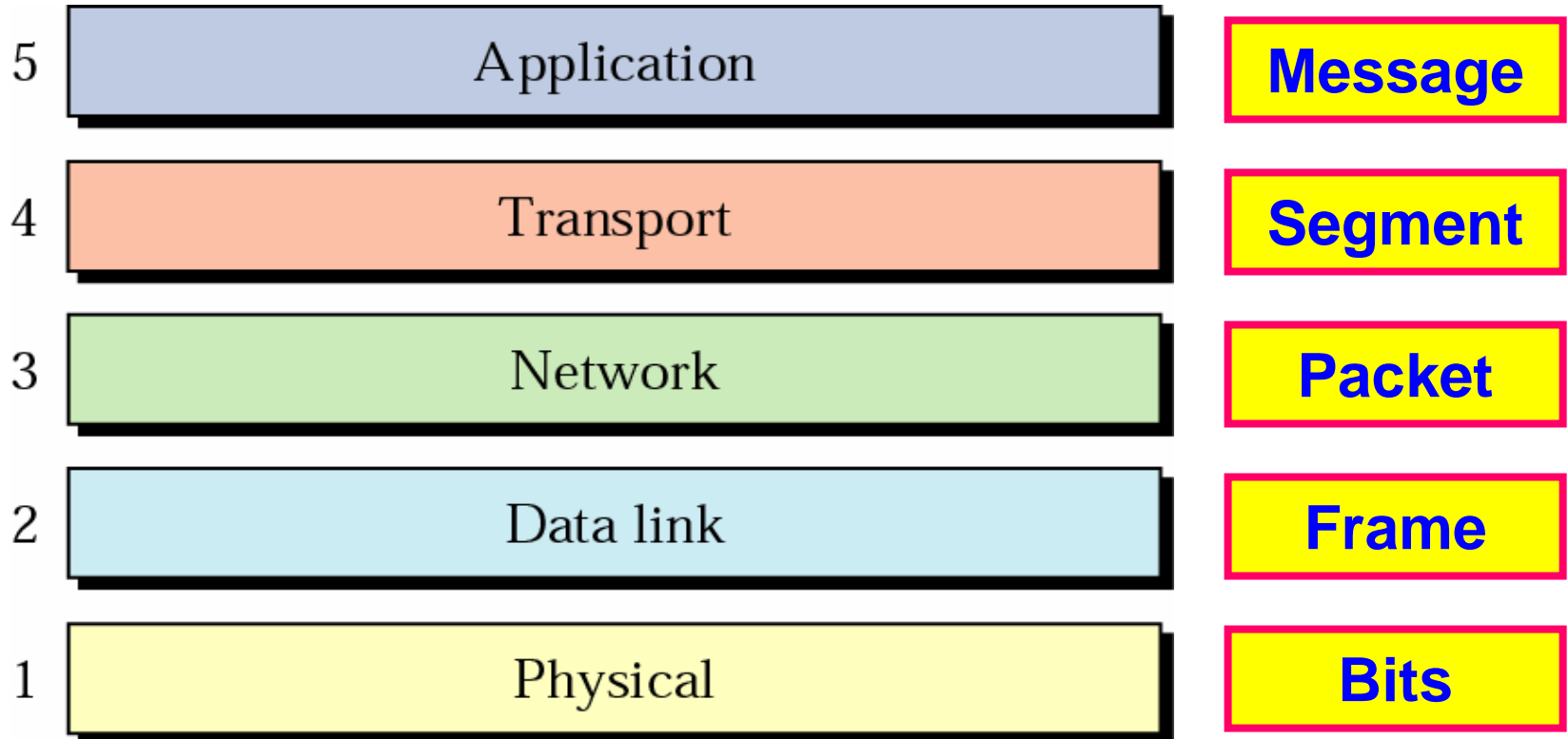
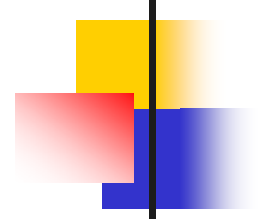


OSI Reference Model

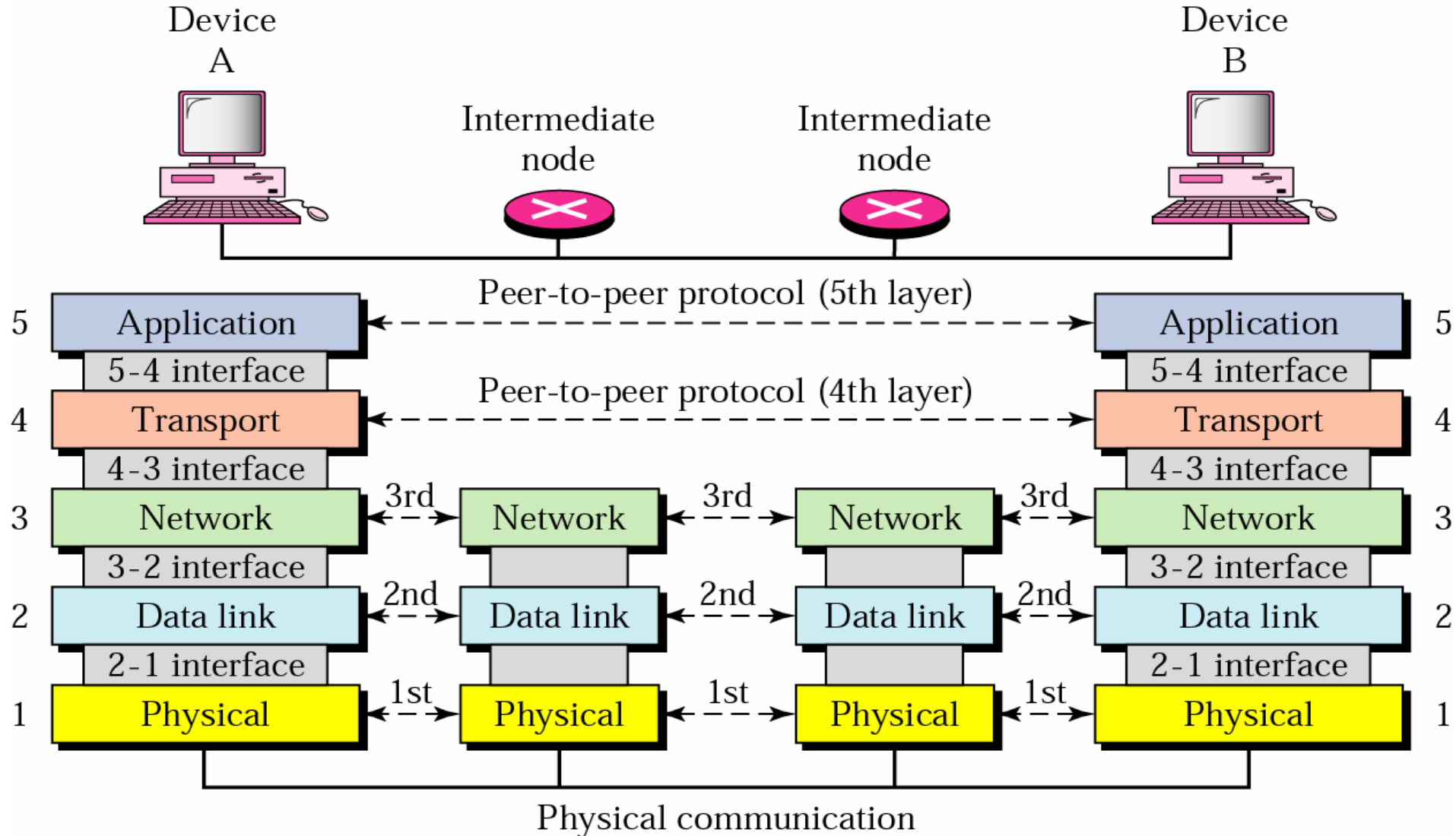
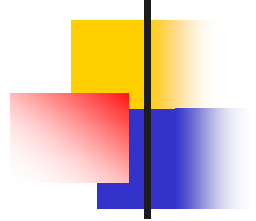
Layered model



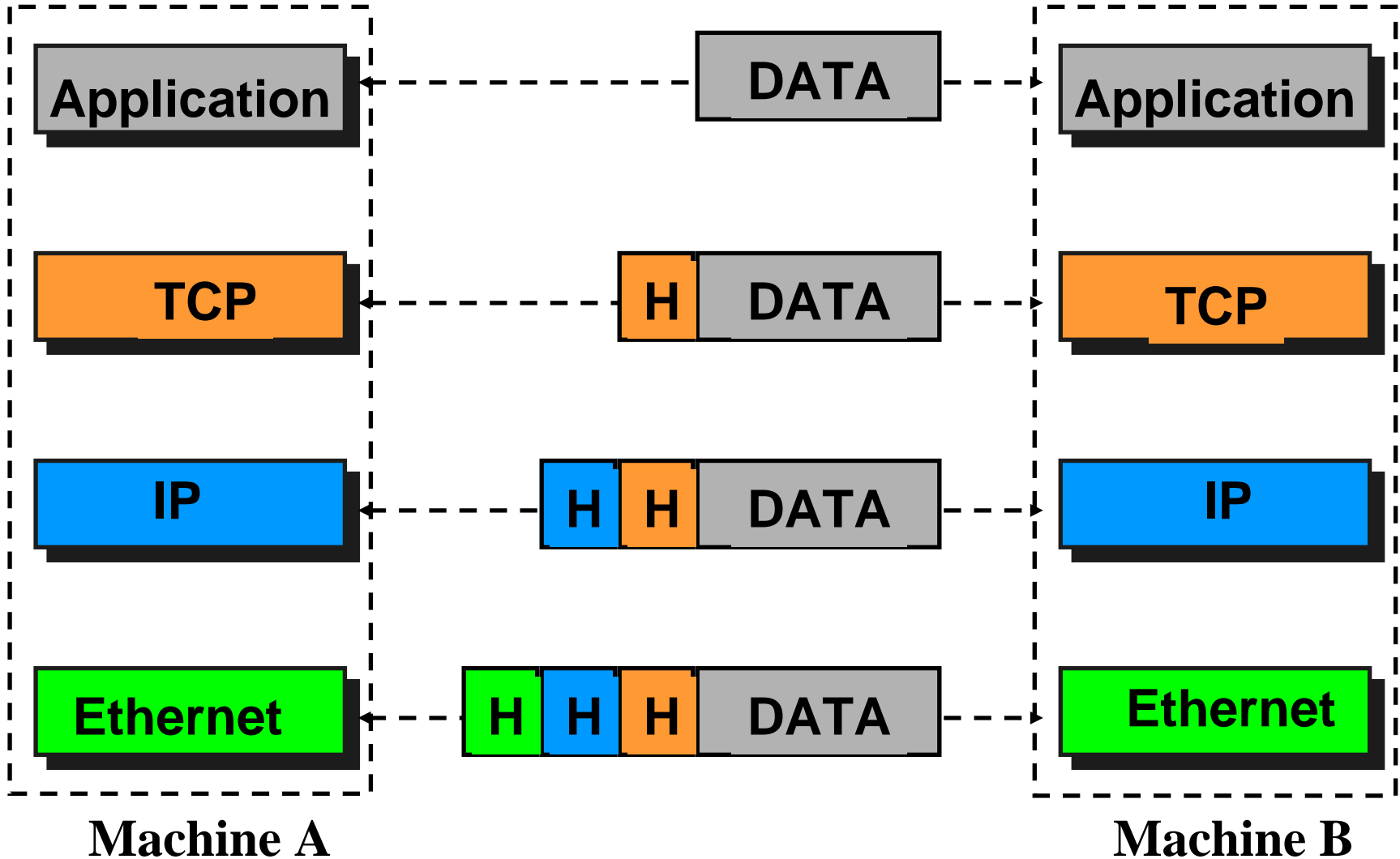
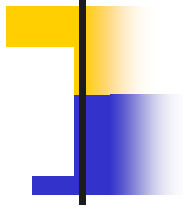
Internet Layers



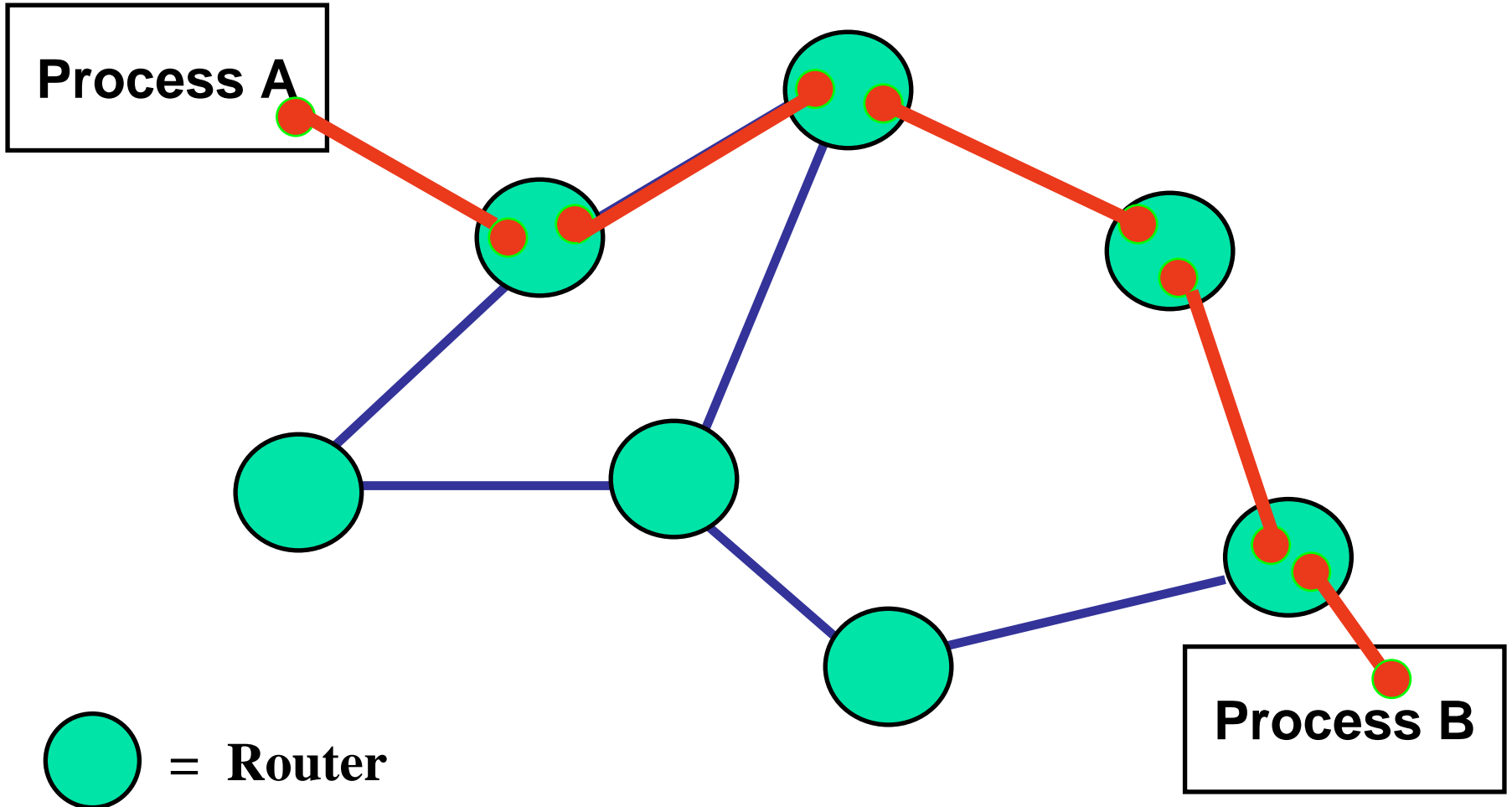
Peer-to-Peer Communications



Layer Headers



TCP End-to-End Semantics



Transmission Control Protocol



■ Flow Control (FC)

- Prevents the sending process from overwhelming the receiving process
- Is implemented using a single variable: the receiver's advertised window, *awnd*

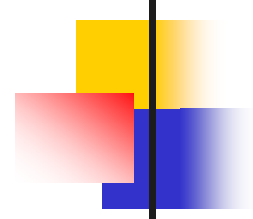
■ Congestion Control (CC)

- Adapts the transmission rate of the sender to match the available capacity of the network
- The congestion window *cwnd* is a *limit* on the amount of data the sender can transmit into the network before receiving an *ACK*

■ Combined FC and CC

- The minimum of *cwnd* and *awnd* governs data transmission

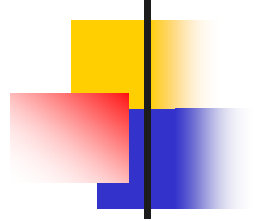
Phases of TCP Congestion Control



- Slow Start Phase
 - Increments the sending rate exponentially
- Congestion Avoidance Phase
 - Increments the sending rate linearly
- Retransmission & Recovery Phase
 - Timeout-based retransmission and recovery
 - Fast retransmission and fast recovery

The slow-start threshold, *ssthresh*, determines whether the *slow-start* or *congestion avoidance* algorithm is used. When $cwnd < ssthresh$, the *slow-start* algorithm is used, otherwise the *congestion avoidance* algorithm is used.

Updating *cwnd*



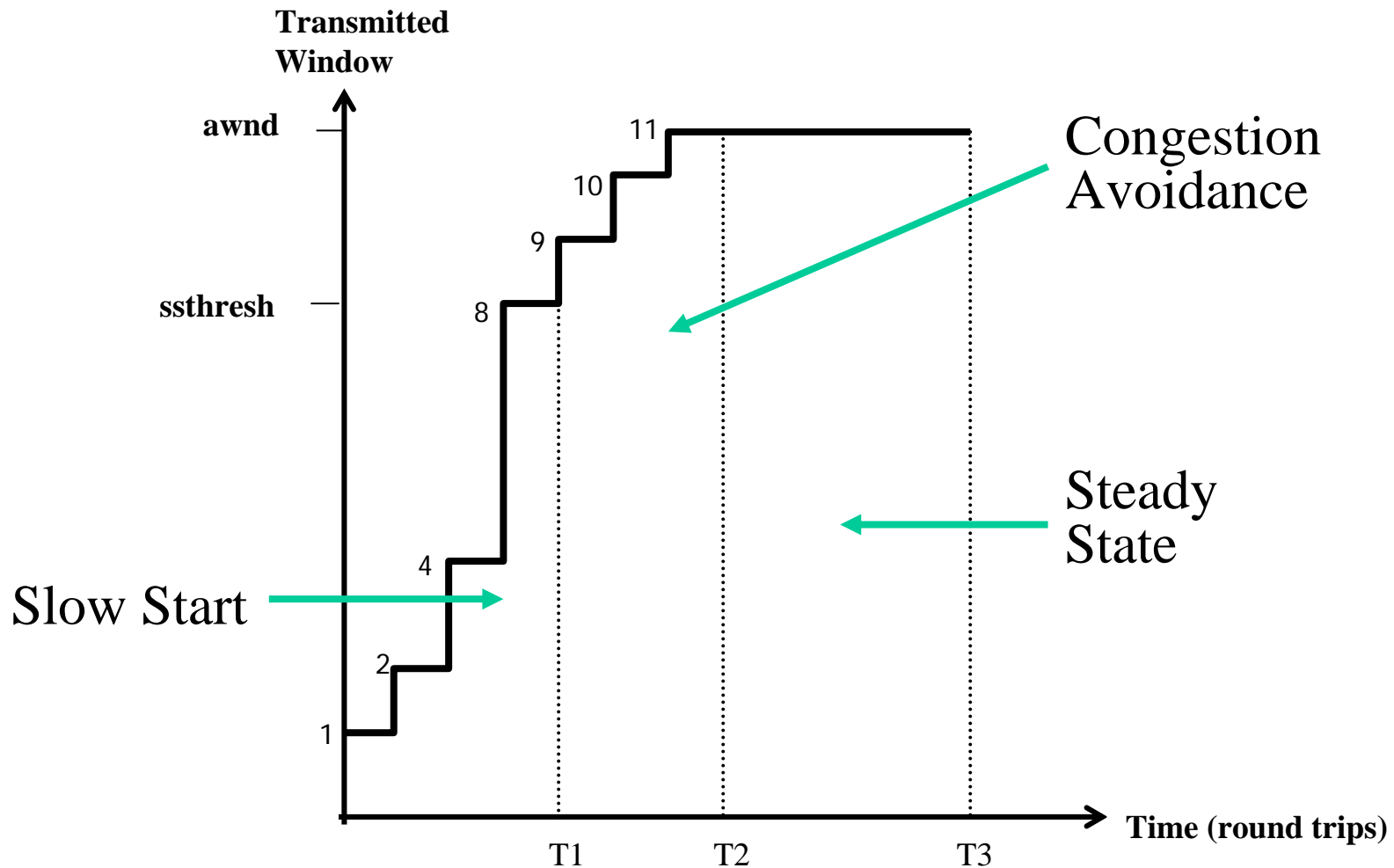
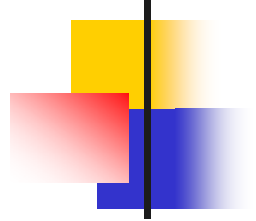
- Maximum Segment Size (*MSS*) : size of the largest segment that the sender can transmit
- During *slow-start*, TCP increments *cwnd* by *MSS* bytes for each new *ACK* received. Thus *cwnd* is doubled every *round-trip time (RTT)* For each new ACK:

$$cwnd = cwnd + MSS$$

- During *congestion avoidance*, TCP increments *cwnd* by *MSS* every *round-trip time*. For each new ACK:

$$cwnd = cwnd + MSS * (MSS / cwnd)$$

Ideal TCP Behavior (No Packet losses)

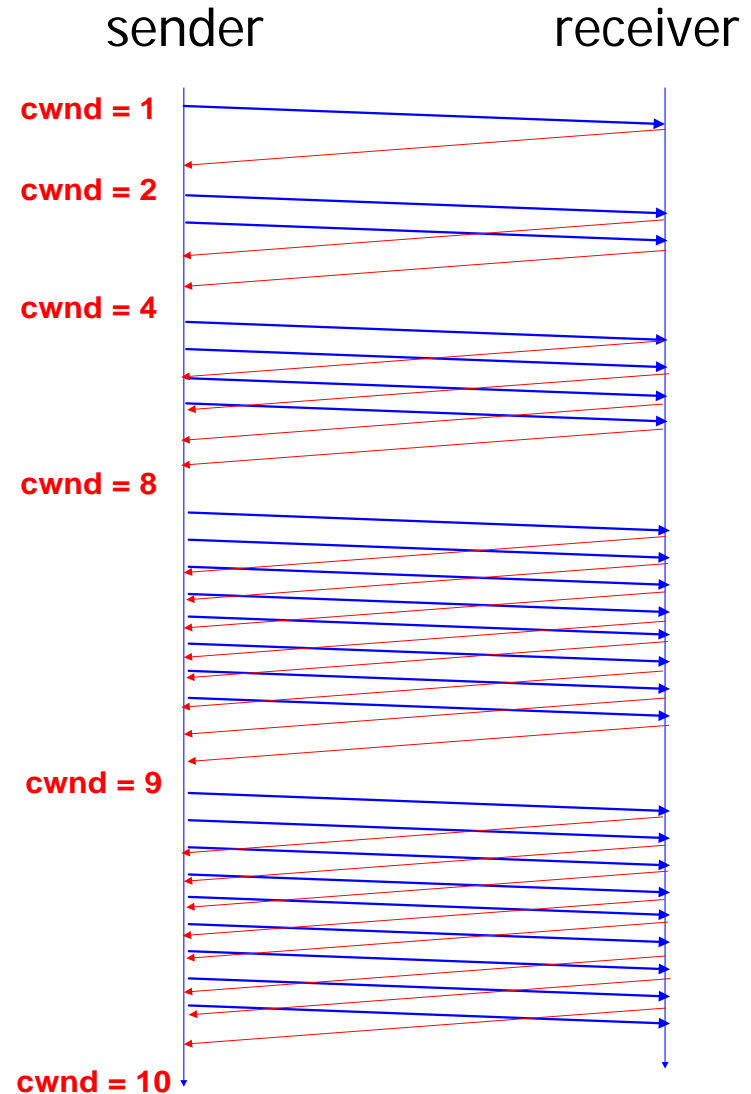
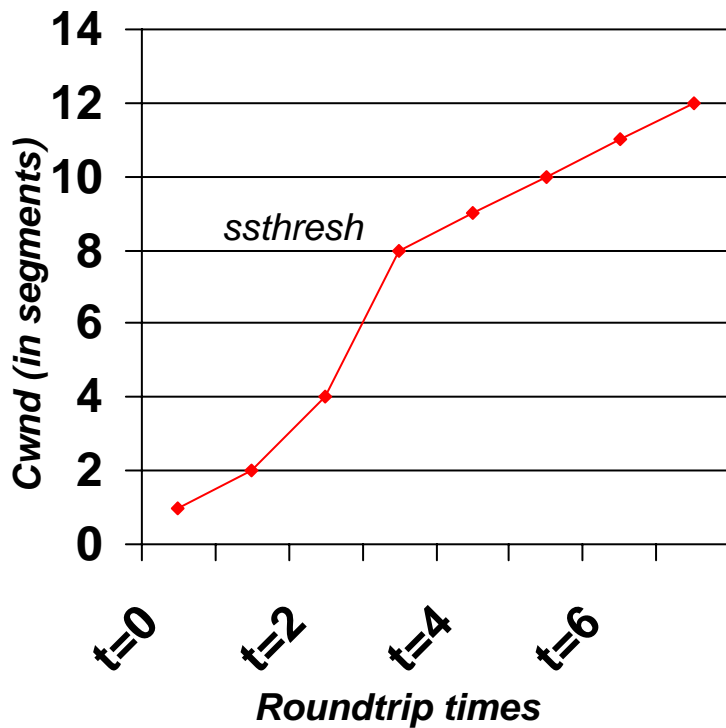


Window size vs. time for ideal TCP

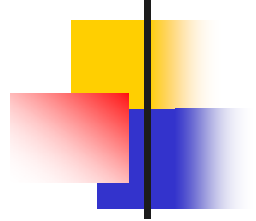
TCP Example



Initially *ssthresh* = $8 * MSS$

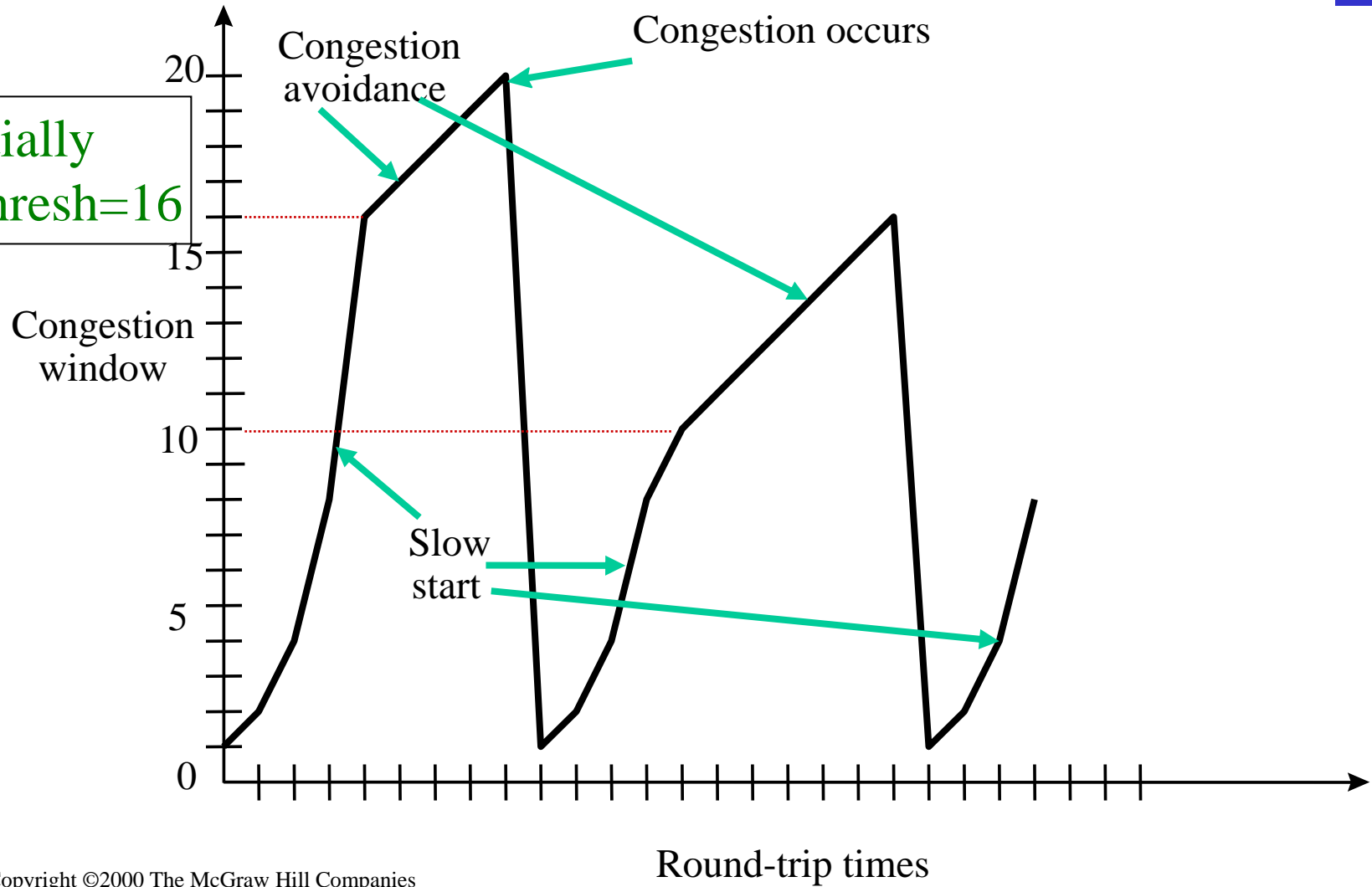
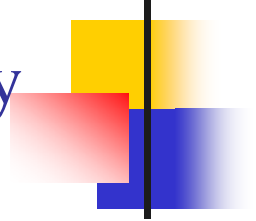


RTO-Based Recovery



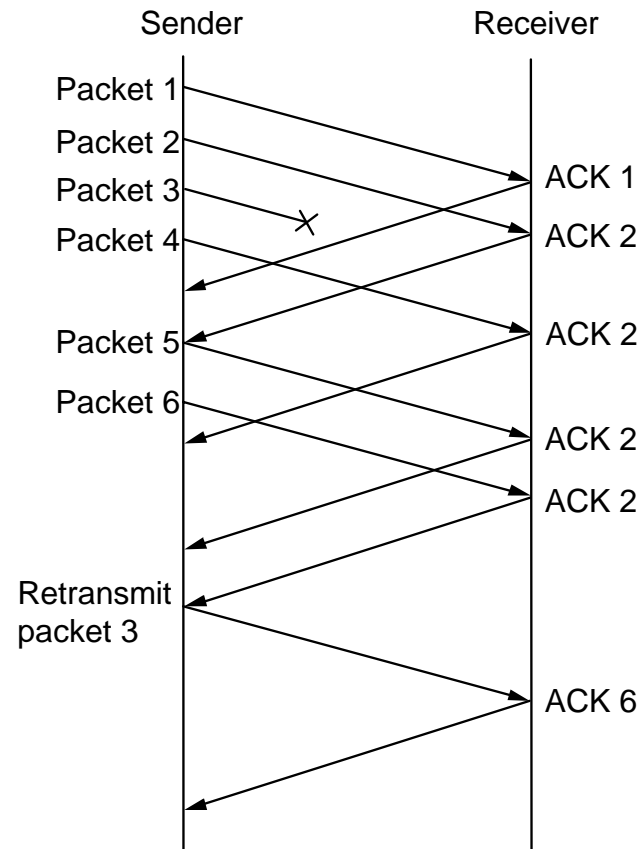
- When the TCP sender has not received an ACK of a segment for a period equal to *RTO* (*retransmission timeout*) after the transmission of that segment, the segment is retransmitted
- *ssthresh* is set to $0.5 * cwnd$
- *cwnd* is set to 1
- TCP enters *slow-start*

TCP Congestion Control with RTO-based Recovery

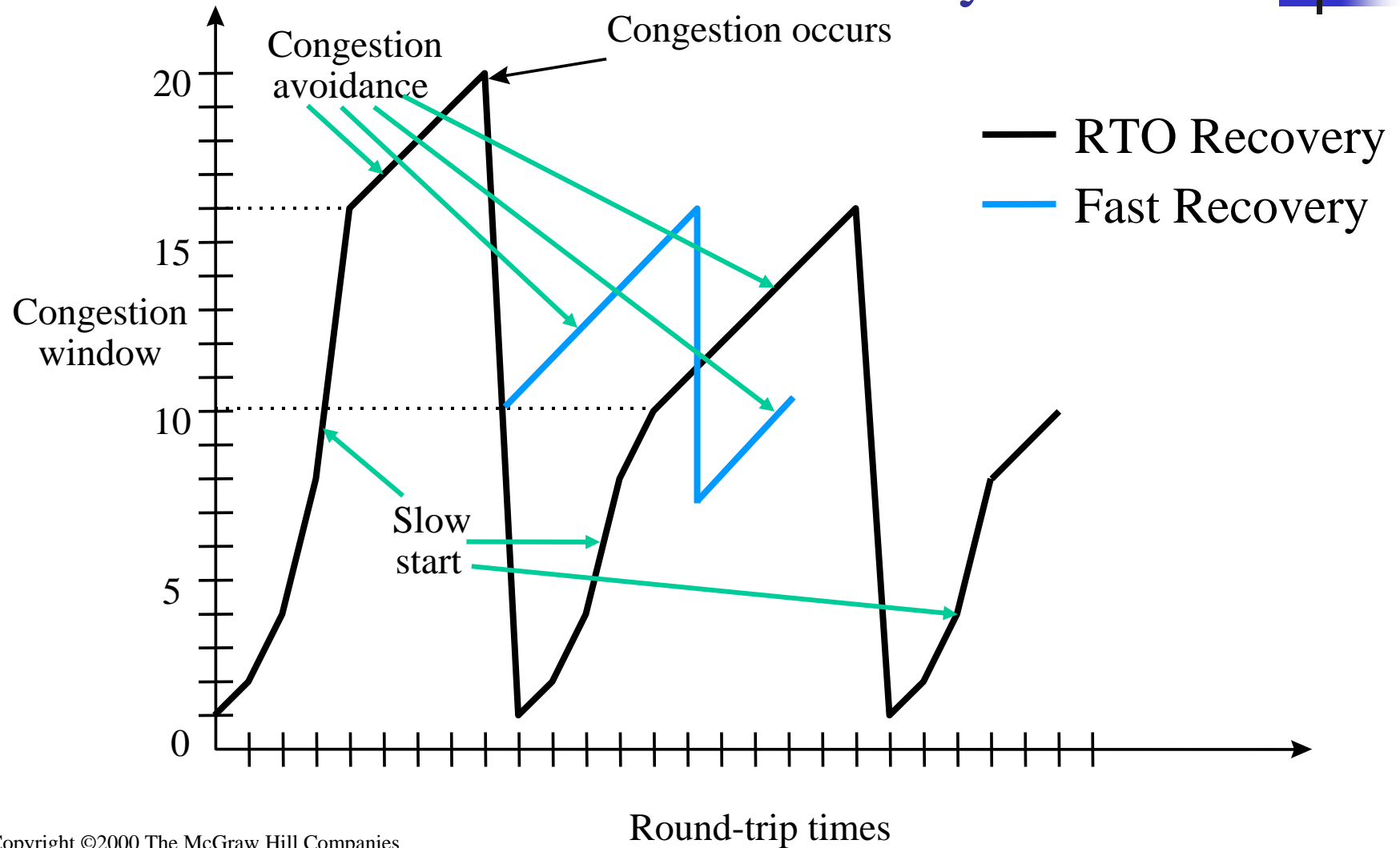
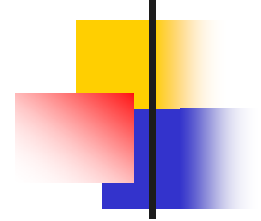


Fast Retransmit and Fast Recovery

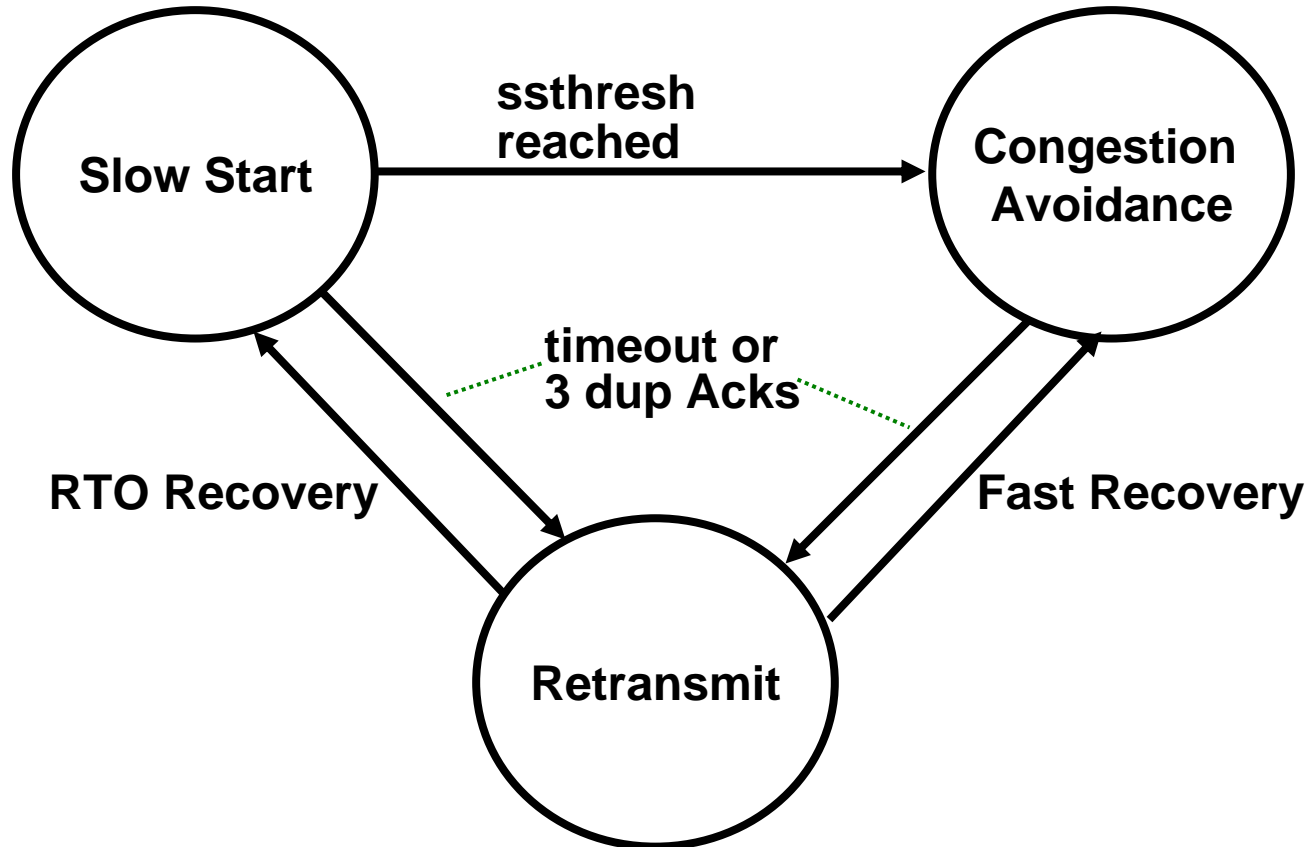
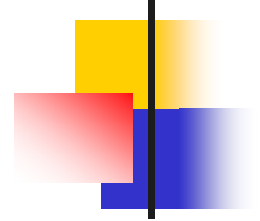
- Fast retransmission is triggered by 3 duplicate ACKs
- $ssthresh$ is set to $0.5 * cwnd$
- When a new Ack arrives, fast recovery starts at $cwnd = ssthresh$ and enters *congestion avoidance*
- TCP operates in AIMD (additive increase/multiplicative decrease) mode



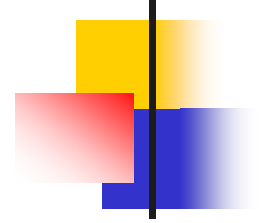
Fast Retransmission/Recovery vs. RTO-based Retransmission/Recovery



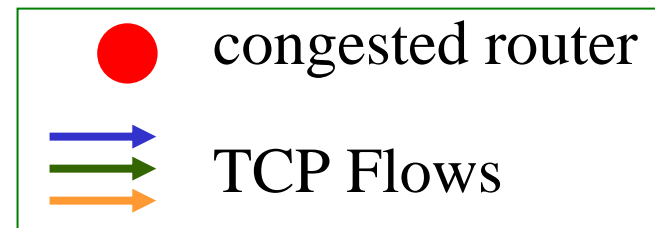
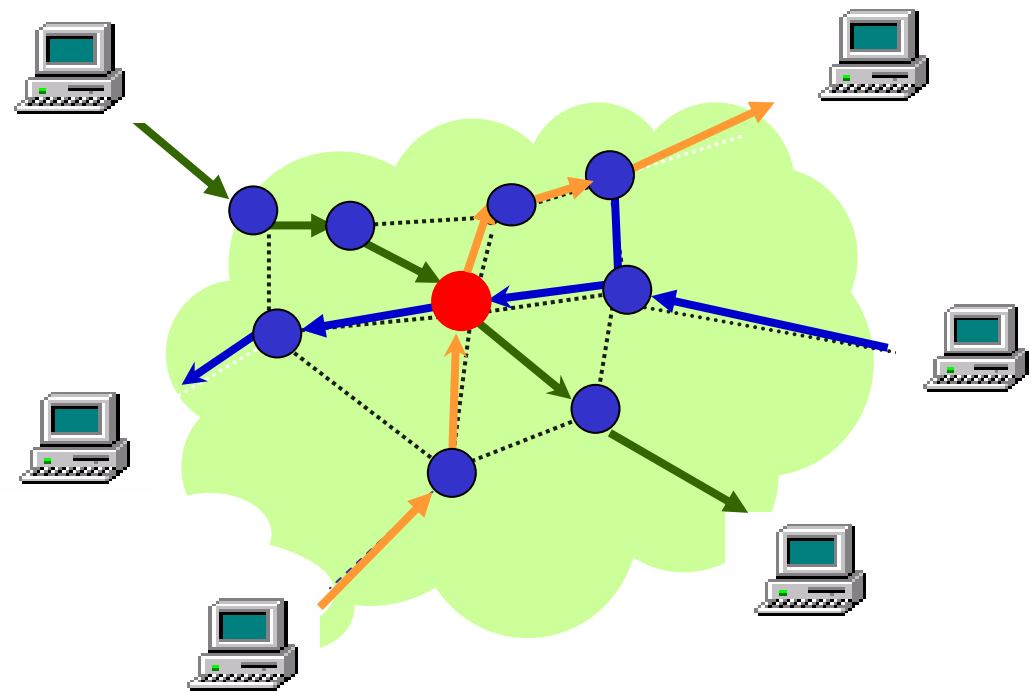
Phases of TCP Congestion Control



TCP Problems



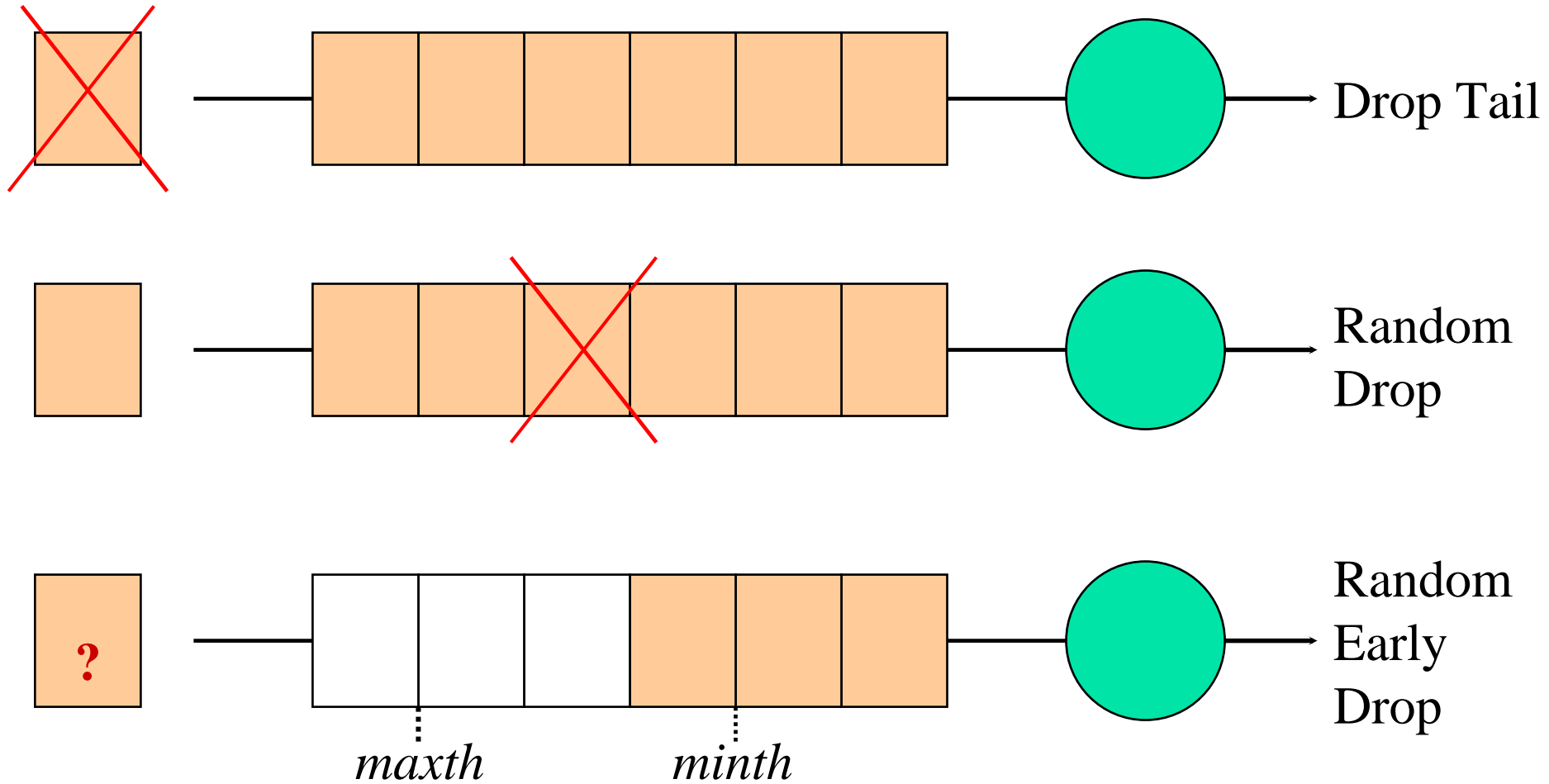
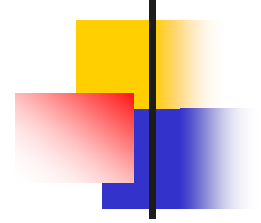
- Congestion Collapse
- Global Synchronization
- The “beat down” problem



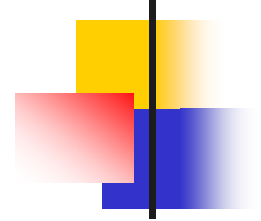
Random Early Detection (RED)

- Can be used to avoid congestion collapse and global synchronization
- Can be used to maintain a smaller average queue, thereby reducing delay and its variability
- Can improve fairness by alleviating bias against bursty traffic

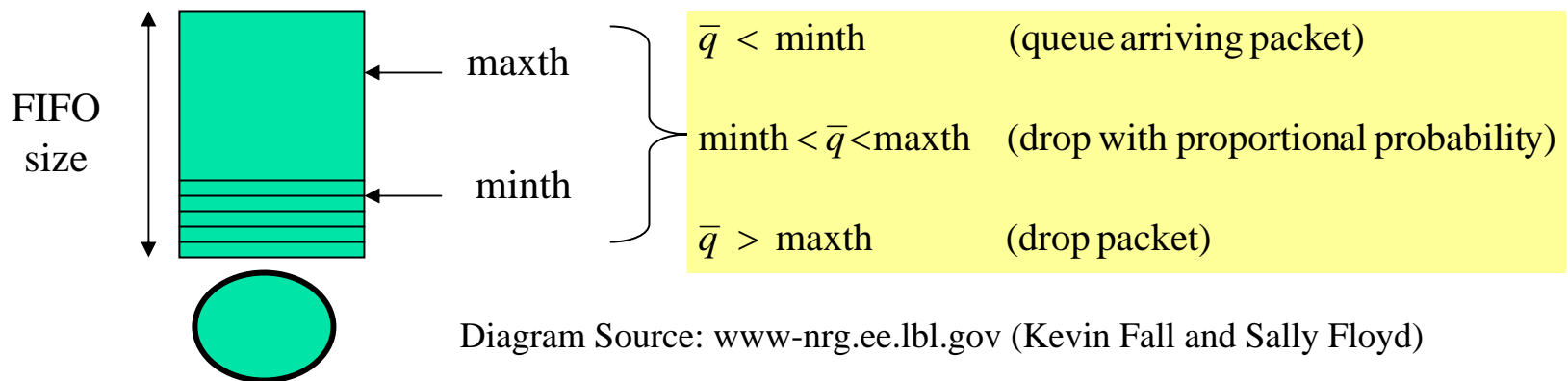
Packet Dropping in Routers



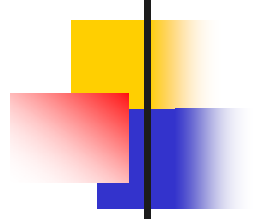
Random Early Detection (RED)



- Red is an active buffer management technique

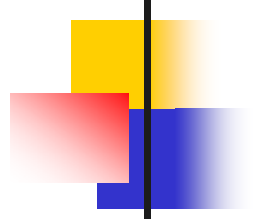


- Type of packet dropping
 - Forced dropping
 - Unforced (proactive) dropping



RED Algorithm

for each packet arrival
calculate the average queue size q_{avg}
if $min_{th} \leq q_{avg} < max_{th}$
 calculate the probability p_a
 with probability p_a :
 drop the arriving packet
else if $max_{th} \leq q_{avg}$
 drop the arriving packet
else if $min_{th} > q_{avg}$
 accept the arriving packet



RED Algorithm (continued)

- RED Drop Probability (p_a)

$$p_b = \max P_p * (q_{avg} - \min_{th}) / (\max_{th} - \min_{th})$$

where

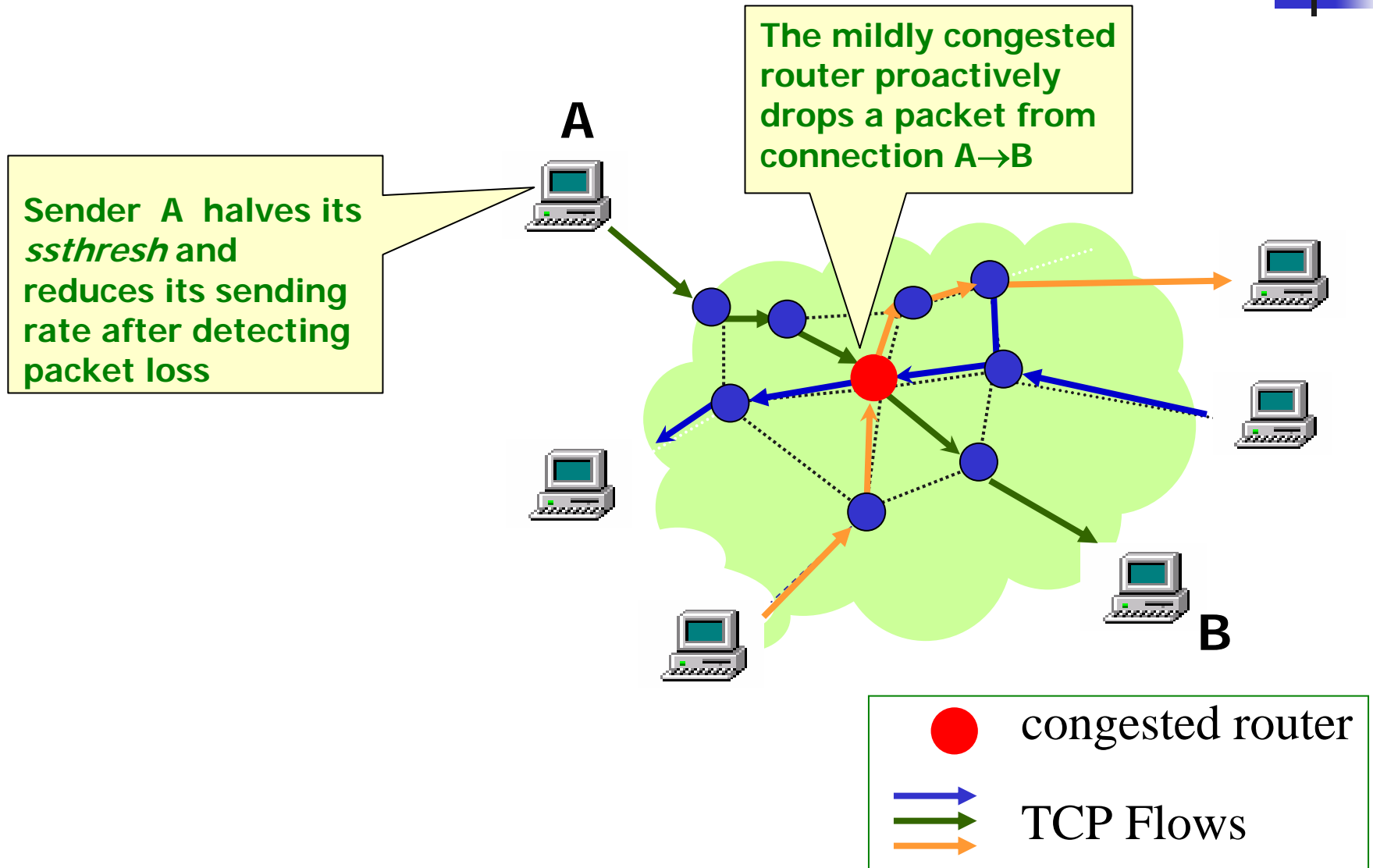
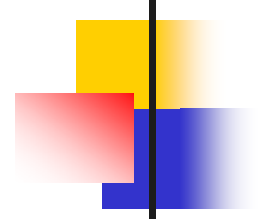
$$p_a = p_b / (1 - count * p_b)$$

- Average Queue Length (q_{avg})

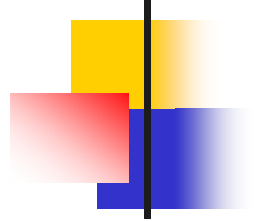
$$q_{avg} = (1 - w_q) * q_{avg} + w_q * q$$

where q is the newly measured queue length

Avoiding Congestion



Using RED for QoS Differentiation



Liang Guo and Ibrahim Matta: *ICNP, November 2001.*

■ Observation

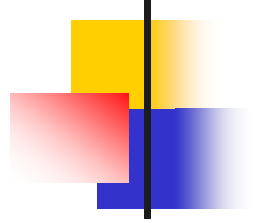
- 80% of the traffic is due to a small number of flows {elephants}
- The remaining traffic volume is due to many short-lived flows {mice}

■ Goal

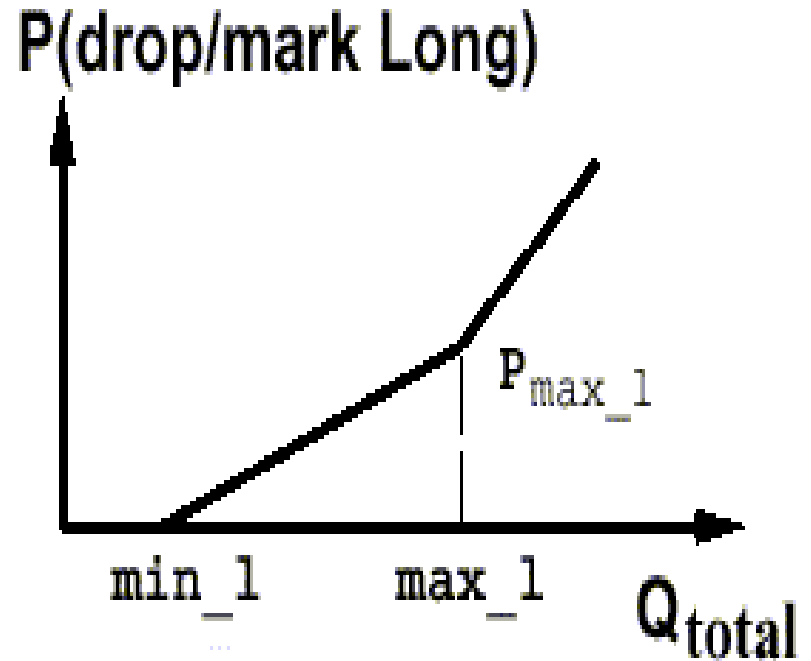
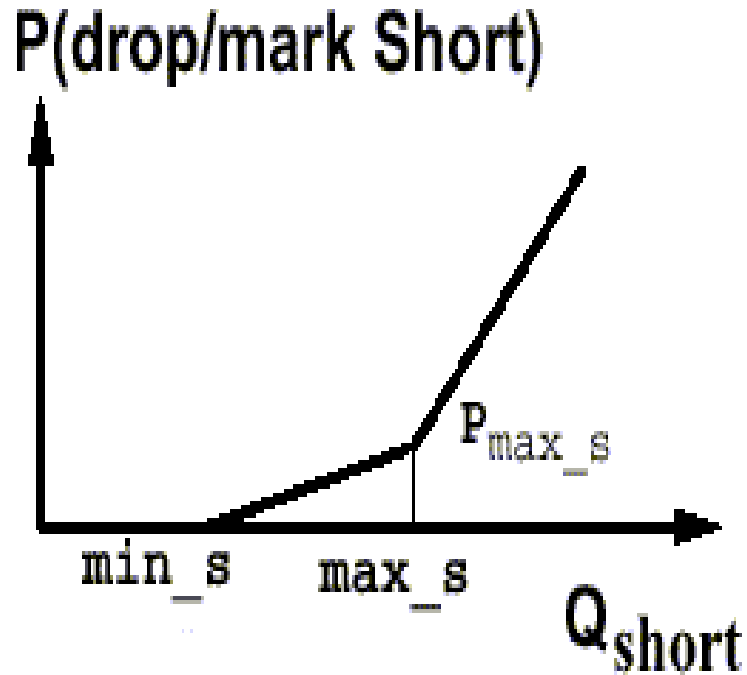
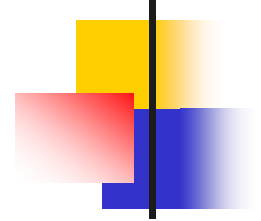
- Provide long-lived flows with expected data rate
- Provide better-than-best-effort service for short TCP flows {Web traffic}

Preferential Treatment to Short Flows

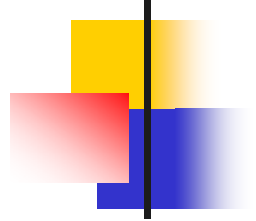
Liang Guo and Ibrahim Matta: *ICNP, November 2001.*



- RED with two flow classes (short and long flows)
- There are two separate sets of RED parameters for each flow class
- Only one real queue exists to avoid packet reordering



RED with Preferential Treatment to Short Flows

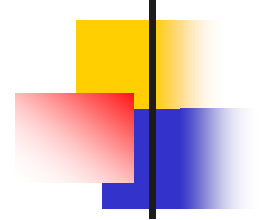


Implementing Preferential Treatment

- Routers divided into edge and core routers
- Intelligence pushed out to edge (ingress) routers; core routers are to be “simple”
- Edge router ‘classifies’ flows and tags packet with classification (e.g., short or long)
- The tag is used by core router to yield preferential treatment for short flows

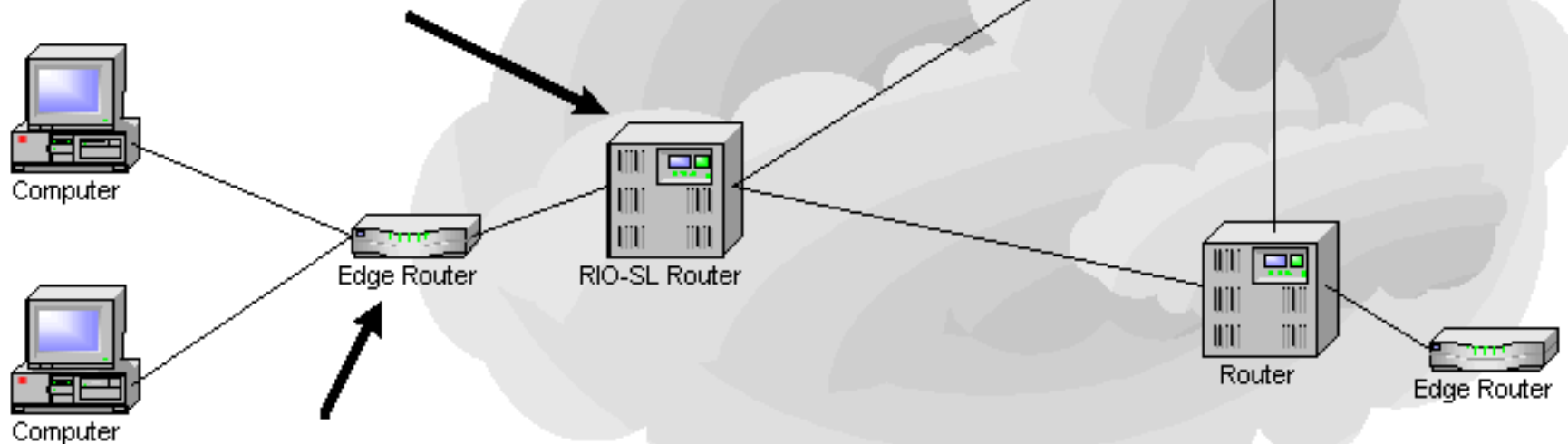
Preferential Treatment to Short Flows

Liang Guo and Ibrahim Matta: *ICNP, November 2001.*



RIO-SL Router (core)

- Uses RIO RED
- Less Drops Short-Lived Flows
- Uses separate RED parameters for each Class (\min_{μ} , \max_{μ} , and \max_p)

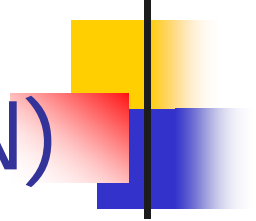


Edge Router

- Classifies Packets as Short or Long
- Perflow State Packet Count

RIO-SL is a version called RED with In and Out to enforce preferential treatment for Short-Lived flows

Explicit Congestion Notification (ECN)

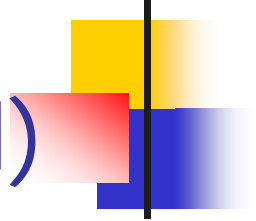


RFC 3168 (K. Ramakrishnan, S. Floyd and D. Black), September 2001

Motivation

- Active queue management (RED router) is successful in
 - avoiding synchronization of loss across multiple flows
 - maintaining a smaller average queue size
 - attempting to increase fairness
- But packet drops are still the indication of congestion

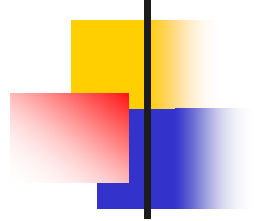
Explicit Congestion Notification (ECN)



RFC 3168 (K. Ramakrishnan, S. Floyd and D. Black), September 2001

- ECN uses “marked” instead of “dropped” packets for congestion indication
- ECN can react to congestion *before* packets get lost
- Currently ECN is only defined for data packets not for ACK packets

Applications that can benefit from ECN

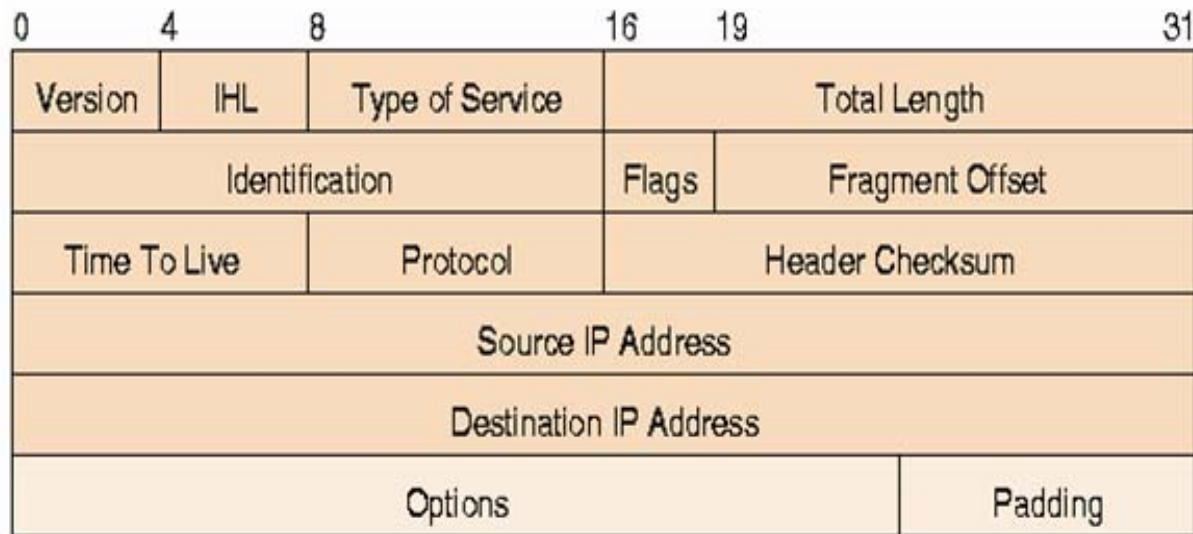


- Very short web transfers (e.g., with one or two packets)
- Real-time flows with fixed or adaptive playback times; user would rather receive the packet on time
- Low bandwidth telnet connections
- Reliable multicast - where overhead costs of retransmitting dropped packets can be expensive

Changes in IP Header



Old Format



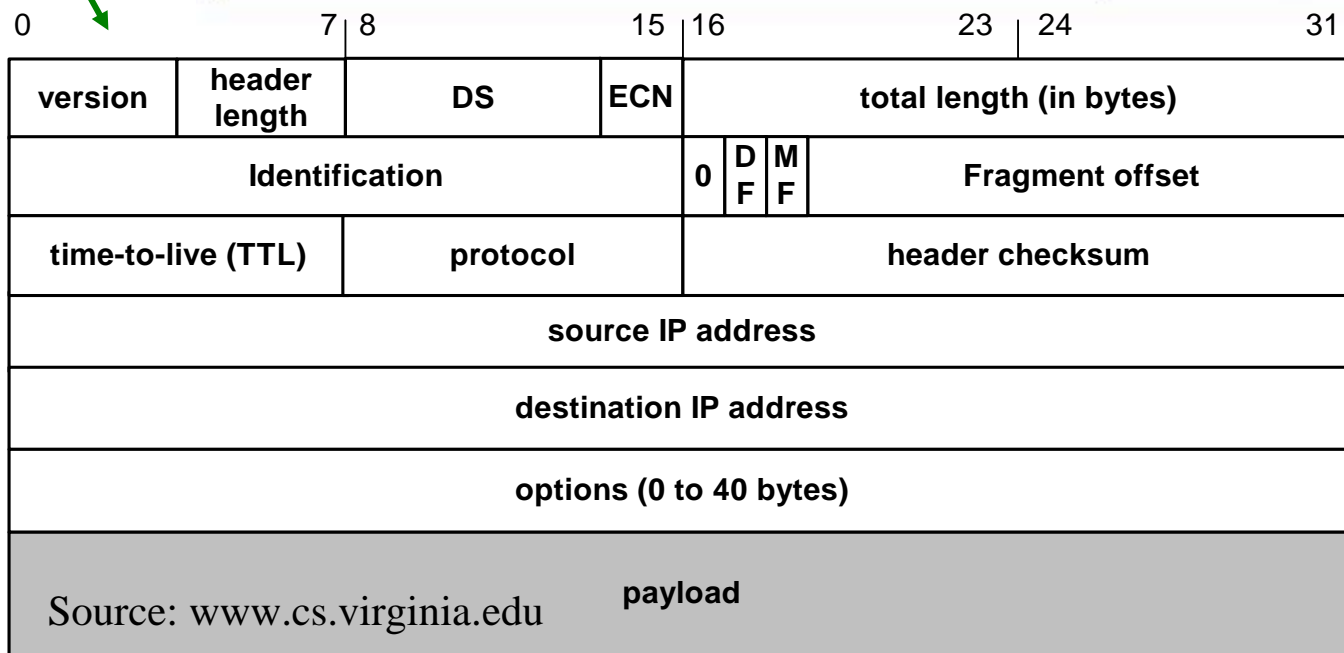
New Format uses two new ECN bits

ECT (ECN-Capable Transport)

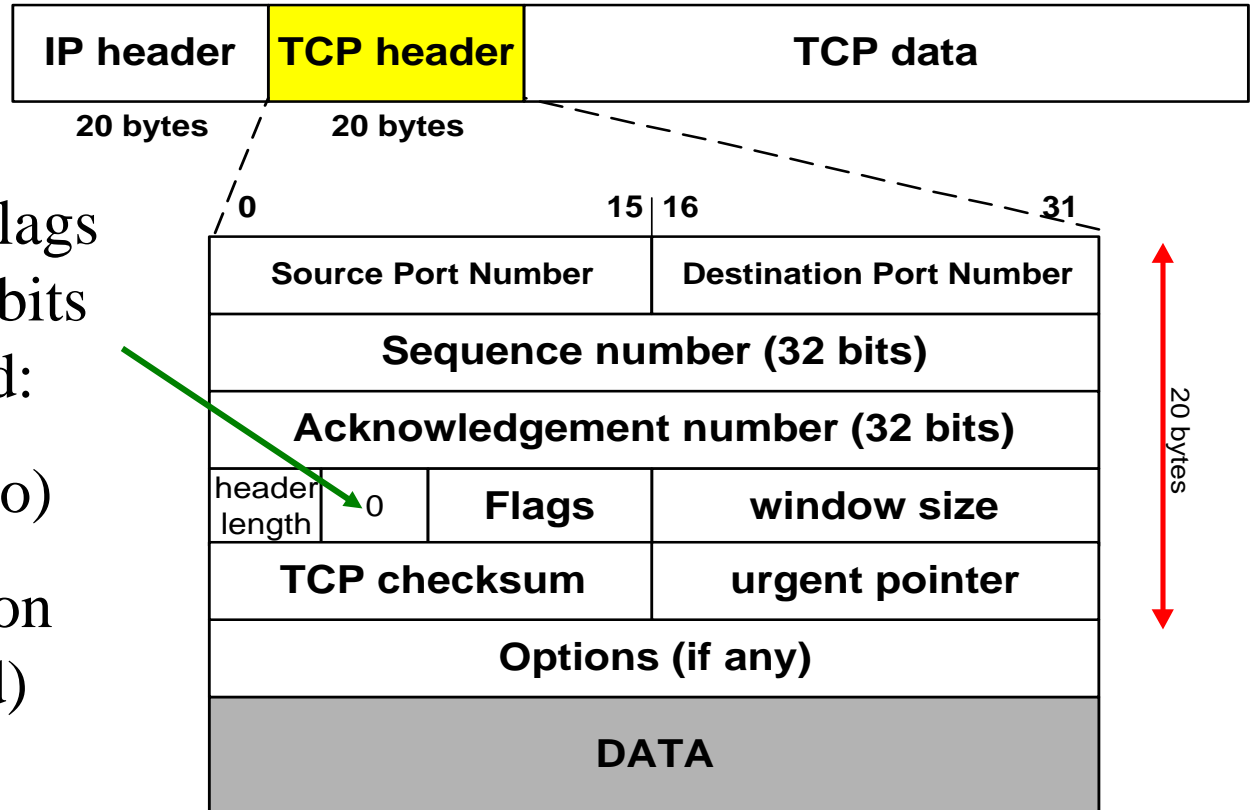
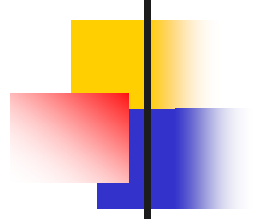


CE (congestion experienced)

bit # 0



Changes in TCP Header

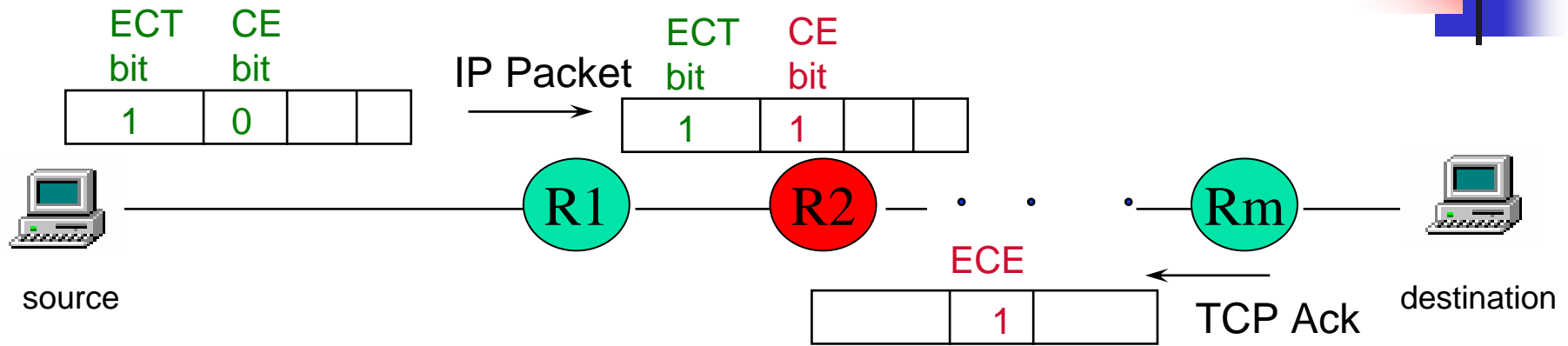


Two new 1-bit flags
in the Reserved bits
of the Flags field:

ECE (ECN-Echo)

CWR (congestion
window reduced)

Implementing ECN at Routers

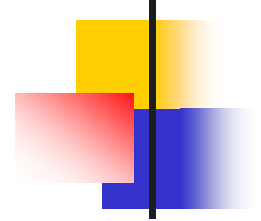


- Following rules for RED, if average queue length is between *minth* and *maxth*, router behaves as follows
 - if ECT (ECN Capable Transport) bit is set for packet, packet's CE bit is set probabilistically
 - if packet's ECT bit is *not* set, packet is dropped probabilistically
- When the average queue length exceeds *maxth*, router drops incoming packet
- When router encounters a packet with its CE bit set, it leaves it unchanged

Implementing ECN at End Hosts



- TCP Source and destination must agree to use ECN at connection setup
- If the two entities agree to use ECN
 - Sender sets ECT bit and clears CE bit in transmitted packets
 - Receiver would examine the CE bit in each received packet. If CE is set, it sends an ACK with ECE bit set. The receiver continues to set the ECE bit in its ACKs until it receives a data packet with the CWR bit set
- When sender detects ECE bit set in an Ack, it behaves just as if a congestion loss was detected
 - Sender does not increase “cwnd”; rather it halves “ssthresh” and reduces congestion window
 - Sender sets CWR bit in the next transmitted packet



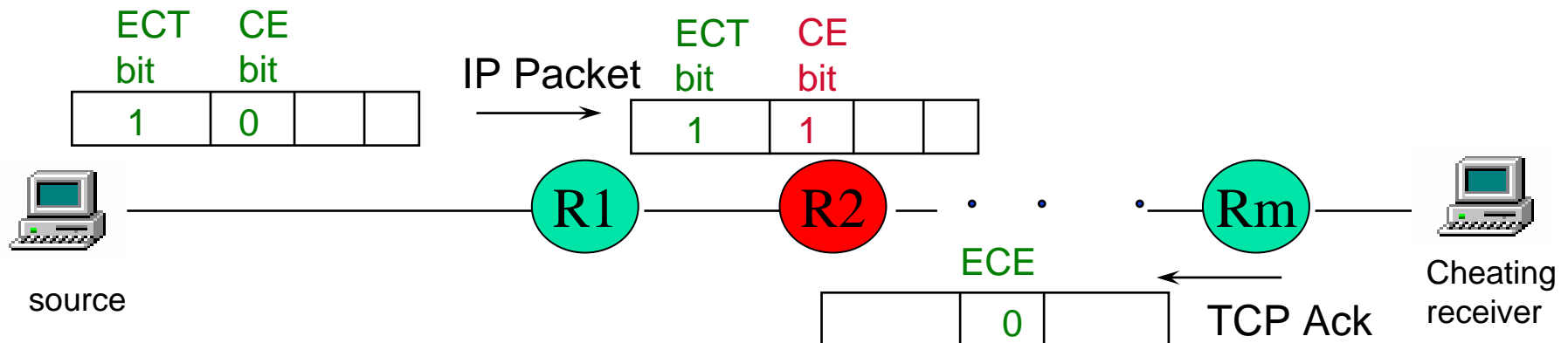
ECN Rules

- TCP should react at most once per RTT to any congestion indication, including
 - timeout of the RTO timer
 - duplicate ACKs
 - ECE ACK
- The above rule means TCP should not react to Explicit Congestion indication more than once every window of data
- TCP should set the CWR flag in the next outgoing packet after it has reduced its congestion window for any reason

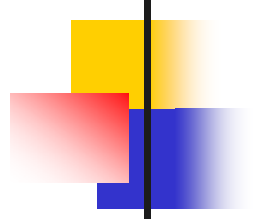
Robust ECN

RFC 3540 (N. Spring, D. Wetherall, and D. Ely) June 2003

- Motivation
 - Cheating receivers can hide ECN marks and still get data
 - Opportunistic receivers can use ACK acceleration
 - In general, misbehaving receivers benefit
- Policy
 - Senders should not trust receivers

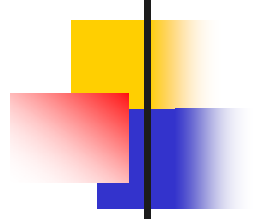


Properties of Robust ECN



- Catches a misbehaving receiver with a high probability, and never implicates an innocent receiver
- Does not change other aspects of ECN, nor does it reduce the benefits of ECN for behaving receivers
- Is simple and requires only one extra bit in the TCP header, called **NS** (Nonce Sum)

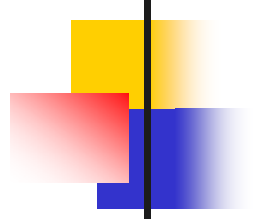
Utilizing ECN Nonces



- The use of two ECN bits in the IP packet headers essentially gives a one-bit ECN nonce as follows

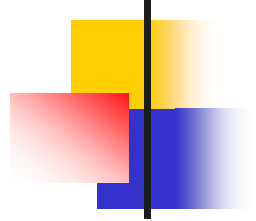
ECT	CE	
0	0	not ECT
0	1	ECT with nonce = 1
1	0	ECT with nonce = 0
1	1	CE

- Sender chooses a random nonce (0 or 1) in the IP header of each transmitted packet
- In each ACK, the receiver sets the NS bit to the value of XOR of all packets received without congestion notification



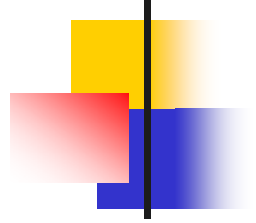
Robust ECN Operation

- Sender verifies the nonce sum returned by the receiver to ensure that congestion indications are not being concealed
- When a congested router marks a packet, it effectively erases the nonce value written by the sender
- Senders have a 50-50 chance of catching a lying receiver whenever an ACK conceals a mark. Because each guess by the receiver is an independent trial, cheaters will be caught quickly if there are repeated congestion signals



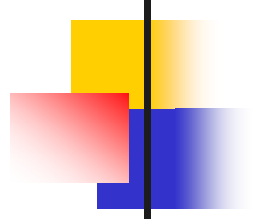
Summary: Robust ECN

- Sender
 - Sender manages the CWR bit
 - Sender places a nonce (0 or 1) on each packet
 - Sender checks the validity of the NS bit returned in the receiver's ACKs
- Receiver
 - Receiver manages the ECE bit
 - Receiver maintains the nonce sum for in-order packets and uses it to set the NS bit in each ACK
- Router
 - Router proactively sets the ECT and CE bits if mildly congested



Summary: Robust ECN (continued)

- Sender should ignore the nonce sum returned on any ACK with ECE bit set
- After sending CWR, the sender re-synchronizes its internal nonce sum by setting it to the NS value of the receiver's ACK
- Sender's response to an incorrect nonce sum
 - Rate limit the connection
 - Disable ECN

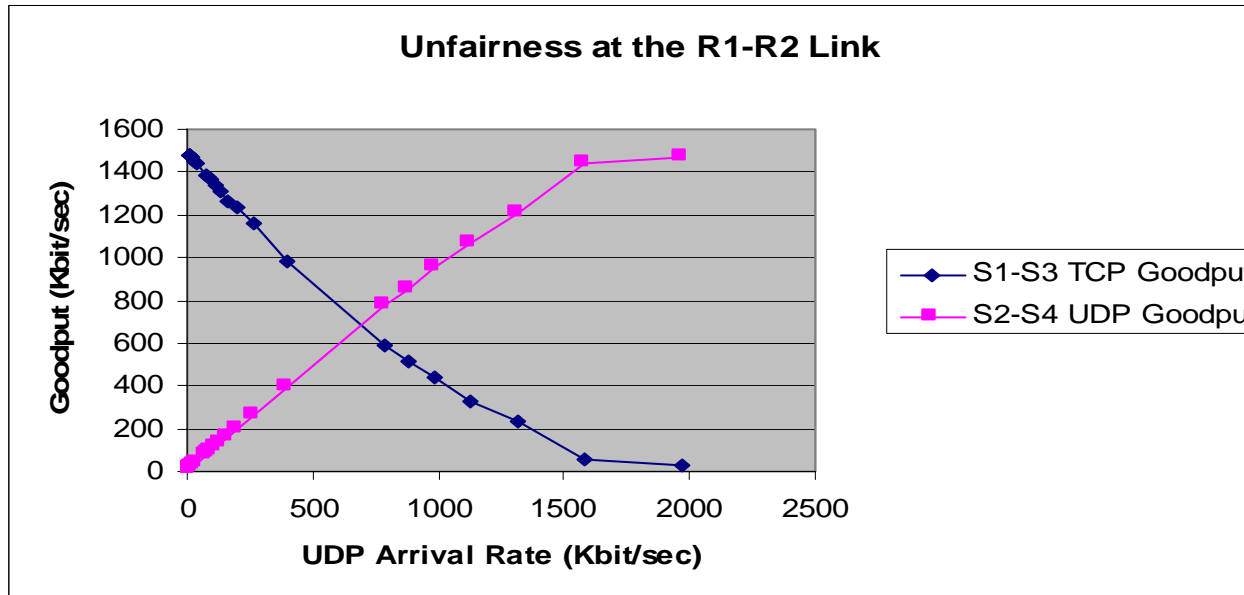
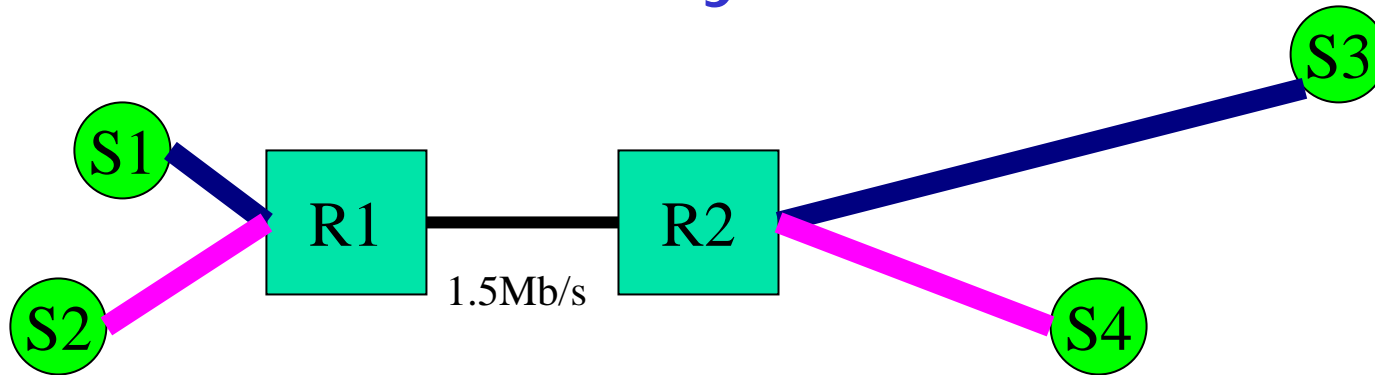


TCP-Friendly Protocols

Motivation

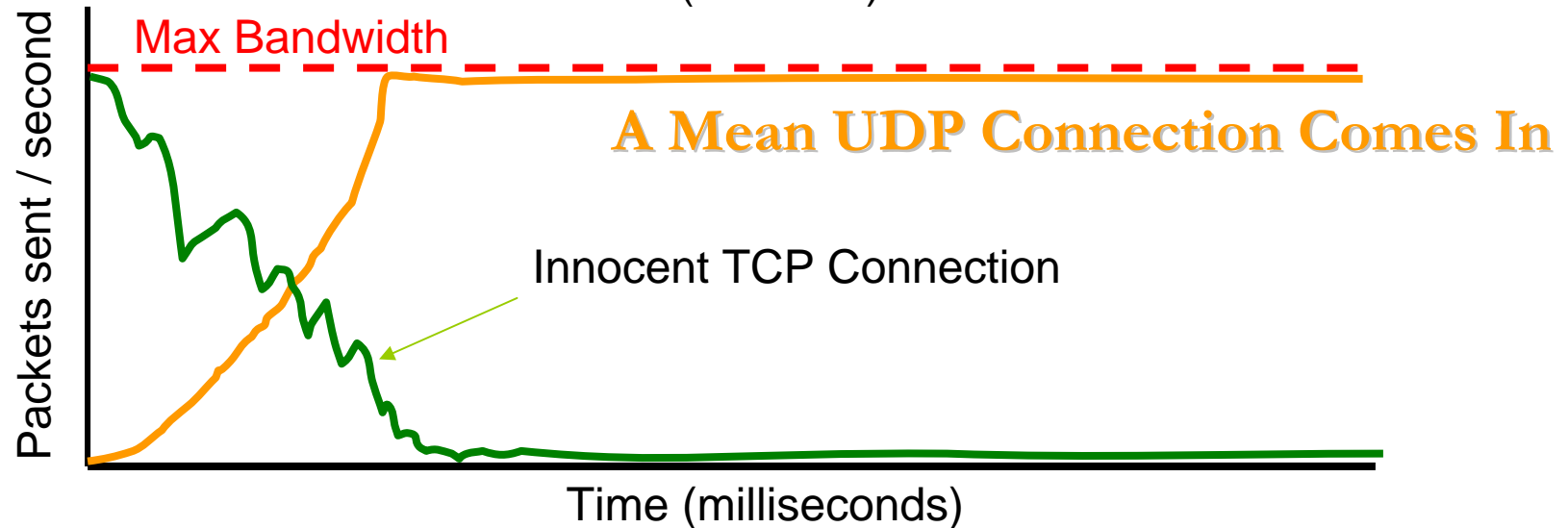
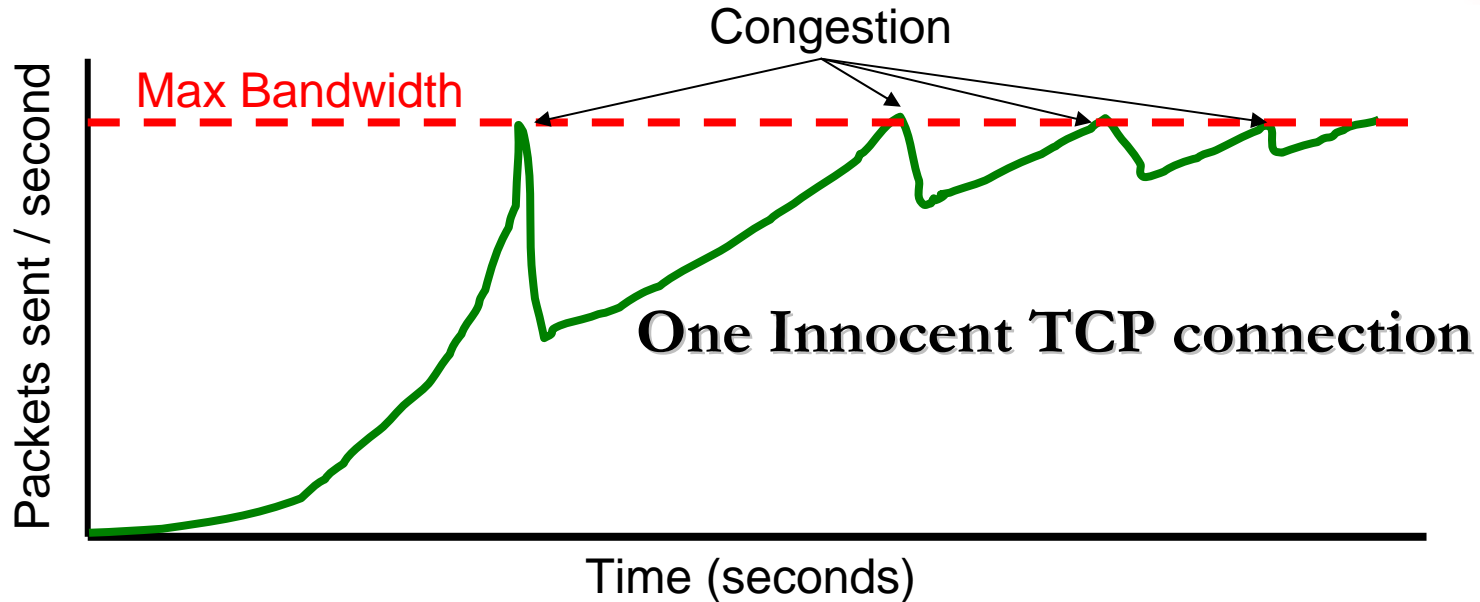
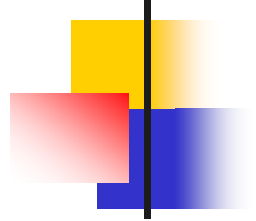
- TCP uses retransmission and is reliable
 - But not good for interactive applications
- UDP has no retransmission
 - It doesn't use retransmission and does not respond to congestion
 - Not reliable but is good for media streaming
- When TCP and UDP flows coexist on the Internet, TCP flows suffer bandwidth loss and congestion collapse

Unfairness Caused by UDP Flows

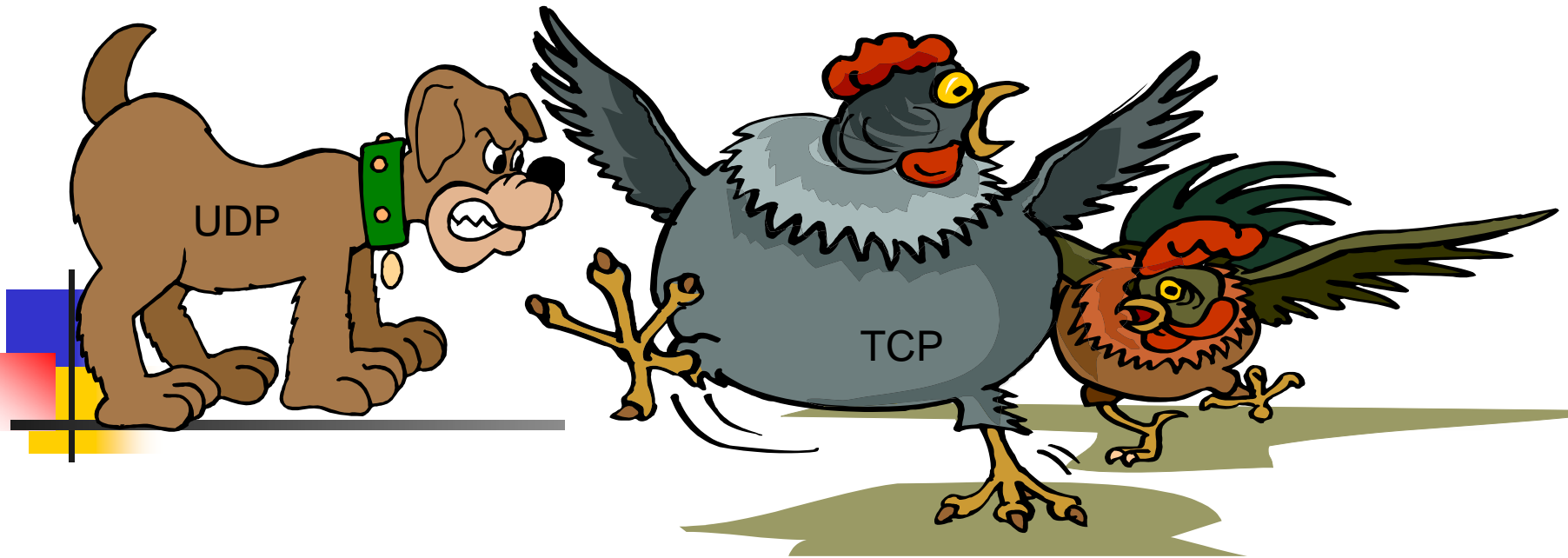


Source: www-nrg.ee.lbl.gov (Kevin Fall and Sally Floyd)

Unfairness Caused by UDP Flows

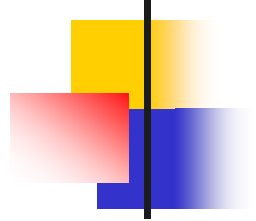


UDP is not TCP-Friendly



Why wasn't this an issue before?

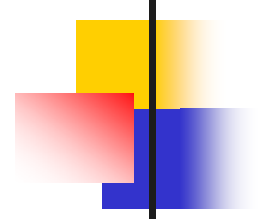
- Lots of unused bandwidth (not anymore)
- Small number of users (not anymore)
- Video streams? Audio Streams? Who'll use it? (Cable/DSL increased the demand)



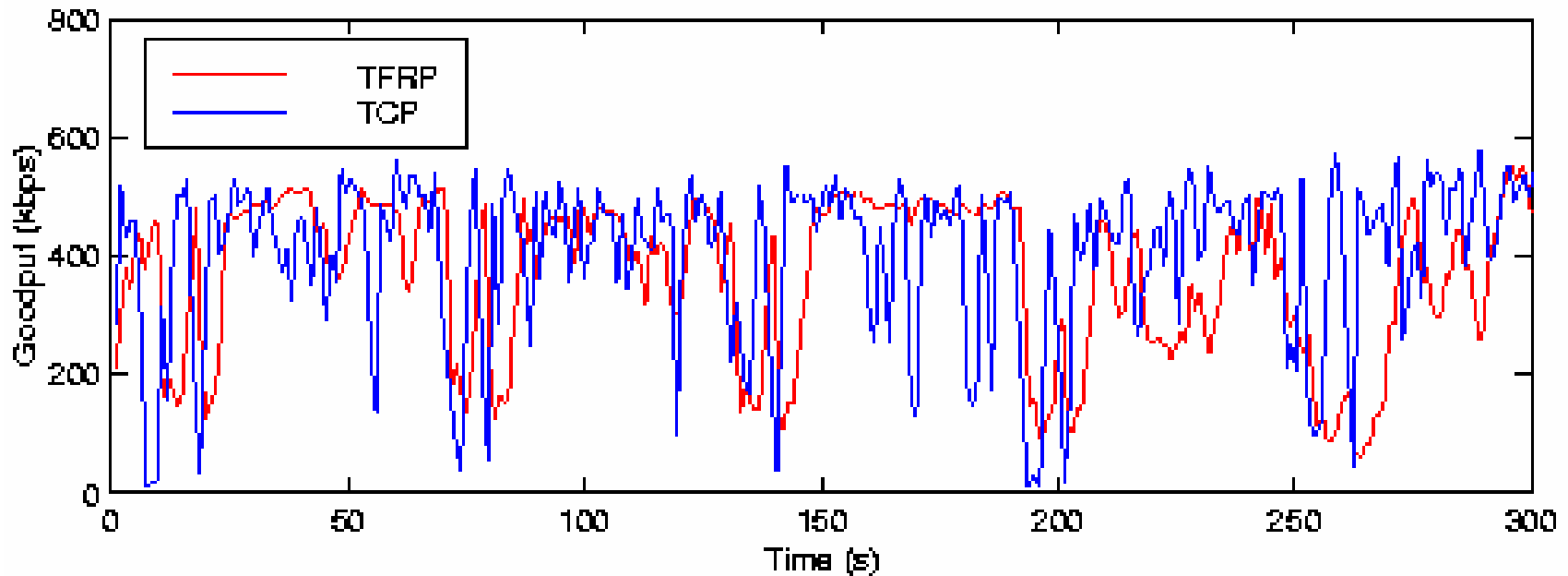
TCP-Friendly Protocols

- Unresponsive flows get unfair share of network bandwidth
- Streaming flows need to be *TCP-Friendly*
- A *TCP-Friendly* flow's bandwidth is no more than a conformant TCP flow running under comparable network conditions
- A “cooperative” TCP-friendly protocol should:
 - share bandwidth fairly with itself
 - share bandwidth fairly with TCP

TCP-Friendliness



- Throughput for concurrent transmission
Toronto to Berkeley: one TCP flow and one
UDP flow with TCP-friendly rate control
protocol



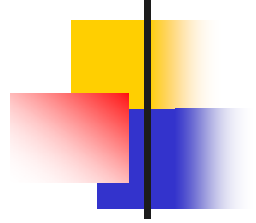
Source: Video and Image Processing Lab- UC Berkeley

Forward Error Correction (FEC)



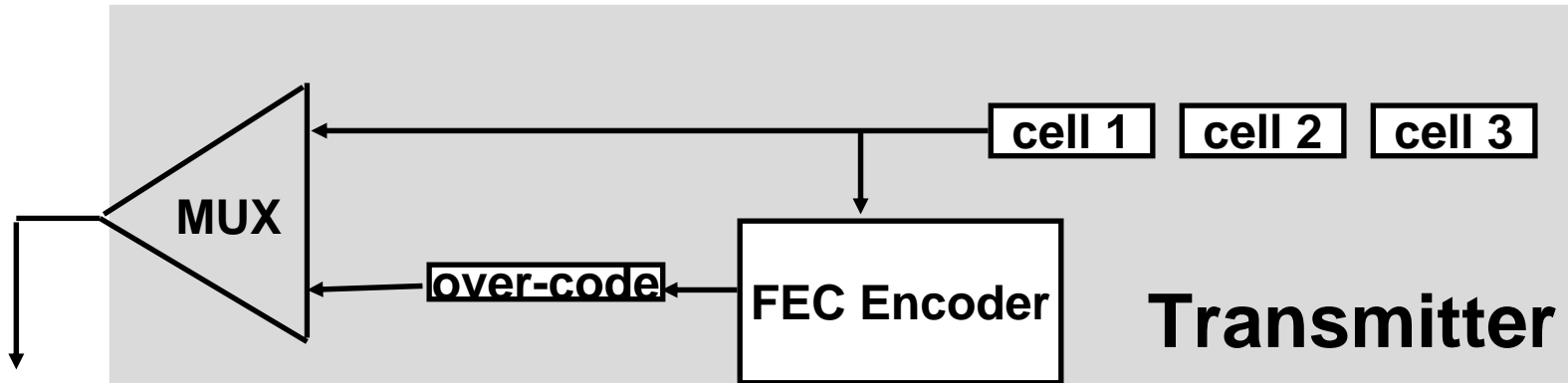
- FEC is a repair scheme that attempts to recover lost packets without incurring the long delay of retransmission
- FEC is suitable for TCP-friendly media streaming
- Example: TCP-Friendly MPEG Video Streaming with FEC repair technique, Worcester Polytechnic Institute, Worcester, MA

Example: FEC in ATM Networks



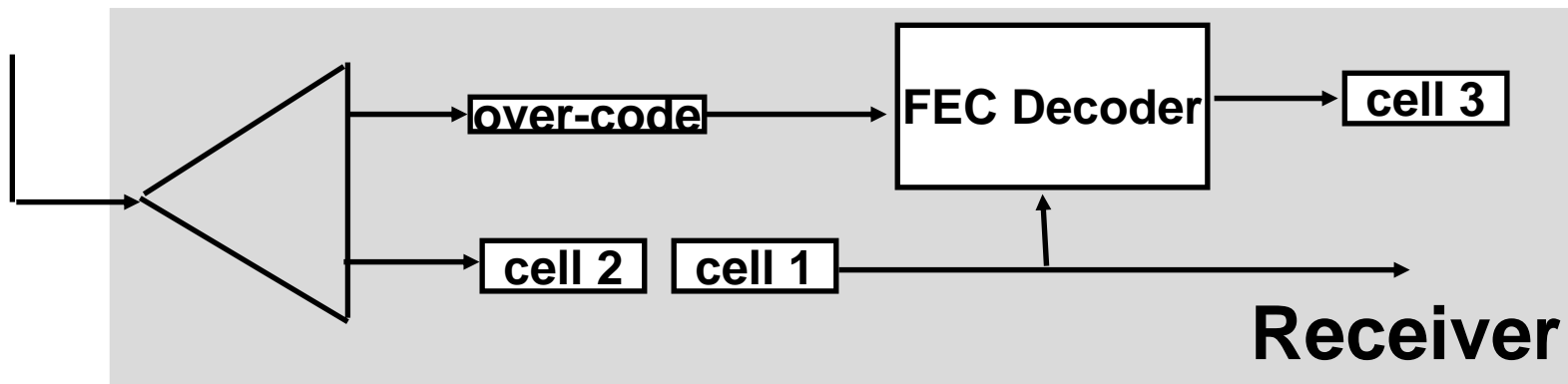
- At the transmitter, the FEC system takes k data cells as input and produces $k+h$ cells as output. This output consists of the k original data cells along with h redundancy cells obtained by performing certain computation on the k data cells
- The k data cells are referred to as the block of data and the h redundant cells are called the over-code. The data block and its over-code are transmitted to the receiver as an expanded block
- The FEC computation is such that any k cells of the expanded block can be used to recover the entire information ($k+h$ cells) of the expanded block
- At the receiver, if at most h cells of the expanded block are missing/lost, the FEC computation can be used to reconstruct any missing data cells

Example: FEC with $k = 3$, $h = 1$

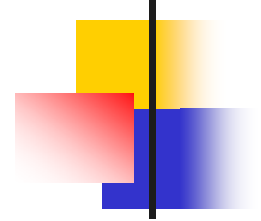


Cell 3
is lost

Network



Example: FEC with $k = 3$ and $h = 1$



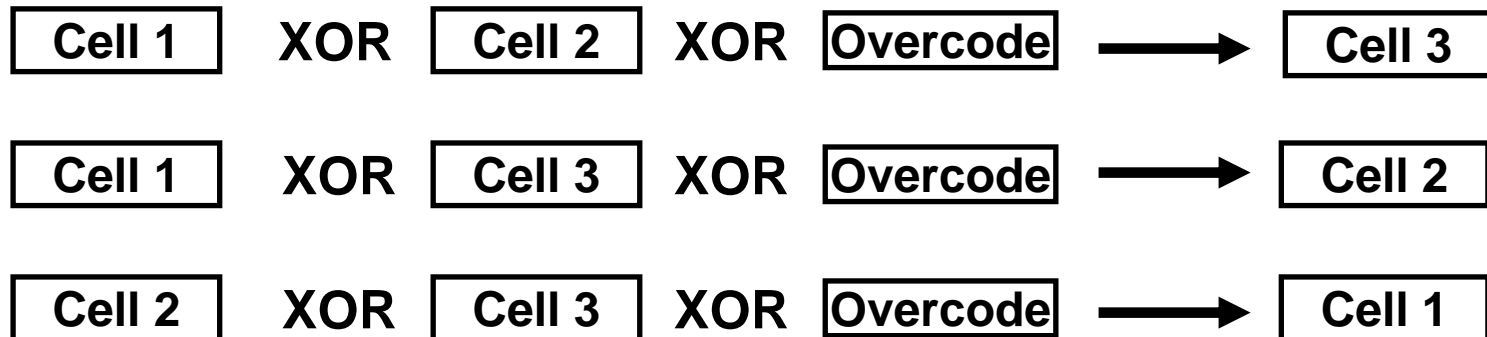
At Transmitter

Cell 1	1010111001
Cell 2	1110101010
Cell 3	0011001101

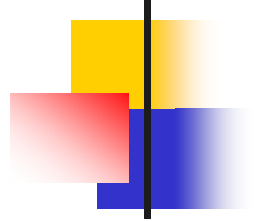
Exclusive OR
(XOR)

Over-code	0111011110
------------------	-------------------

At Receiver



Conclusions



- TCP has been a major factor in the success of the Internet
- RED routers have been introduced to avoid congestion collapse, reduce queue length and avoid global synchronization
- ECN is proposed to avoid retransmission and improve application throughput
- TCP-friendly protocols are needed to cope with the proliferation of media streaming application