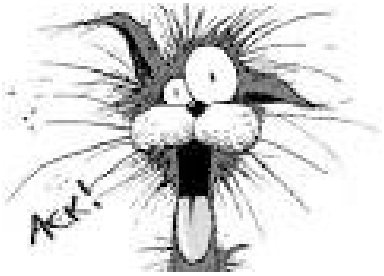
CSCD 330
Network Programming
Spring 2017

Lecture 11a
Transport Layer

Reading: Chapter 3

Who is this?

Bill The Cat



Bloom County was an American comic strip

By Berkeley Breathed which ran from December 8, 1980, until August 6, 1989

It looked at events in politics and culture through viewpoint of small town in Middle America, where children have adult personalities and vocabularies and where animals can talk

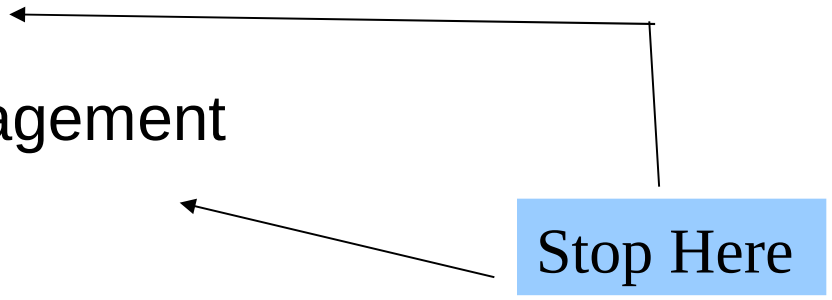
Bill the Cat is a large orange tabby cat, a parody of the comic character Garfield, says little beyond his "Ack" and "Pbthhh"

His persistent near-catatonic state was result of drug use or brain damage resulting from once being legally dead and then revived after too long of a period

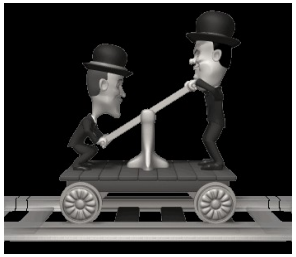
http://en.wikipedia.org/wiki/Bloom_County#

Chapter 3 Sections

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
  - Pipelined vs. Stop and Wait Protocols
- Connection-oriented transport: TCP
  - segment structure
  - reliable data transfer
  - Flow control
  - Connection management



Introduction



- Said that TCP is reliable
  - TCP protocol does many things to manage reliability
    - Acknowledgments,
    - Retransmit lost segments,
    - Timer for retransmit management
  - Look at other features used by TCP
    - Flow Control
    - Congestion Control
  - First, examine Connection Management
    - The States of a TCP connection

Connection Management

TCP Connection Management

- **TCP sender, receiver establish "connection" before exchanging data segments**

- Initialize TCP variables
  - Sequence numbers
  - Buffers,
  - Flow control info (e.g. RcvWindow)

- Client - Connection initiator

```
Socket clientSocket =
        new Socket("hostname",6789);
```

- Server - Contacted by client

```
Socket connectionSocket = welcomeSocket.accept();
```

TCP Connection Management

Opens Connection

Three way handshake

Step 1 Client host sends TCP SYN segment to server, Sets SYN flag

- Specifies initial sequence number
- No data

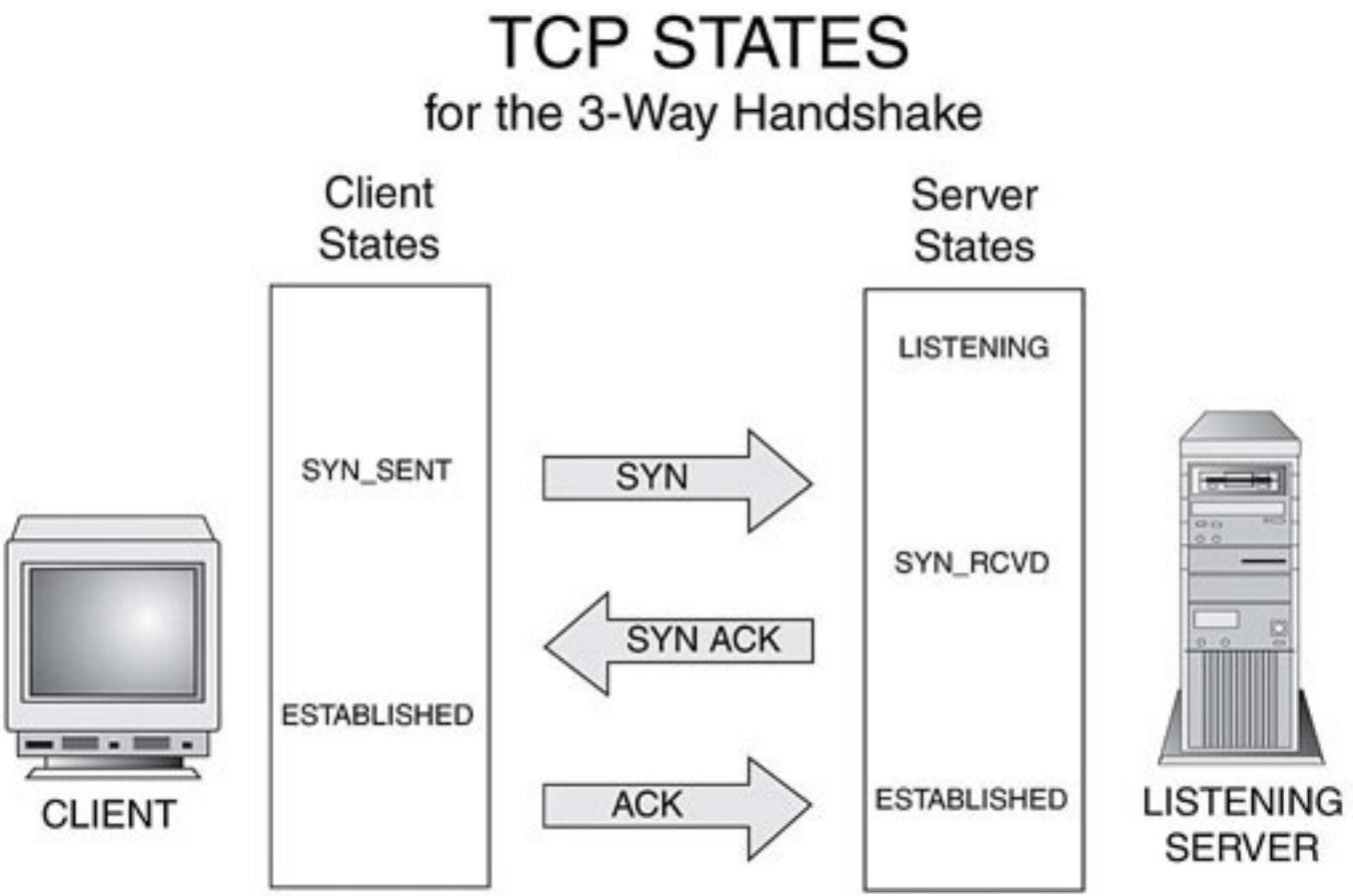
Step 2 Server host receives SYN, replies with SYN/ACK segment

- Server allocates buffers
- Specifies server initial sequence number

Step 3 Client receives SYNACK, replies with ACK segment

- Which may contain data
- Allocates buffers

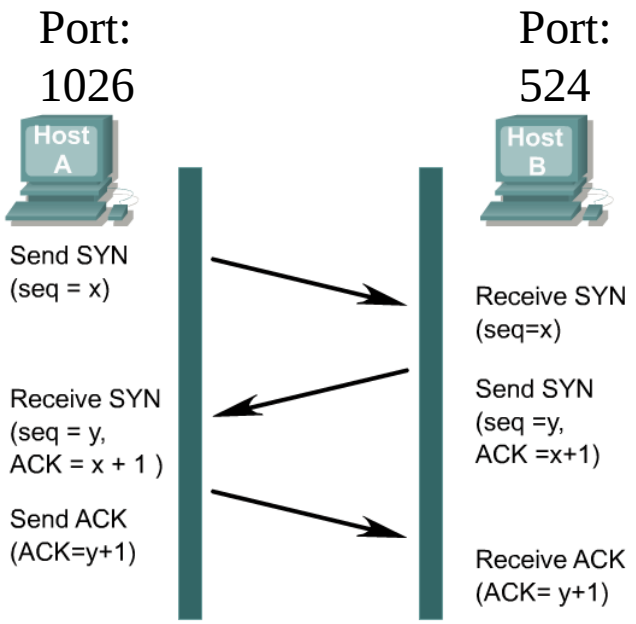
TCP 3-Way Handshake



```
 Packet 1: source: 130.57.20.10  dest.:130.57.20.1
TCP: ----- TCP header -----
      TCP: Source port              = 1026
      TCP: Destination port         = 524
      TCP: Initial sequence number = 12952
      TCP: Next expected Seq number= 12953
      TCP:                .... ..1. = SYN
      TCP: Window                   = 8192
      TCP: Checksum                 = 1303 (correct)
      TCP: Maximum segment size     = 1460 (TCP Option)
```

```
Packet 2: source: 130.57.20.1   dest: 130.57.20.10
TCP: ----- TCP header -----
      TCP: Source port              = 524
      TCP: Destination port         = 1026
      TCP: Initial sequence number = 2744080
      TCP: Next expected Seq number= 2744081
      TCP: Acknowledgment number    = 12953
      TCP:                .... ..1. = SYN
      TCP: Window                   = 32768
      TCP: Checksum                 = D3B7 (correct)
      TCP: Maximum segment size     = 1460 (TCP Option)
```

```
Packet 3: source: 130.57.20.10  dest: 130.57.20.1
TCP: ----- TCP header -----
      TCP: Source port              = 1026
      TCP: Destination port         = 524
      TCP: Sequence number          = 12953
      TCP: Next expected Seq number= 12953
      TCP: Acknowledgment number    = 2744081
      TCP:                ...1 .... = Acknowledgment
      TCP: Window                   = 8760
      TCP: Checksum                 = 493D (correct)
      TCP: No TCP options
```



- Only part of the TCP headers are displayed.

TCP Connection Management

- Closing a Connection
  - While it takes three segments to establish a connection …
    - Takes four to terminate a connection
  - Why do you think it takes four segments?

TCP Connection Management

- Why 4 Segments to Close TCP?
  - **TCP connection full-duplex**
    - Each direction must be shut down independently!!
    - Rule that either end can send FIN when done sending data
    - When TCP receives FIN, notifies application that other end has terminated that direction of data flow
    - FIN is typically result of application issuing close,
      Socket.cose();

TCP Connection Management (cont.)

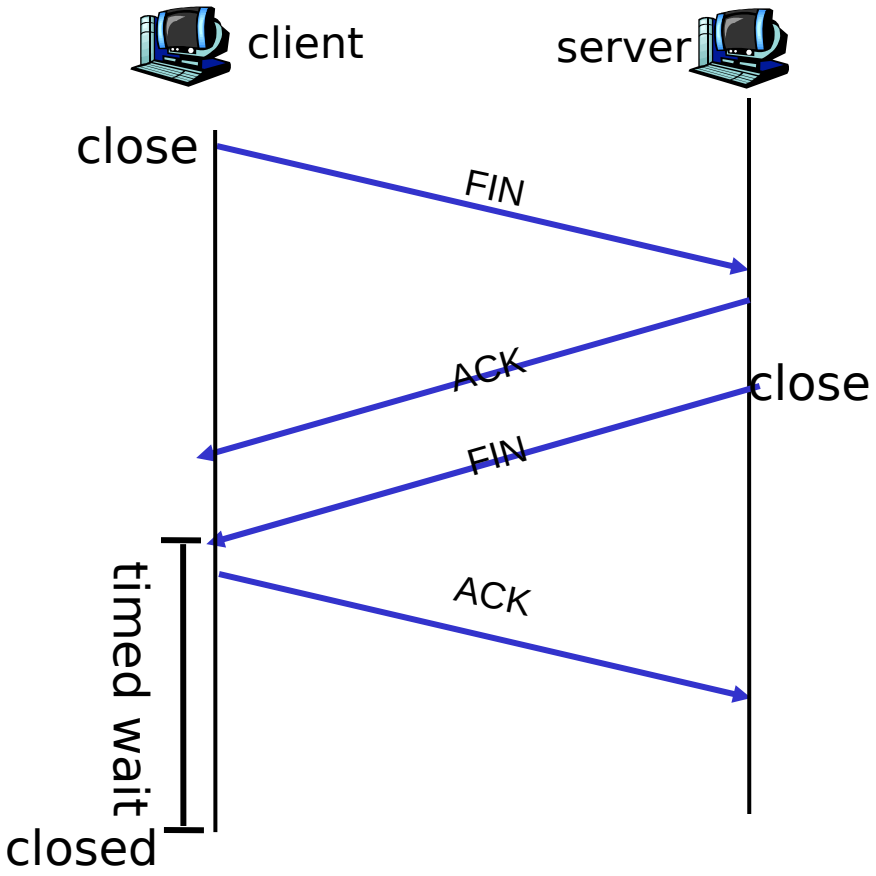
Closing a connection

client closes socket:
clientSocket.**close();**

Step 1: Client end system sends TCP FIN control segment to server

Step 2: Server receives FIN, replies with ACK

Closes connection, sends FIN

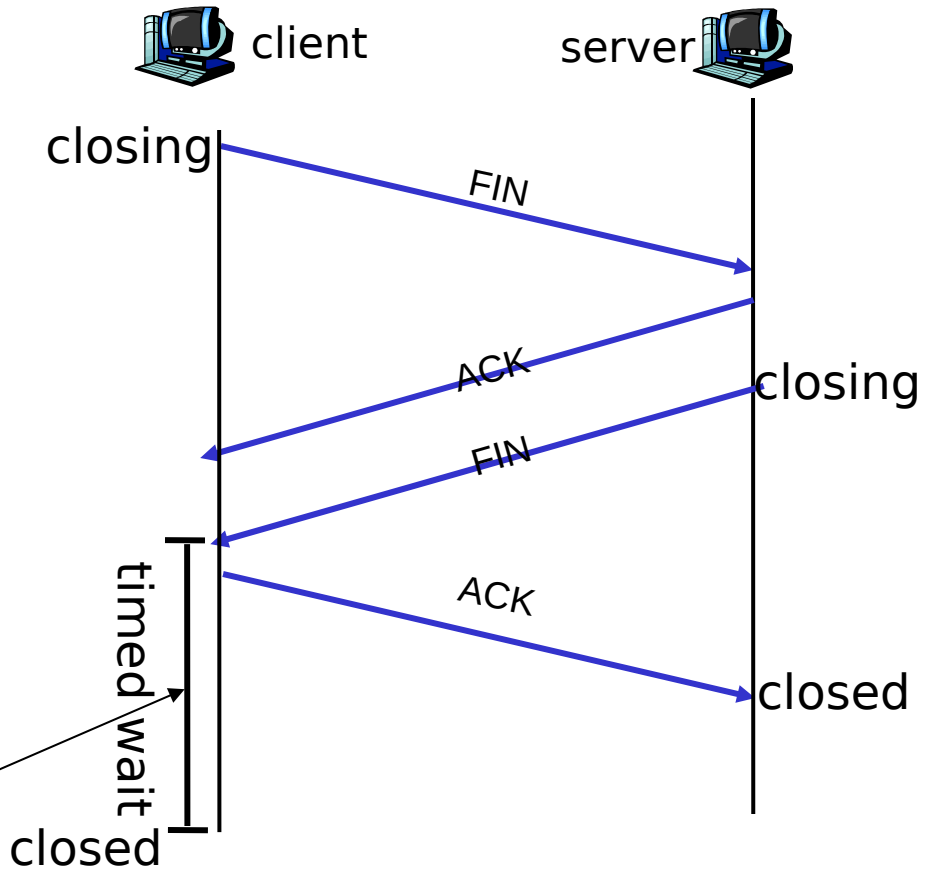


TCP Connection Management (cont.)

Step 3: Client receives FIN, replies with ACK.

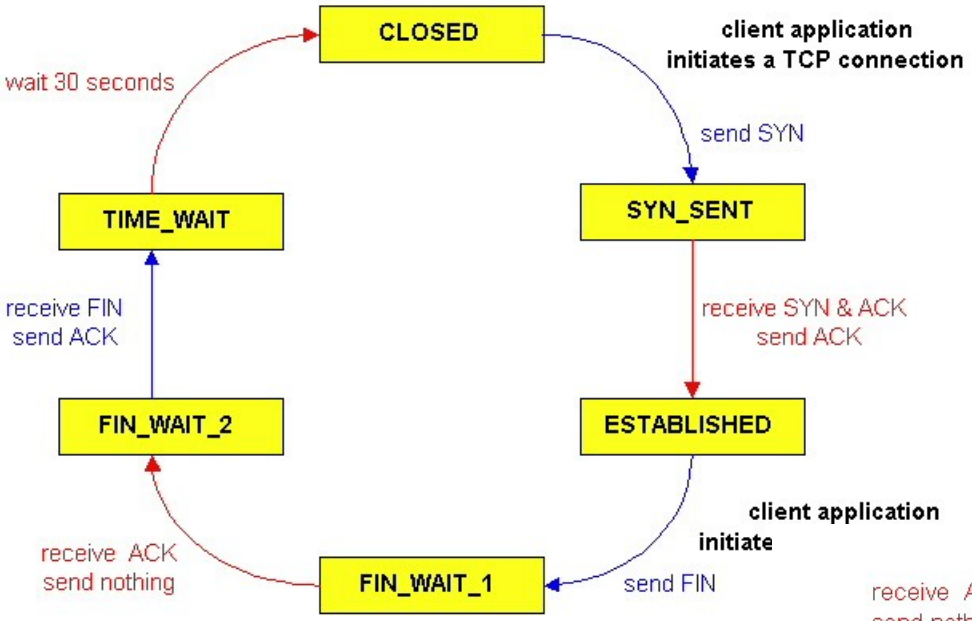
- Enters "timed wait" - will respond with ACK to received FINs

Step 4: Server, receives ACK. Connection closed.



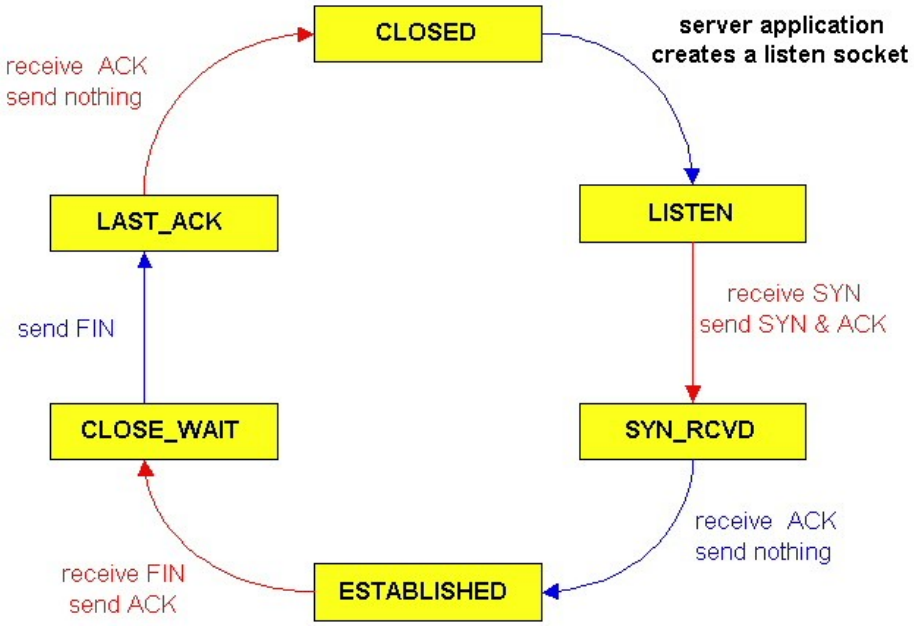
Timed wait is so that make sure ACK truly makes it, or Fin will get resent, connection will not be over!!

TCP Connection Management (cont)



TCP Client lifecycle

TCP Server lifecycle



Wireshark Tutorial on TCP Sequence Numbers and ACK's

- Nice short tutorial plus example file on how TCP manages sequence numbers and acks

  http://packetlife.net/blog/2010/jun/7/understanding-tcp-sequence-acknowledgment-numbers/

Two Types of TCP Performance

- Two issues to consider
  - **Flow control**
    - Coordination between sender/receiver
  - **Congestion control**
    - Behavior altered based on network conditions

- First, look at Flow Control

Buffers at Hosts

- Sending Buffer
  - Maintains data sent but not ACKed
  - Data written by application but not sent
- Receive Buffer
  - Data that arrives out of order
  - Data that is in correct order but not yet read by application

TCP Flow Control

- TCP connection created between Hosts A,B
  - Buffers play important role in conversation between A and B!!!
    - May be slow to read information
    - Not required to be in sync with sender
  - Sender can overflow buffer – send data too fast
  - How to coordinate sender/receiver flow?
- Use something called, Receiver Window

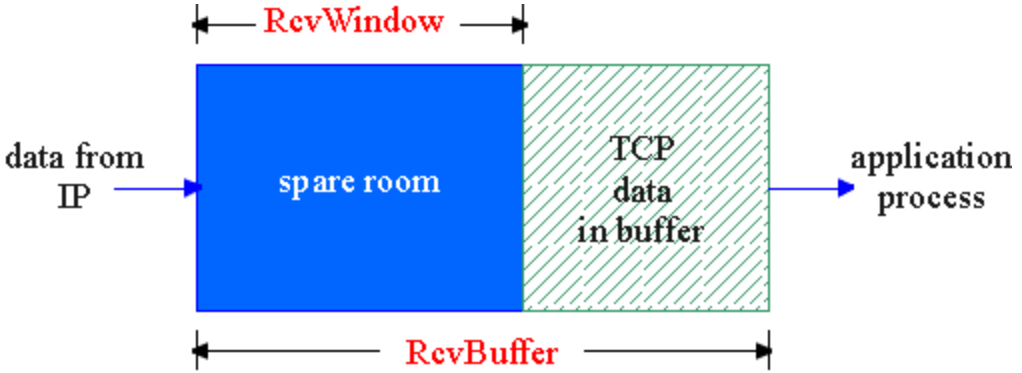
TCP Flow Control

- TCP is a sliding window protocol
  - For window size *n*, can send up to *n* bytes without receiving an Acknowledgment
  - When the data is acknowledged then the window slides forward
- Each packet advertises a window size
  - Indicates number of bytes for which receiver has space

TCP Flow Control

- Receiver Advertises window size to sender based on buffer size allocated for the connection
  - Advertised Window field in TCP header
- Sender cannot have more than
  - Advertised Window bytes of unacknowledged data
- Buffers are of finite size
  - RcvBuffer
  - SendBuffer

Setting the Advertised Window



- On TCP Receiver side,
  - LastByteRcvd -LastByteRead ≤ RcvBuffer
- Thus, it advertises space left in buffer i.e.,

RcvWindow =
RcvBuffer - (LastByteRcvd -LastByteRead)

- As more data arrives i.e., more received bytes than read bytes, LastByteRcvd increases and hence, Advertised Window reduces

Sender Side Response

- At sender side, TCP sender should ensure that
  LastByteSent - LastByteAcked ≤ Advertised Window
- Define what is called “Effective Window”
  which limits amount of data that TCP can send

  Effective Window =
  Advertised Window - (LastByteSent - LastByteAcked)

- In order to prevent overflow of Send Side buffer
  LastByteWritten - LastByteAcked ≤ SendBuffer
  - If application tries to write more, TCP blocks

TCP congestion control

Principles of Congestion Control

Congestion: What causes congestion?

- Informally: “too many sources sending too much data too fast for network to handle”
- Different from flow control!
- Problem is result of many senders/receivers

- Symptoms of Congestion
  - Lost packets (buffer overflow at routers)
  - Long delays (queuing at routers)

Performance Metrics

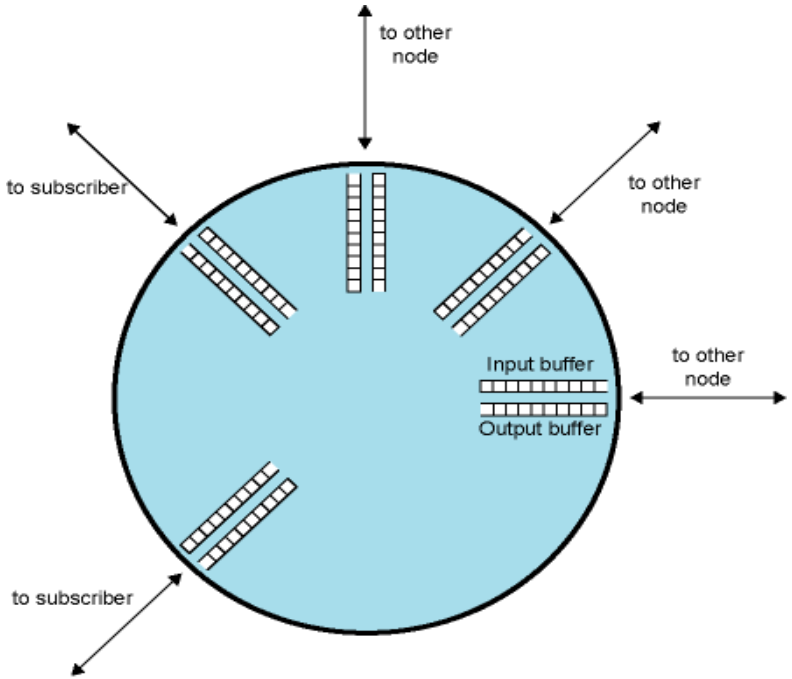
- Two important performance measures
- Throughput
  - Data rate in bps - protocol overhead (trans. delay, propagation, queueing and processing delays)
- Delay
  - Transmission delay
    - Time for transmitter to send all bits of packet
  - Propagation delay
    - Time for one bit to transit from source to destination
  - Processing delay
    - Time required to process packet at source prior to sending, at any intermediate router or switch
- We discussed these. What we did not discuss is
  - Queuing delay: Time spent while waiting in queues

Queuing Delays

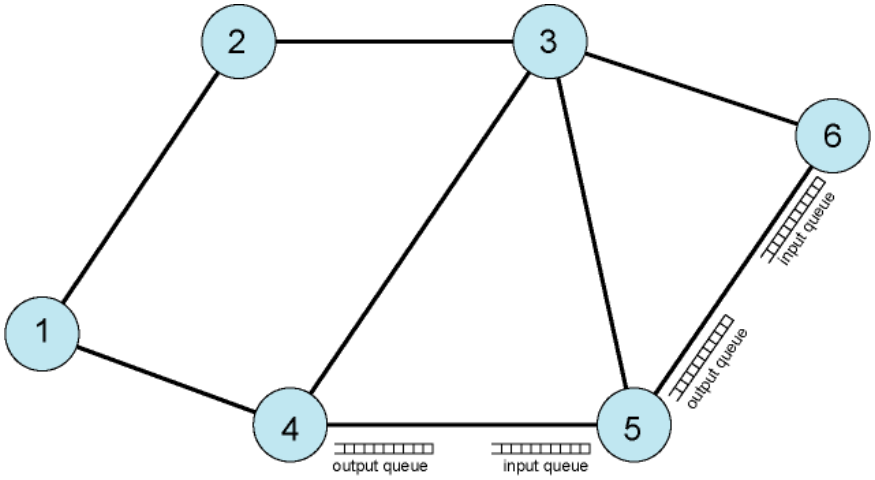
- Queuing delays are significant in performance of communications networks
  - —Grow dramatically as system approaches capacity
- In shared facility
  - – network,
  - – transmission line,
  - – road network,
  - – checkout lines
- Performance worsens exponentially as demand approaches capacity

What Is Congestion?

- Data network is a network of queues
- Congestion occurs when the number of packets being transmitted through the network approaches the packet handling capacity of the network



Input and Output Queues at a Network Node



Congestion is unavoidable

Arguably it's good!

- Packet switching makes efficient use of links
- So, router buffers should be occupied
- If buffers are always empty, delay is low ... but
  - Usage of network is low too
- If buffers are always occupied, delay is high
  - Using network more efficiently

- Goal is manage congestion through our protocols!!!

Response to Congestion

- Most transmission protocols use timeout and retransmission
- Respond to increased delay by retransmitting datagrams, increasing congestion
- This leads to ***congestion collapse***
- So, TCP must reduce transmission rates when congestion occurs

Options for Network Congestion Control

1. Implemented by host versus network
   - Endpoints deal with congestion vs. routers
2. Reservation-based, versus feedback-based
   - Reservation based is to reserve bandwidth as in Circuit Switched Networks
   - Feedback allows adaptation as conditions change
2. Window-based versus rate-based
   - Window sets an amount for segments in flight
   - Rate based negotiates a rate at time of connection

TCP Congestion Control

- TCP implements
  - Host-based,
  - Feedback-based,
  - Window-based

TCP

Congestion control

- TCP sources attempt to determine how much capacity is available
- TCP sends packets, then reacts to observable events (loss)

TCP Congestion Control

- TCP doesn't rely on IP layer to provide congestion control
- Limits rate at which senders can send traffic using perceived congestion on path

- Questions:
  - How to detect congestion?
  - How to limit congestion?
  - What algorithm to use to limit traffic?

TCP Congestion Control

- Old Days ...
  - Old TCPs would start a connection with sender injecting multiple segments
    - Up to window size advertised by receiver
    - No real congestion control, only flow
  - While this is OK when two hosts on same LAN,
    - Could be many routers and slow links between sender and receiver … so
    - Intermediate router(s) packet queues run out of space

TCP Congestion Control

Avoid!!!!

- Main Goal of TCP Congestion Control
  - Avoidance!!!
  - Try to detect congestion and reduce load
  - Because once congestion happens
    - Happens at exponential rate
    - Then, takes a long time for queues to drain
  - Want to keep network pipes full but without danger of saturation

TCP Congestion Control

- Overall Strategy - simple
  - Good times, no congestion
    -> Increase sending rate

  - Bad times, congestion
    -> Decrease sending rate

- Details ... a little more complex

TCP Congestion Control

How does sender detect congestion?

- Concrete evidence …
  - Loss event
    - Timeout, or
    - 3 duplicate ACK's

ACK ACK ACK

- What could you conclude about congestion if you get back acks?
- Acks getting back, conclude that congestion is not extreme … network not completely deadlocked

TCP Congestion Control

How does sender limit congestion?

- TCP sender reduces rate after loss event
  - TCP has one more variable – **CongWin**
  - Amount of unACKed data sender can send
  - Separate from **RcvWin** – flow control variable

  - What does it do?
  - Imposes constraint on rate **sender** sends traffic into network
  - Think of the Congestion Window like this ...

TCP Congestion Control

How does sender limit congestion?

- The Congestion window is flow control imposed by the **sender**
- While Receiver window is flow control imposed by the **receiver**

- **Congestion Window** based on sender's
  - Assessment of perceived network congestion
- **Receiver Window** is related
  - Amount of available buffer space at the receiver for this connection.

TCP Congestion Control

- Essential strategy
- TCP host sends packets into network without reservation and then host reacts to observable events.
  - Originally TCP assumed FIFO queuing
- Basic idea Each source determines how much capacity is available to a flow
- ACKs are used to ***'pace'*** transmission of packets such that TCP is "**self-clocking**"

TCP Congestion Control

- What algorithm(s) to use to limit traffic? 3 of them
  - **AIMD** – Additive Increase Multiplicative Decrease
  - **Slow start** (SS)
  - **Congestion Avoidance** (CA) - After timeout events

- Basically,
  - Algorithms allow TCP to probe bandwidth
  - Increases rate in response to ACK's until loss occurs
  - Decreases rate strategically

TCP Congestion Control: details

- Adds one more constraint based on congestion
- So, sender limited by **Either** Congestion or Flow
- Sender limits transmission:
  LastByteSent-LastByteAcked < min (CongWin,RcvWin)

- Roughly, if we ignore RcvWin …

$$\text{SendRate} = \frac{\text{CongWin}}{\text{RTT}} \text{ Bytes/sec}$$

- **CongWin** is dynamic, function of perceived network congestion
  - Set to 1 MSS at beginning of connection

TCP Congestion Control

- Say, detect a loss (Lost) event
- Basic Idea
  - TCP Sender decreases send rate by decreasing CongWin size
  - Question is, by how much?

- Multiplicative Inverse
  - Half the current value of CongWin size

Example

CongWin

20 Kb -> 10 Kb -> 5 Kb

1 loss 2 loss

Can't go lower than 1 MSS

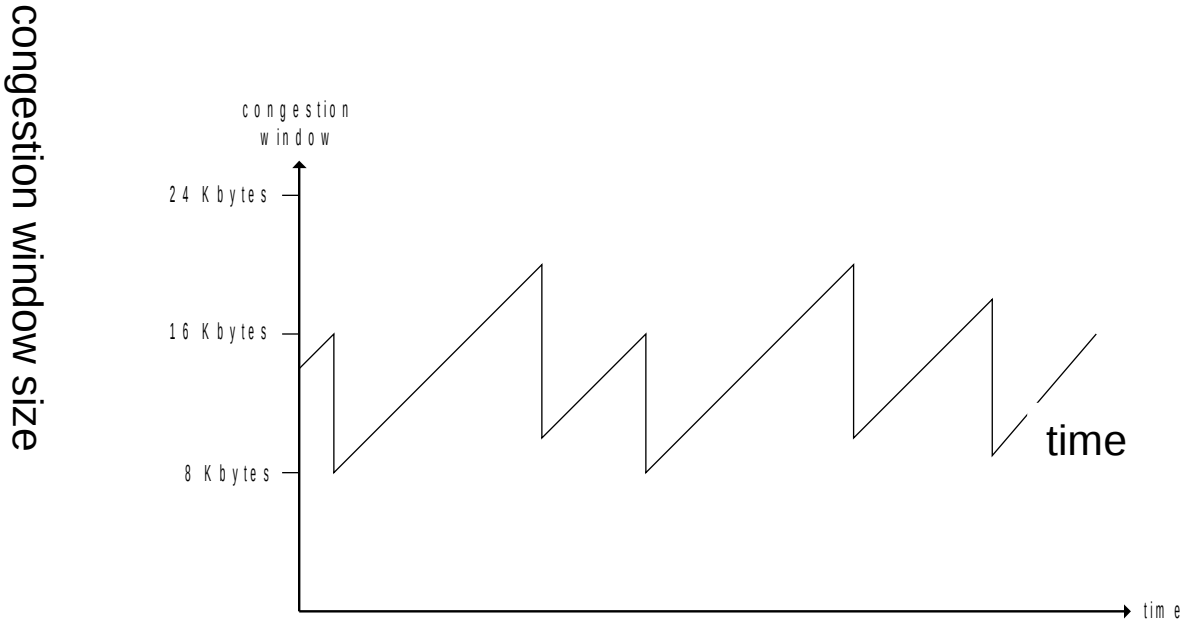
TCP Congestion Control

- But, if no Detected Loss
  - TCP increases its CongWin by 1 MSS
  - **Additive Increase**
    - Cautious increase since its probing for additional bandwidth
    - Assume there is unused bandwidth, could possibly take advantage of
    - Does this every RTT !!!!!
- Thus, TCP
  - Decreases its rate multiplicatively if congestion
  - Increases its rate additively when no congestion

TCP Congestion Control: Additive Increase, Multiplicative Decrease

- Approach Increase transmission rate (window size), probing for usable bandwidth, until loss occurs
  - Additive increase: Increase **CongWin** by **1 MSS** every RTT until loss detected
  - Multiplicative decrease: Cut **CongWin in half** after loss

Saw tooth behavior: **probing** for bandwidth



TCP Slow Start

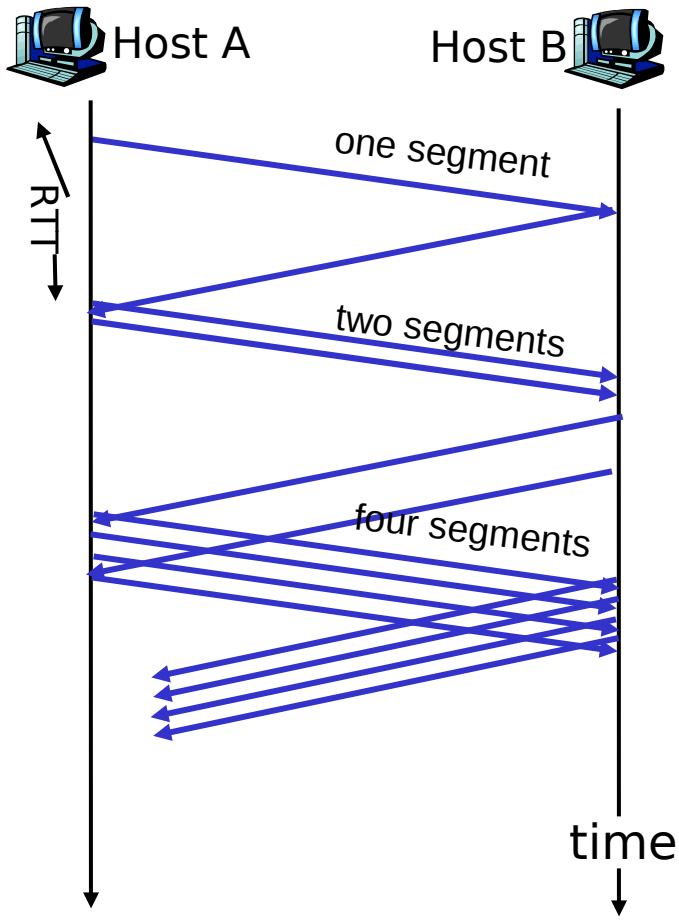
- However, that wasn't fast enough ….

- When Connection Begins, **CongWin** = 1 MSS
  - Example: MSS = 500 bytes & RTT = 200 msec
  - **Initial rate = MSS/RTT or roughly 20 kbps**

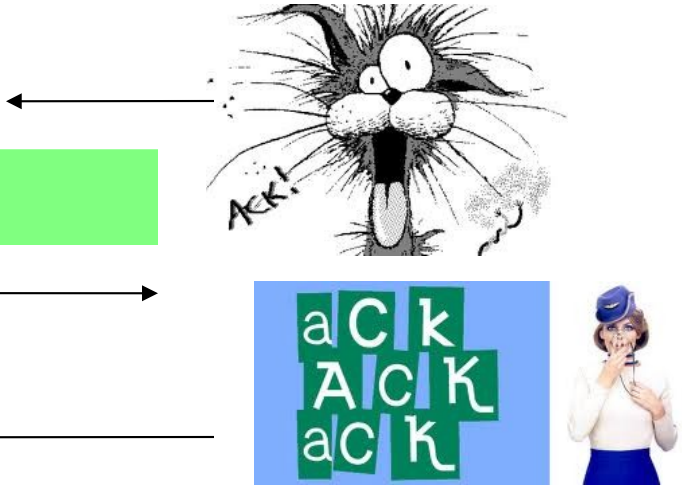
- Available bandwidth may be **more than** MSS/RTT
  - Want to quickly ramp up to respectable rate
  - Or, waste bandwidth
- So … when connection begins
  - Increase rate exponentially fast until first loss event

TCP Slow Start

- When connection begins, increase rate exponentially until first loss event:
  - Double **CongWin** every RTT
  - Done by incrementing **CongWin** for every ACK received
- Summary: Initial rate is slow but ramps up exponentially fast



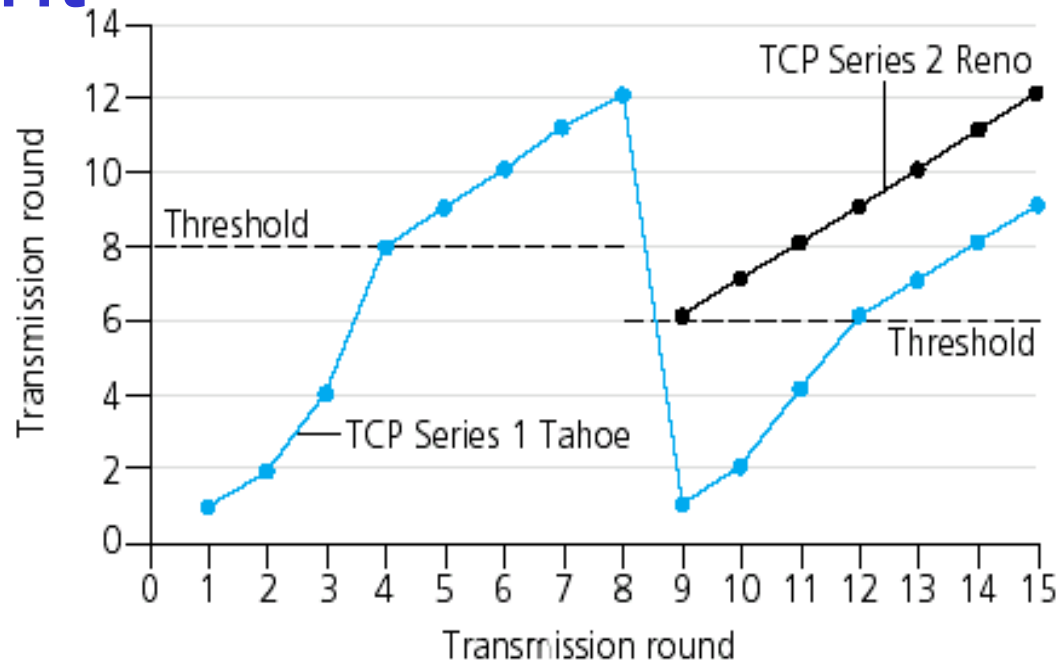
TCP Slow Start



- Philosphy Behind Slow Start ....
- It operates by
- Observing that rate at which new packets should be injected into network
- **Is** rate at which acknowledgments are returned by other end

Refinement

"Reno"



Implementation

- Yet another variable, Threshold
- Window size where slow start ends and linear increase begins, Congestion Avoidance
- At loss event, Threshold is set to 1/2 of CongWin just before loss event (Reno version)

Major TCP Variants

- TCP Tahoe (older)
  - Assumed congestion signals = Lost segments
  - Losses due to packet corruption less frequent than buffer overflows on routers
  - Resorts back to Slow Start, until threshold where it does Congestion Avoidance, increase by 1 MSS
- TCP Reno
  - Changed way it reacts to a loss
  - If sender is still getting Acks, congestion is not so heavy, so sender can still send, since flow still exists
  - But, send should send with less vigor
  - So, does not fall back to slow start

More Refinement: Inferring loss

- After 3 dup ACKs:
  - **CongWin** is cut in half
  - Window then grows linearly
- But after timeout event:
  - **CongWin** instead set to 1 MSS;
  - Window then grows exponentially
  - To a threshold, then grows linearly

Philosophy:

- 3 dup ACKs indicates network capable of delivering some segments
- Timeout indicates a "more alarming" congestion scenario

Summary: TCP Congestion Control

- When **CongWin** is below **Threshold**, sender in slow-start phase, window grows exponentially
- When **CongWin** is above **Threshold**, sender is in congestion-avoidance phase, window grows linearly
- When a triple duplicate ACK occurs, **Threshold** set to **CongWin/2** and **CongWin** set to **Threshold**
- When timeout occurs, **Threshold** set to **CongWin/2** and **CongWin** is set to 1 MSS

Many TCP ‘flavors’

- **TCP New Reno** – Variation of Reno,
- **TCP Vegas**
  - Adjusts window size based on difference between expected and actual RTT
- **TCP Binary Increase Congestion** (**BIC)**
  - Uses optimized congestion control algorithm for high speed networks with high latency
- **TCP Cubic**
  - TCP Cubic allows very fast window expansion however, it also makes attempts to slow the growth of cwnd sharply as cwnd approaches the current network ceiling, it is the default in Linux

Nice overview of newer TCP Congestion Algorithms
http://intronetworks.cs.luc.edu/current/html/newtcps.html#tcp-cubic

TCP Sender Congestion Control

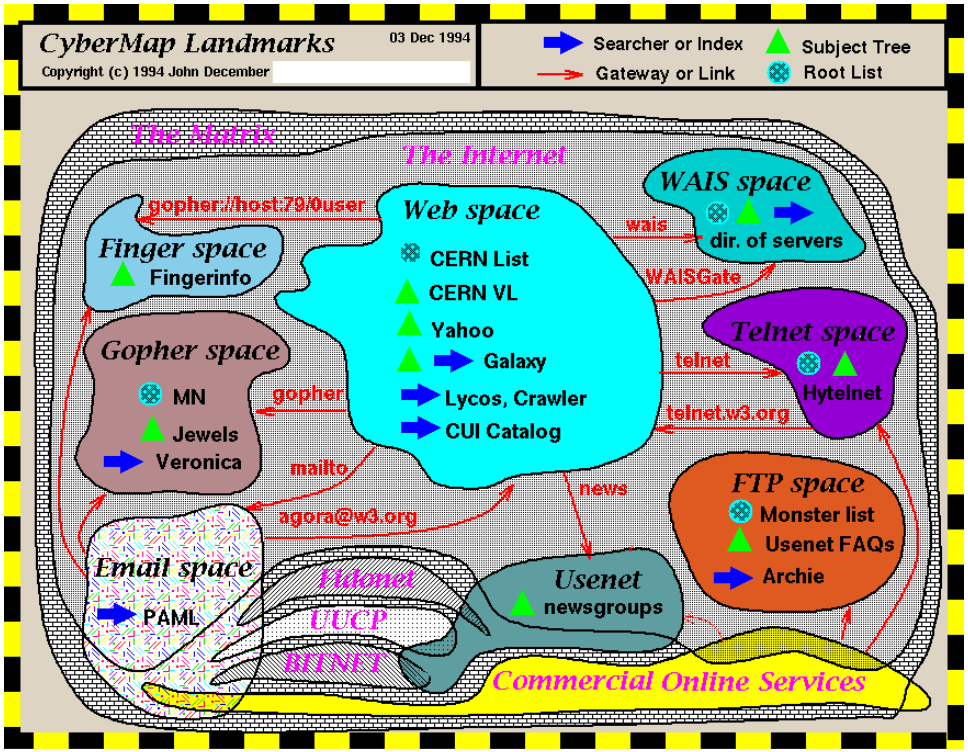
State	Event	TCP Sender Action	Commentary
Slow Start (SS)	ACK receipt for previously unacked data	CongWin = CongWin + MSS, If (CongWin > Threshold) set state to "Congestion Avoidance"	Resulting in a doubling of CongWin every RTT
Congestion Avoidance (CA)	ACK receipt for previously unacked data	CongWin = CongWin+MSS * (MSS/CongWin)	Additive increase, resulting in increase of CongWin by 1 MSS every RTT
SS or CA	Loss event detected by triple duplicate ACK	Threshold = CongWin/2, CongWin = Threshold, Set state to "Congestion Avoidance"	Fast recovery, implementing multiplicative decrease. CongWin will not drop below 1 MSS.
SS or CA	Timeout	Threshold = CongWin/2, CongWin = 1 MSS, Set state to "Slow Start"	Enter slow start
SS or CA	Duplicate ACK	Increment duplicate ACK count for segment being acked	CongWin and Threshold not changed

Summary

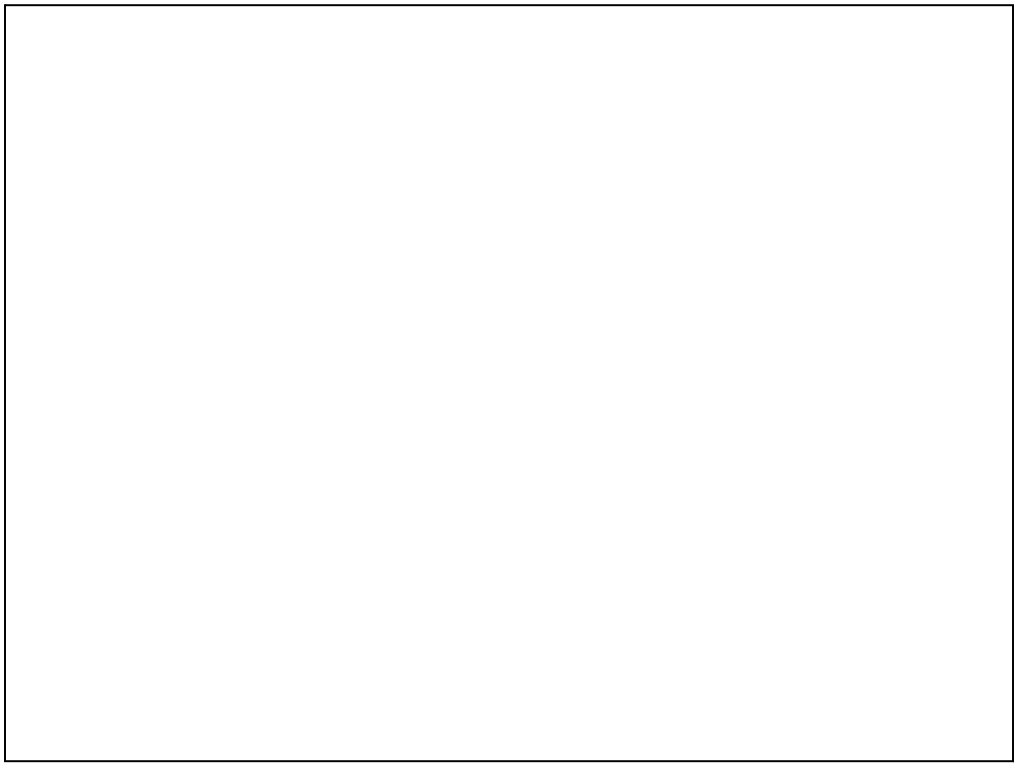
- TCP interacts with routers and reacts to implicit congestion notification,
  - Packet drop
- By reducing TCP sender's congestion window
- TCP increases congestion window using slow start or congestion avoidance
- Does this for every user in the network

Currently, there are a variety of newer TCP algorithms for congestion control

Each has slightly different attributes for optimal congestion control



Assignment due Wednesday - WebServer



Bill The Cat

Bloom County was an American comic strip

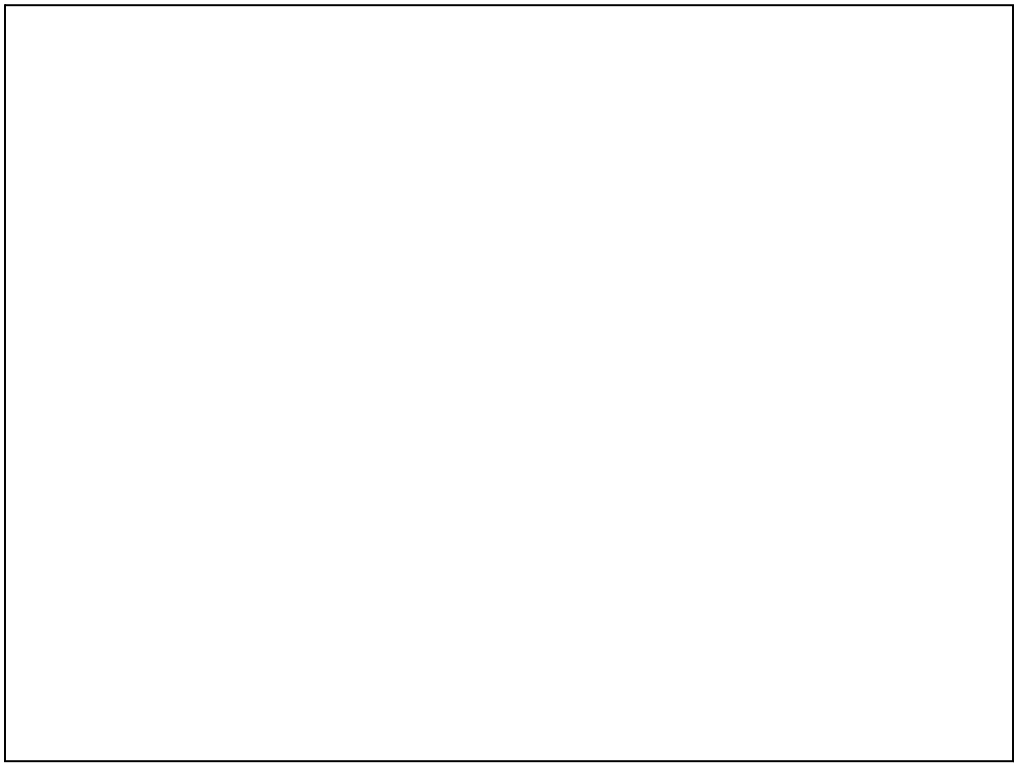
By Berkeley Breathed which ran from December 8, 1980, until August 6, 1989

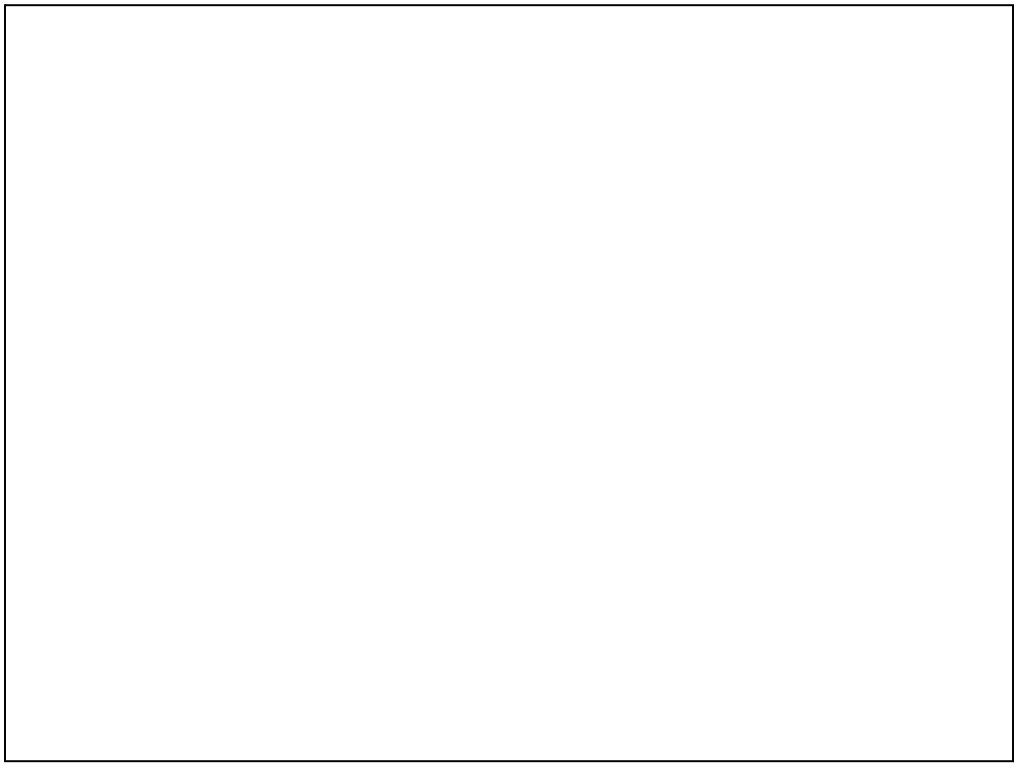
It looked at events in politics and culture through viewpoint of small town in Middle America, where children have adult personalities and vocabularies and where animals can talk

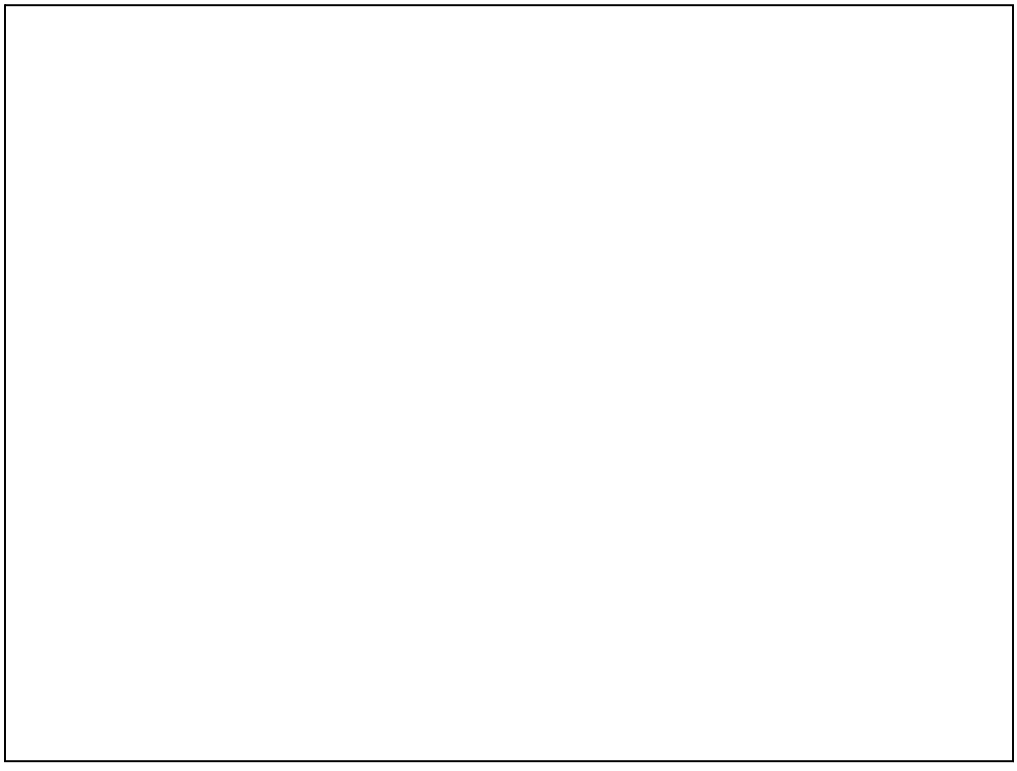
Bill the Cat is a large orange tabby cat, a parody of the comic character Garfield, says little beyond his "Ack" and "Pbthhh"

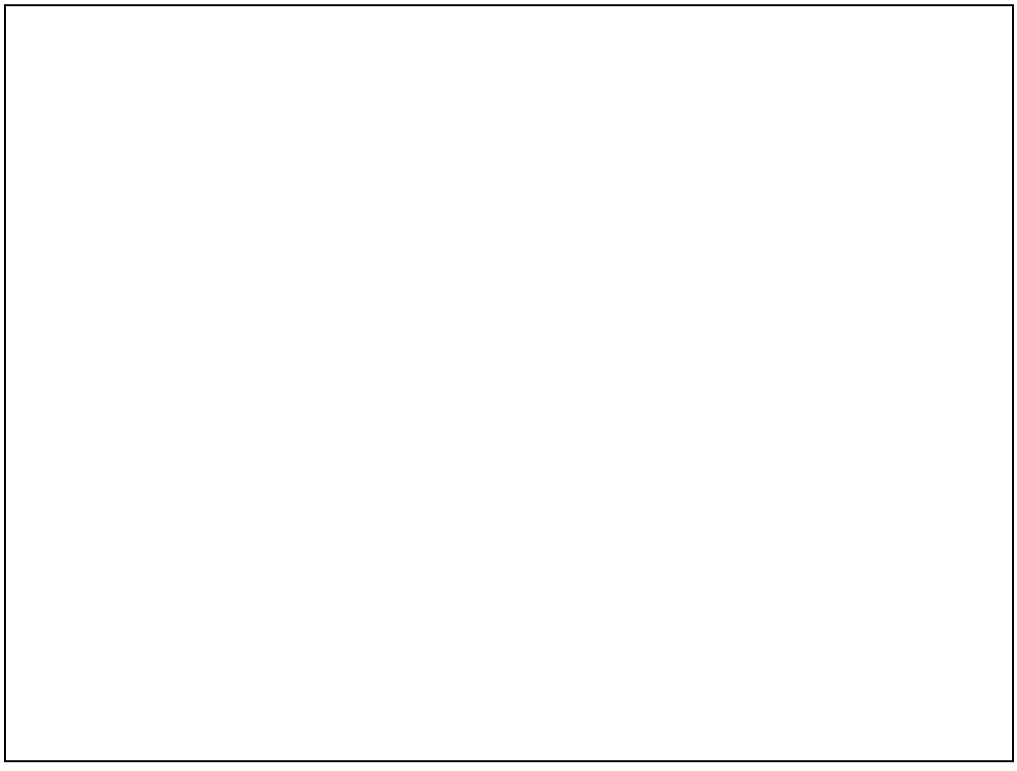
His persistent near-catatonic state was result of drug use or brain damage resulting from once being legally dead and then revived after too long of a period

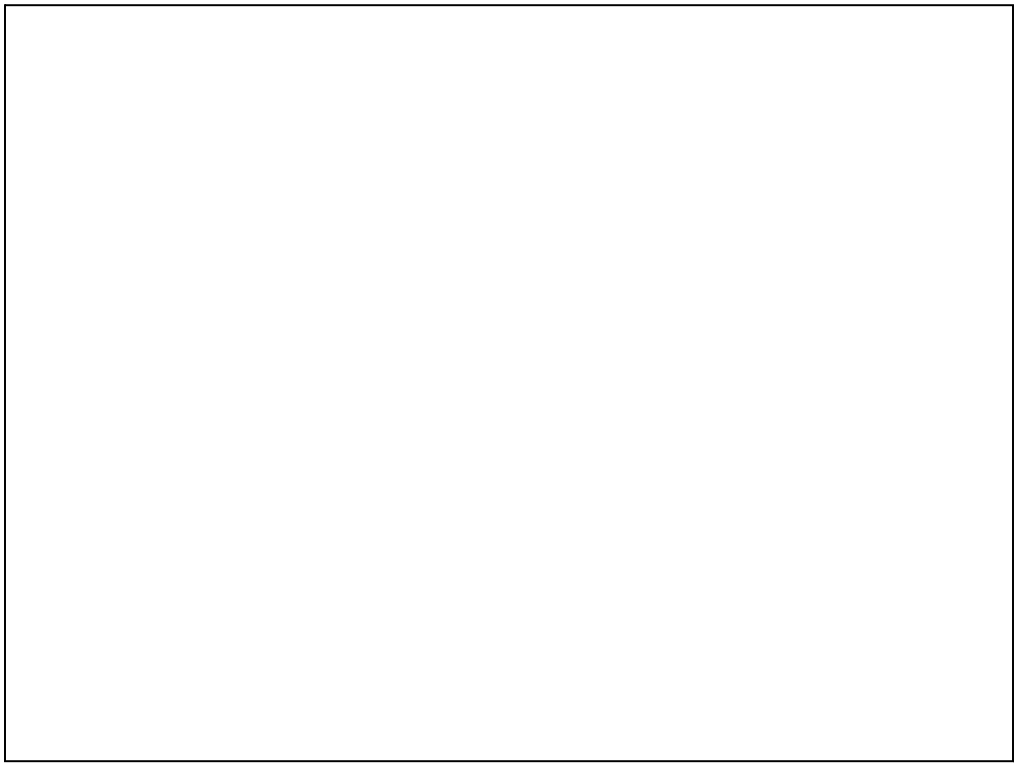
http://en.wikipedia.org/wiki/Bloom_County#

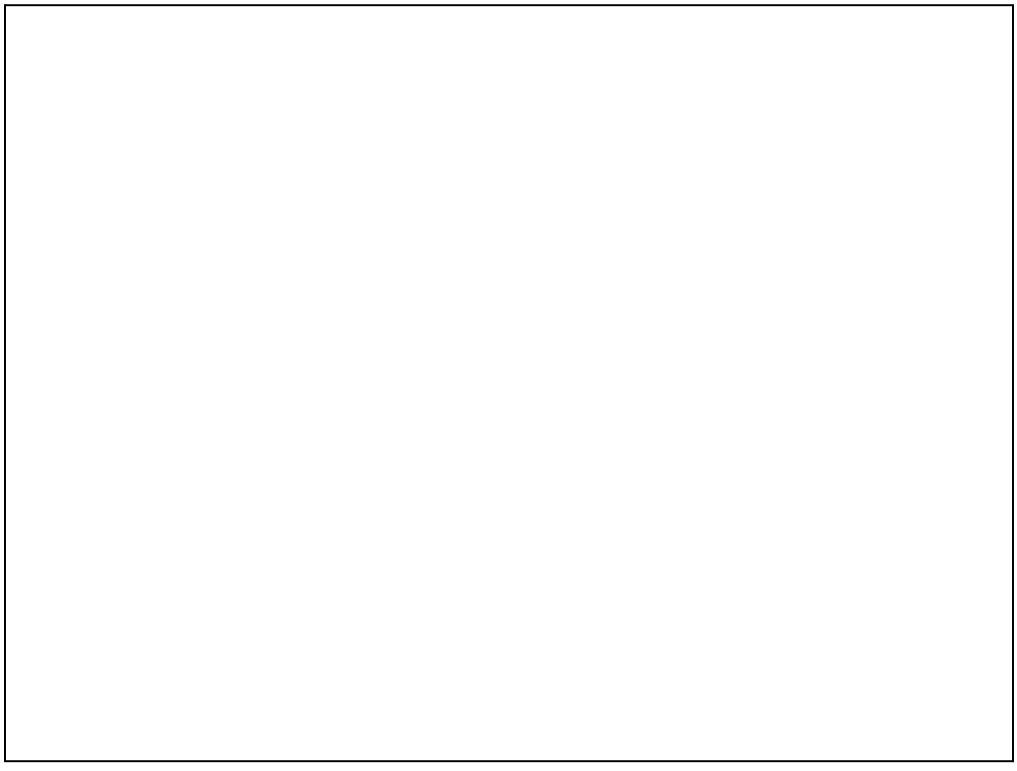


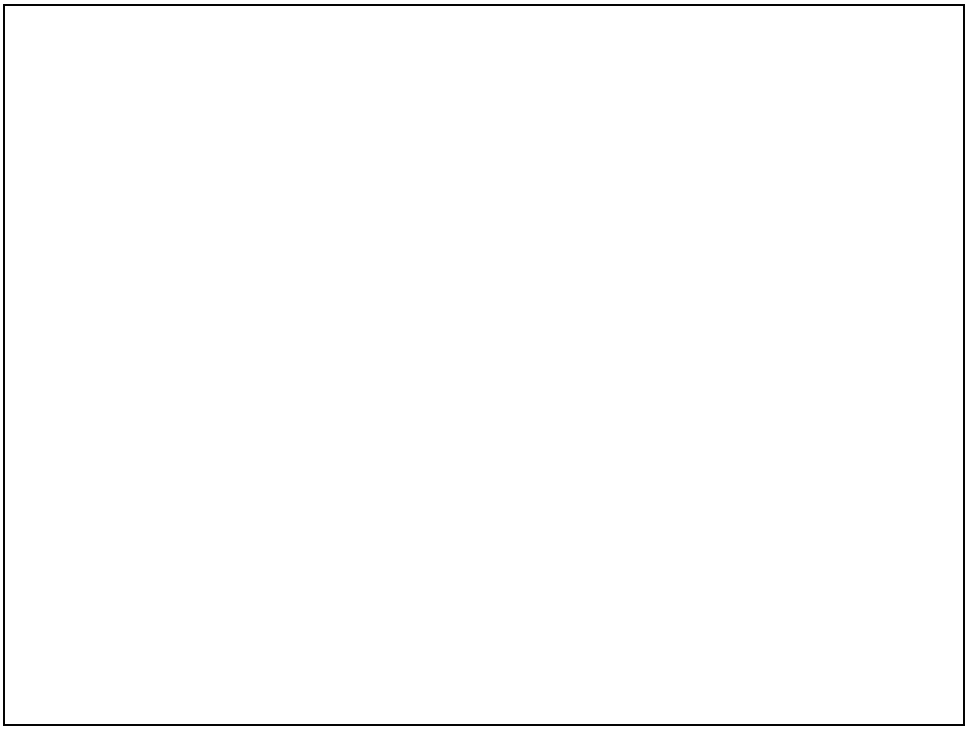


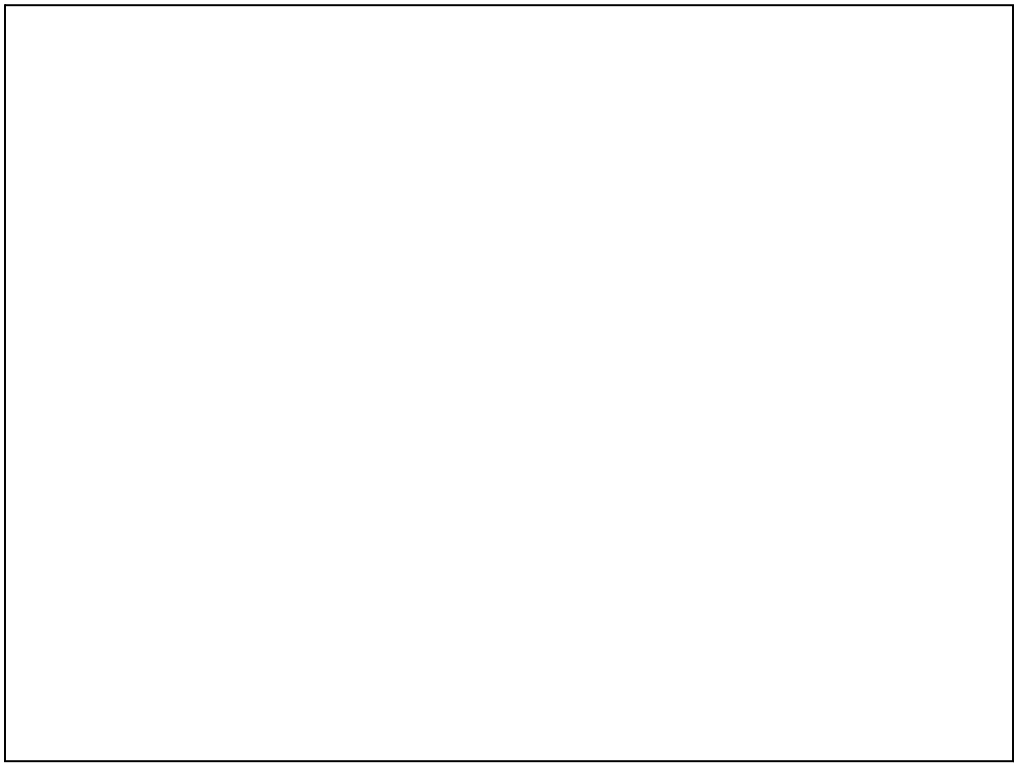


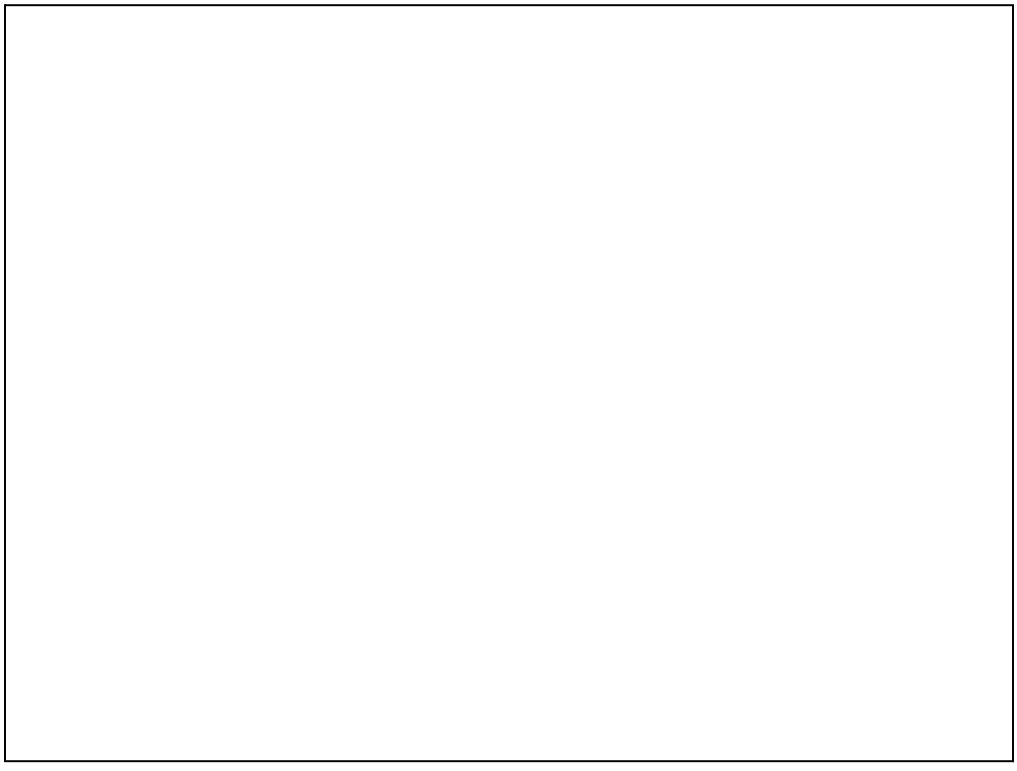


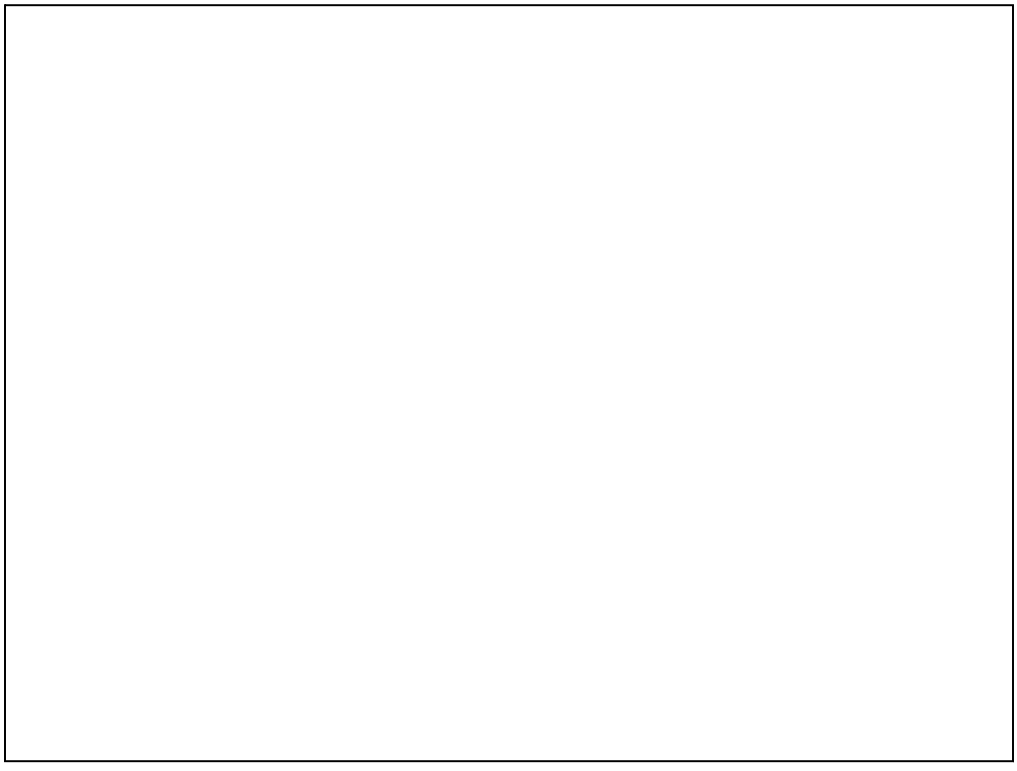


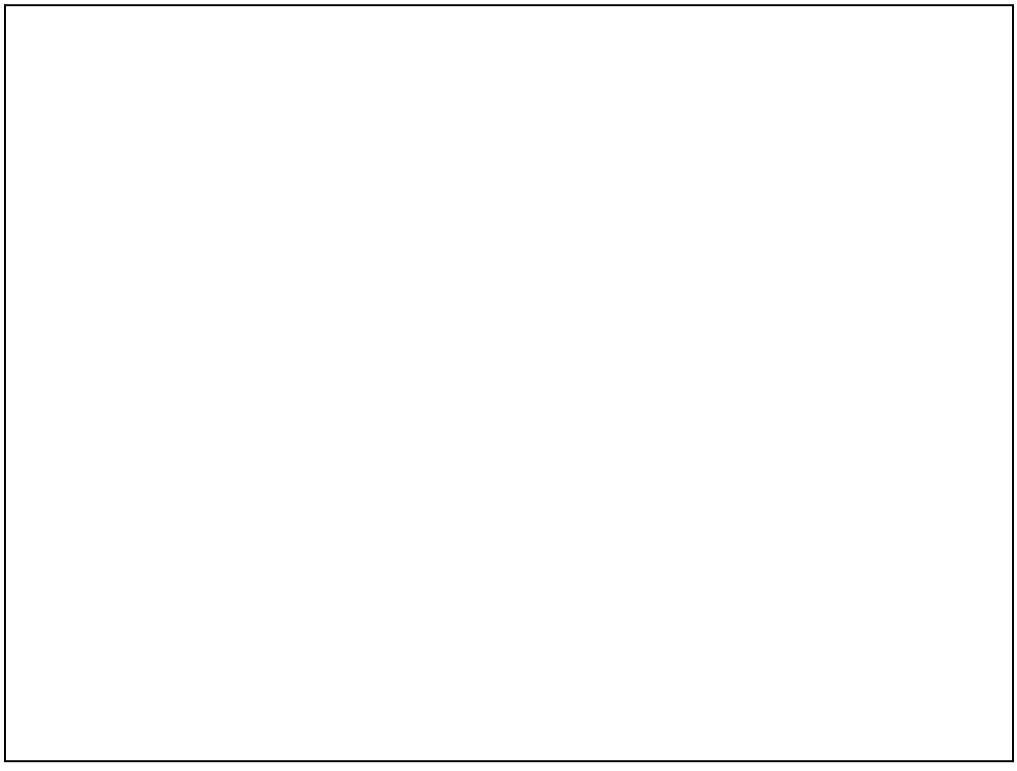


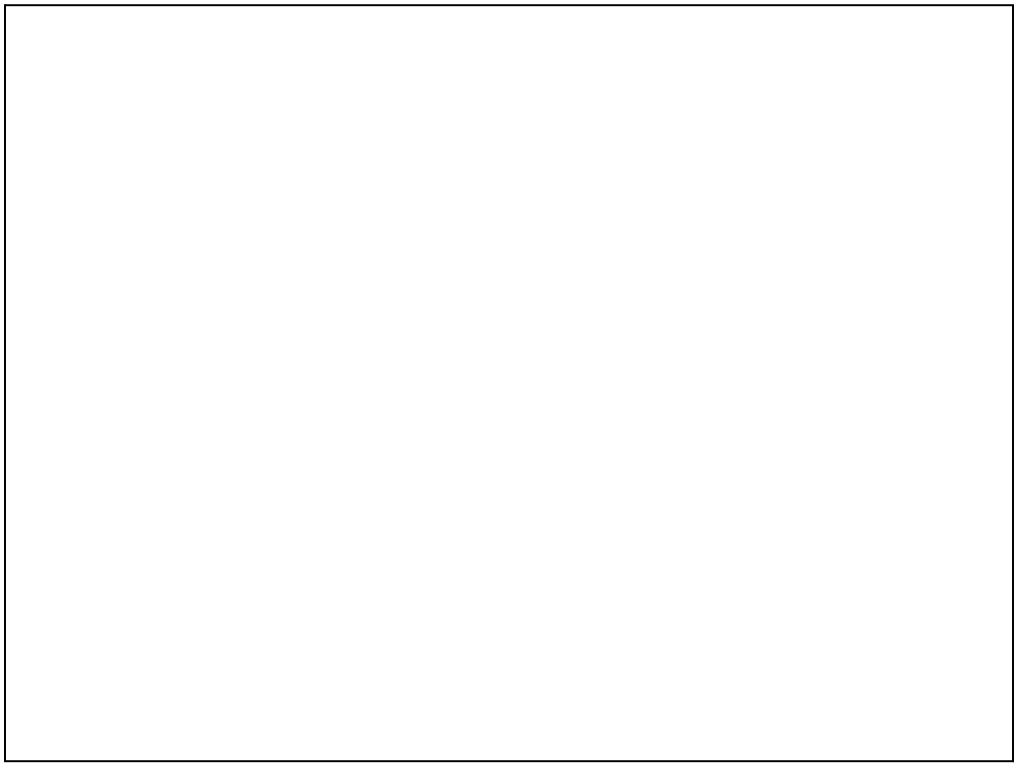








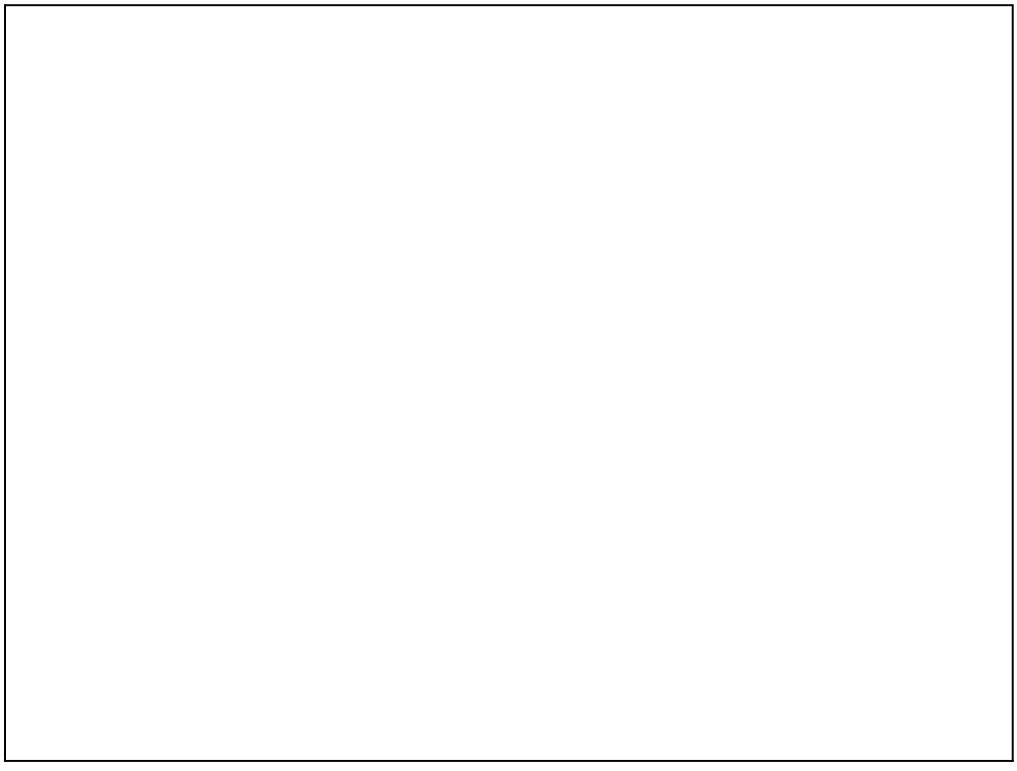


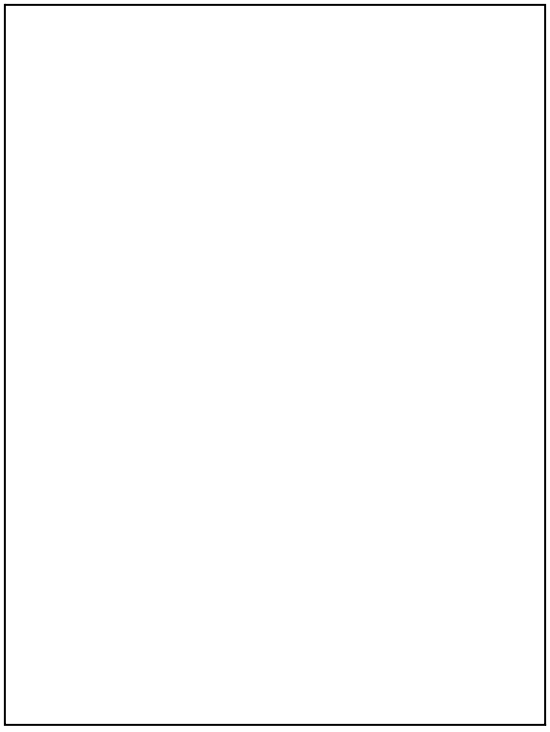


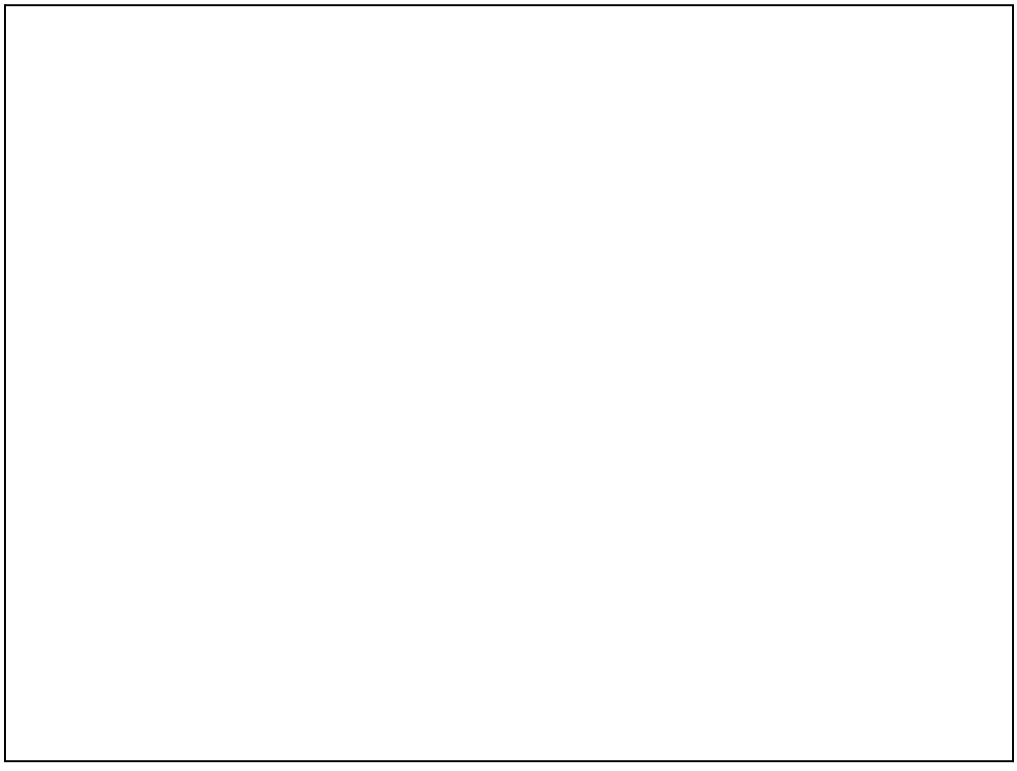
Wireshark Tutorial on TCP Sequence Numbers and ACK's

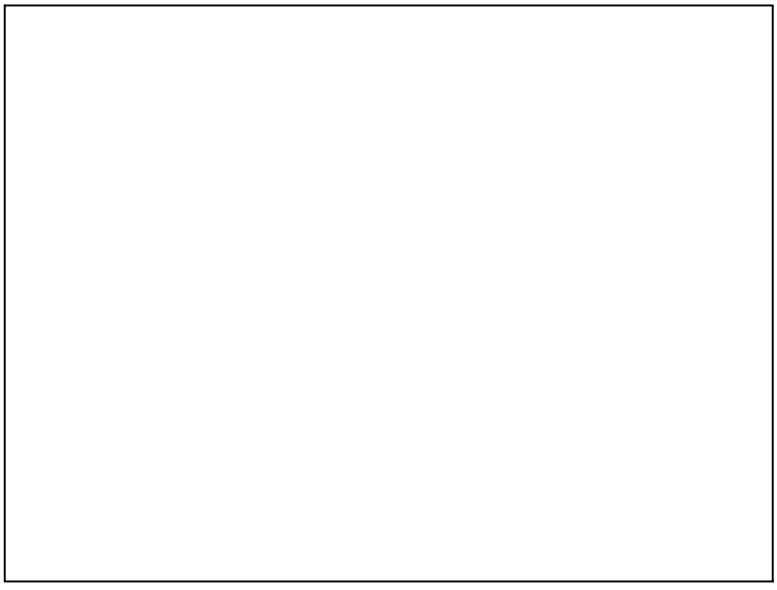
- Nice short tutorial plus example file on how TCP manages sequence numbers and acks

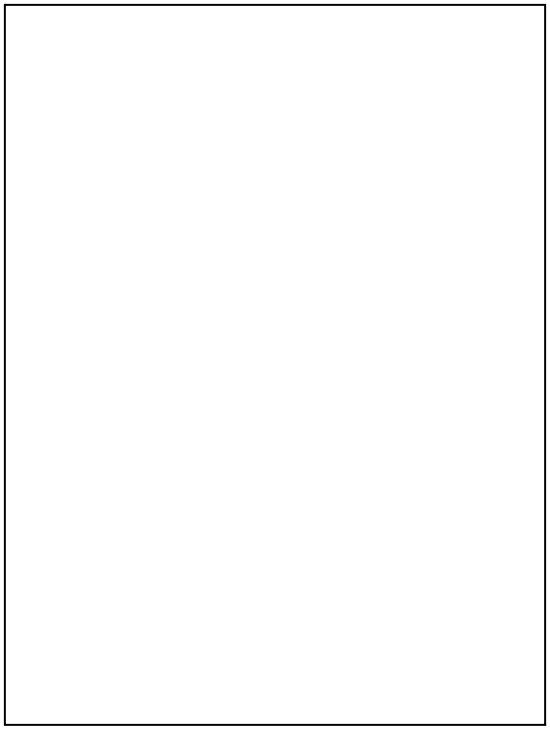
  http://packetlife.net/blog/2010/jun/7/understanding-tcp-sequence-acknowledgment-numbers/

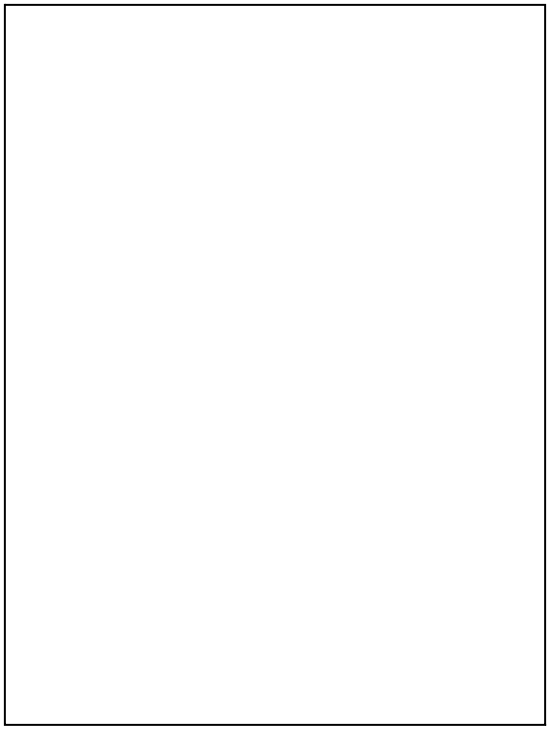


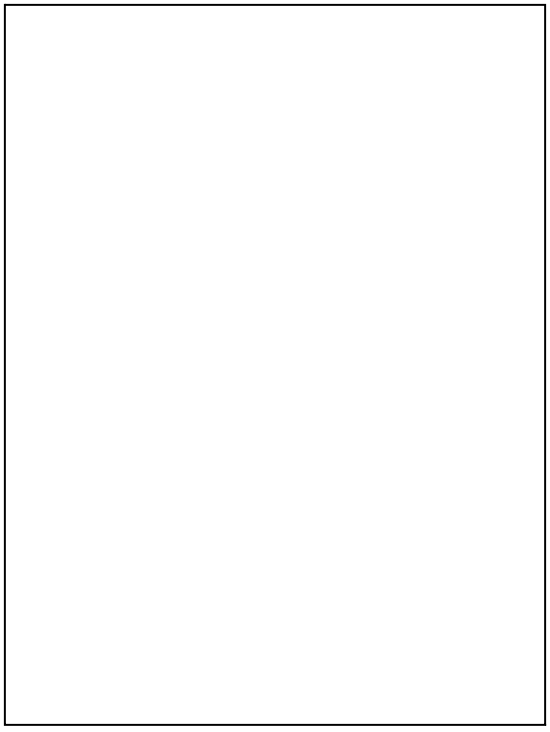


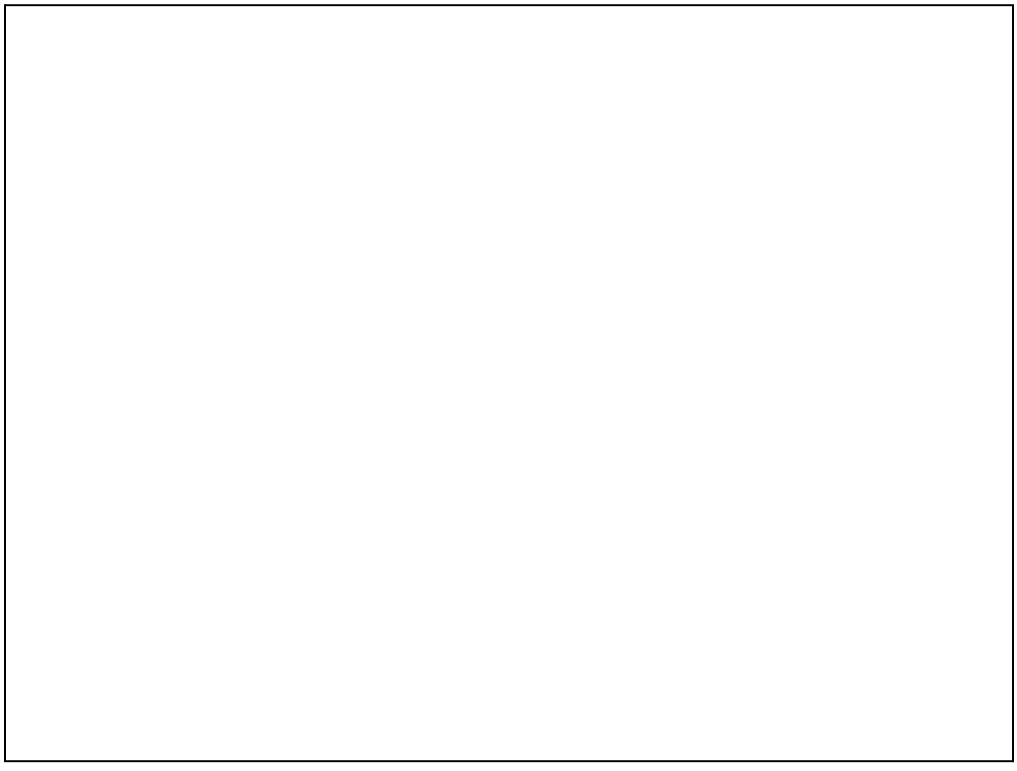


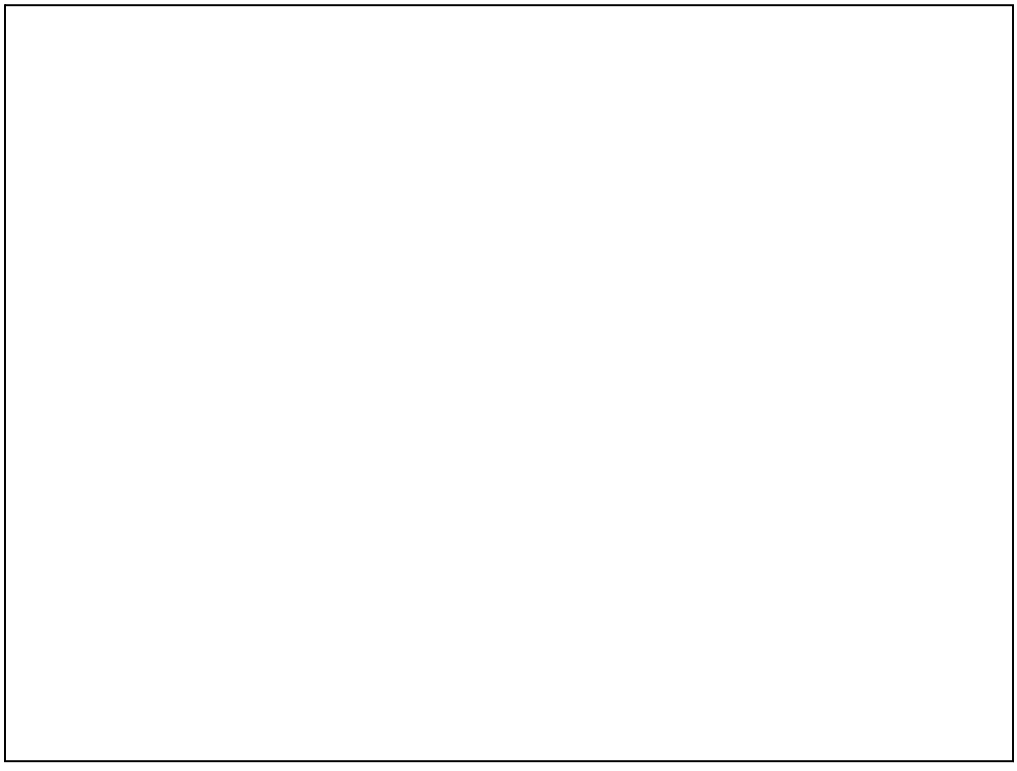












Performance Metrics

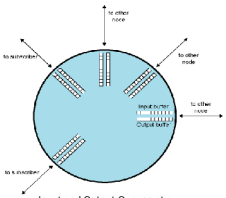
- Two important performance measures
- Throughput
  - Data rate in bps - protocol overhead (trans. delay, propagation, queueing and processing delays)
- Delay
  - Transmission delay
    - Time for transmitter to send all bits of packet
  - Propagation delay
    - Time for one bit to transit from source to destination
  - Processing delay
    - Time required to process packet at source prior to sending, at any intermediate router or switch
- We discussed these. What we did not discuss is
  - Queuing delay: Time spent while waiting in queues

Queuing Delays

- Queuing delays are significant in performance of communications networks
  - —Grow dramatically as system approaches capacity
- In shared facility
  - – network,
  - – transmission line,
  - – road network,
  - – checkout lines
- Performance worsens exponentially as demand approaches capacity

What Is Congestion?

- Data network is a network of queues
- Congestion occurs when the number of packets being transmitted through the network approaches the packet handling capacity of the network



Input and Output Queues at a Network Node

Figure 5.9 Input and Output Queues at Node

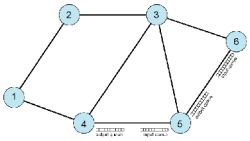
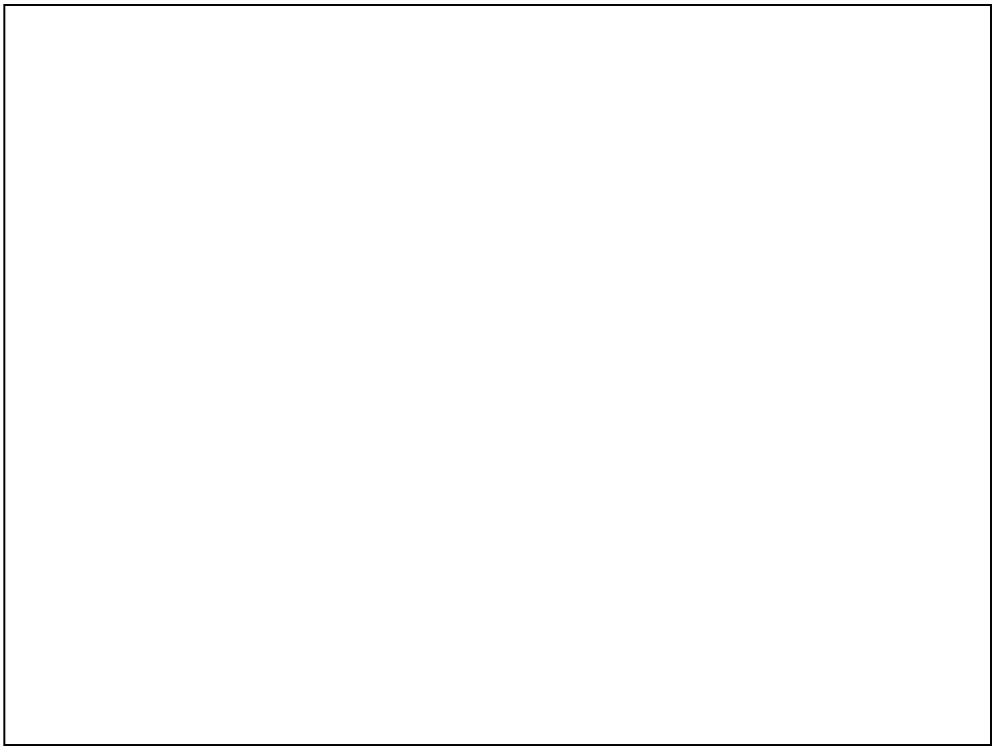
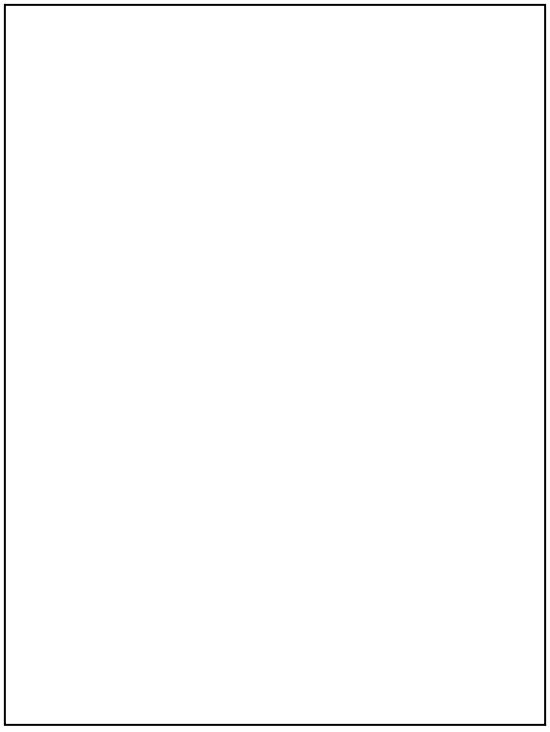
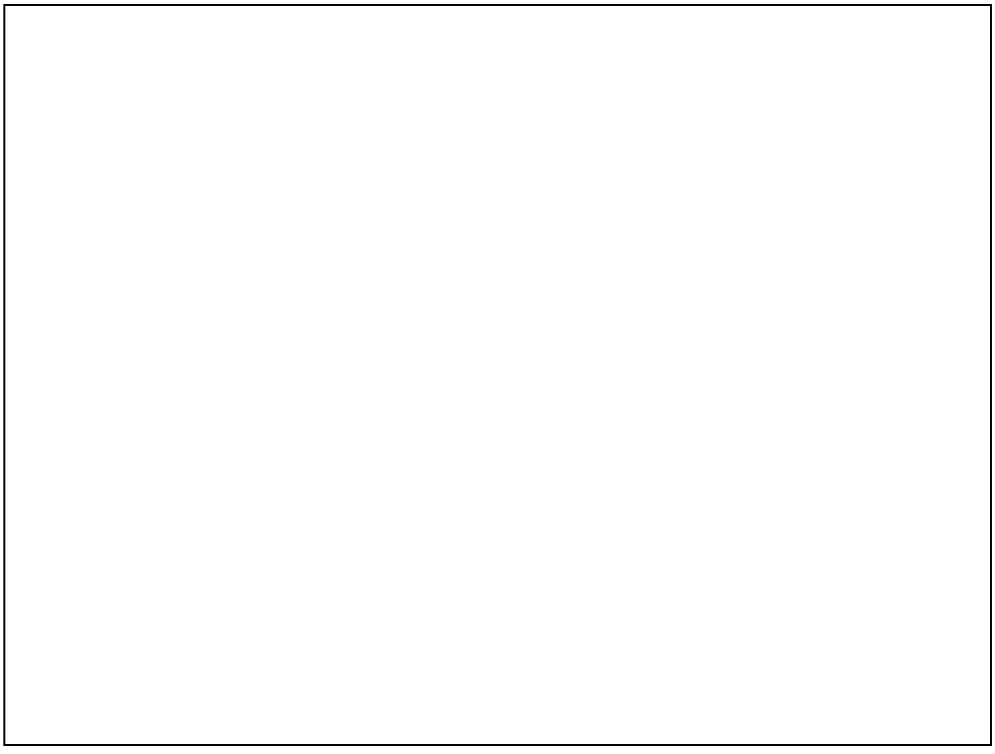
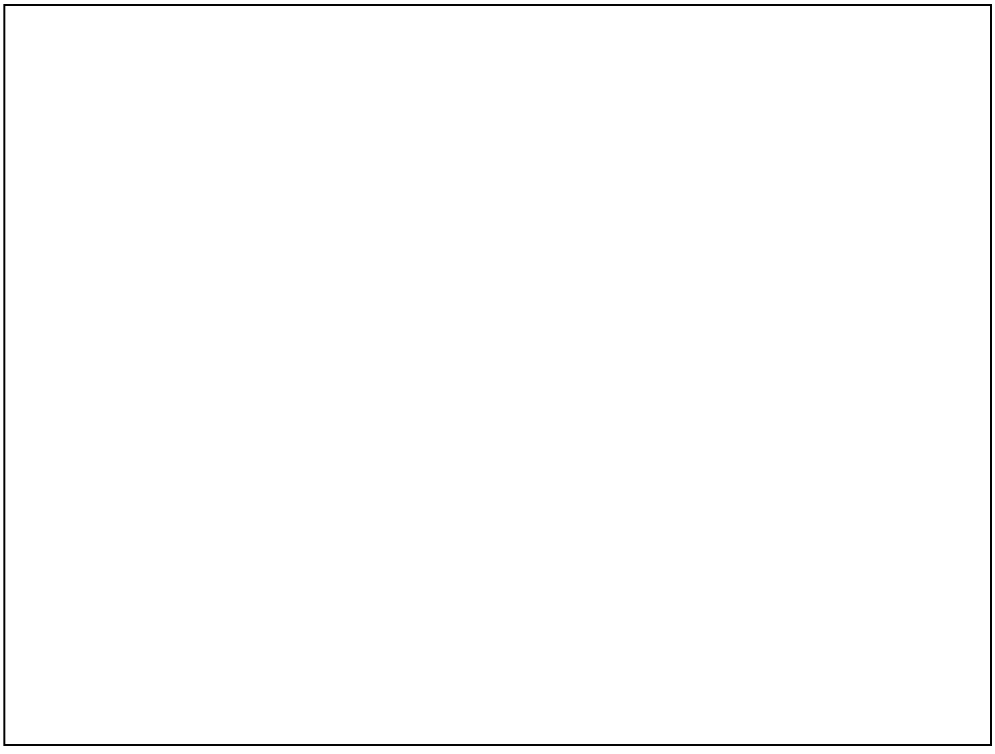


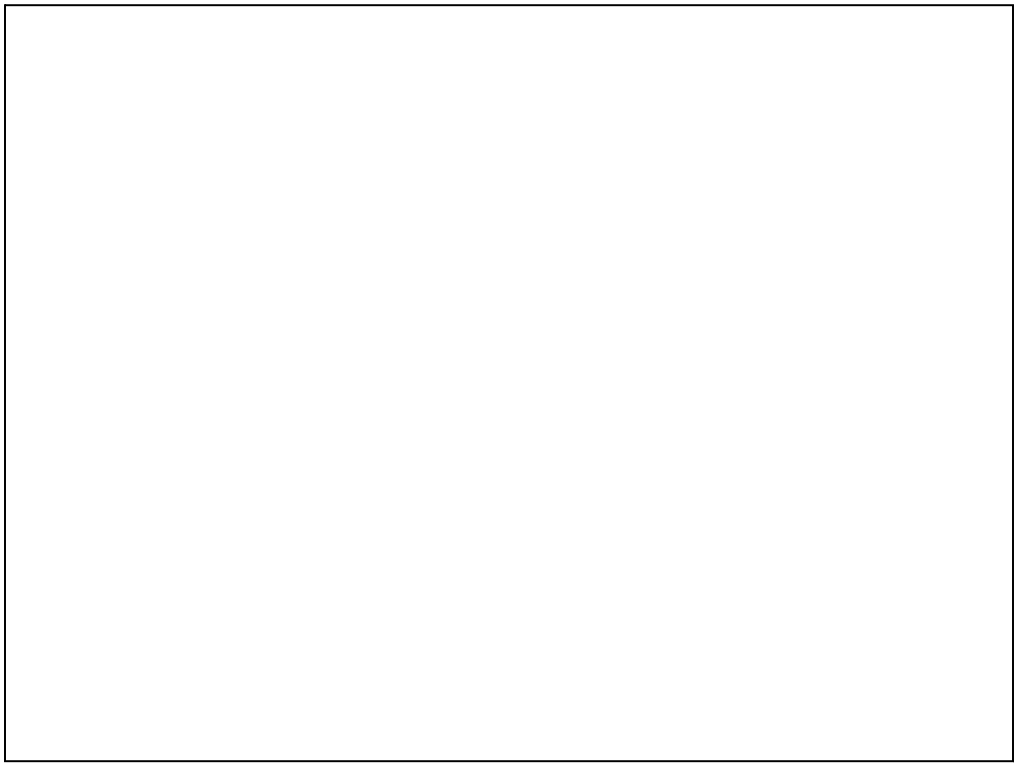
Figure 5.10 Interaction of Queues in a Data Network

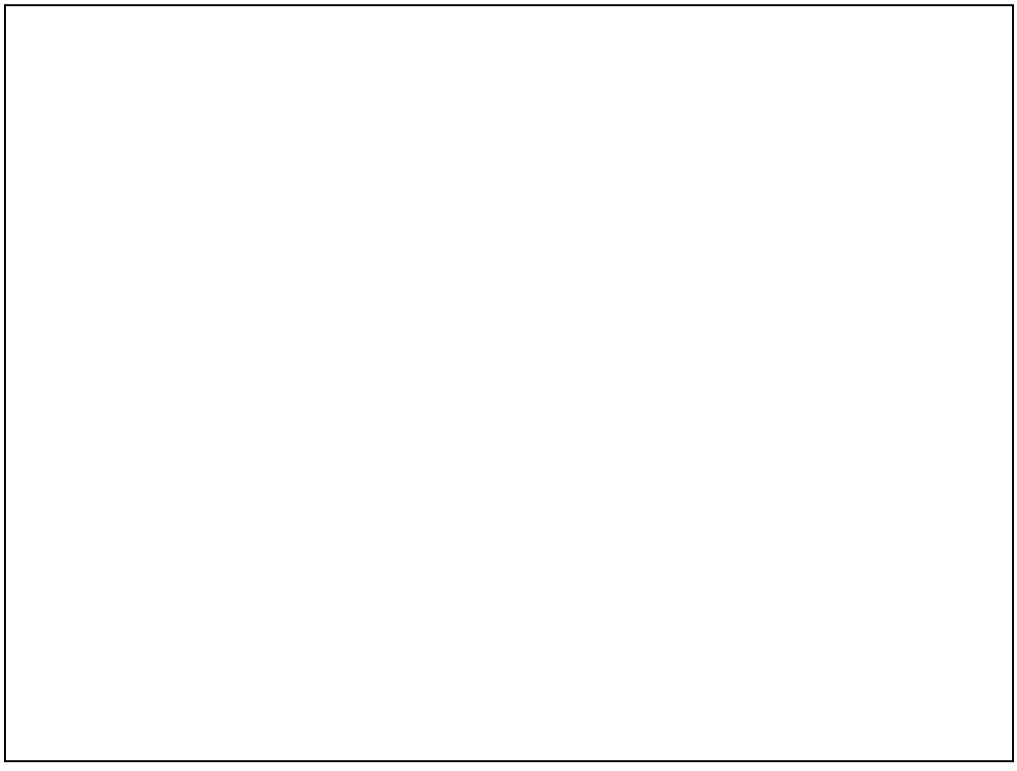


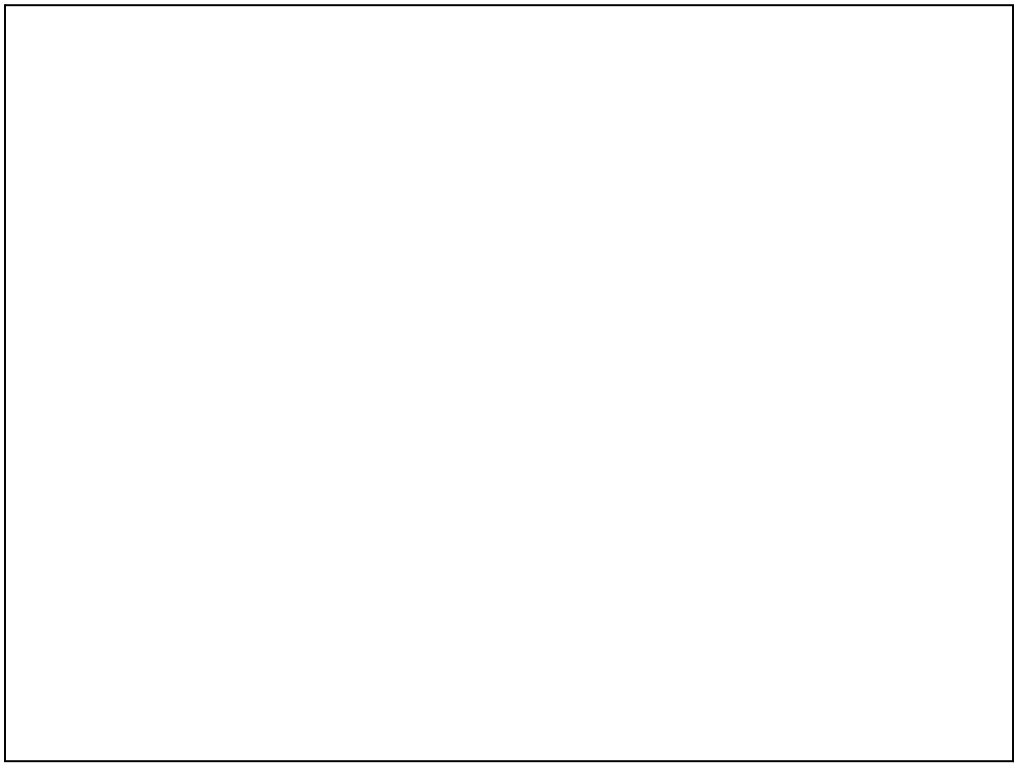


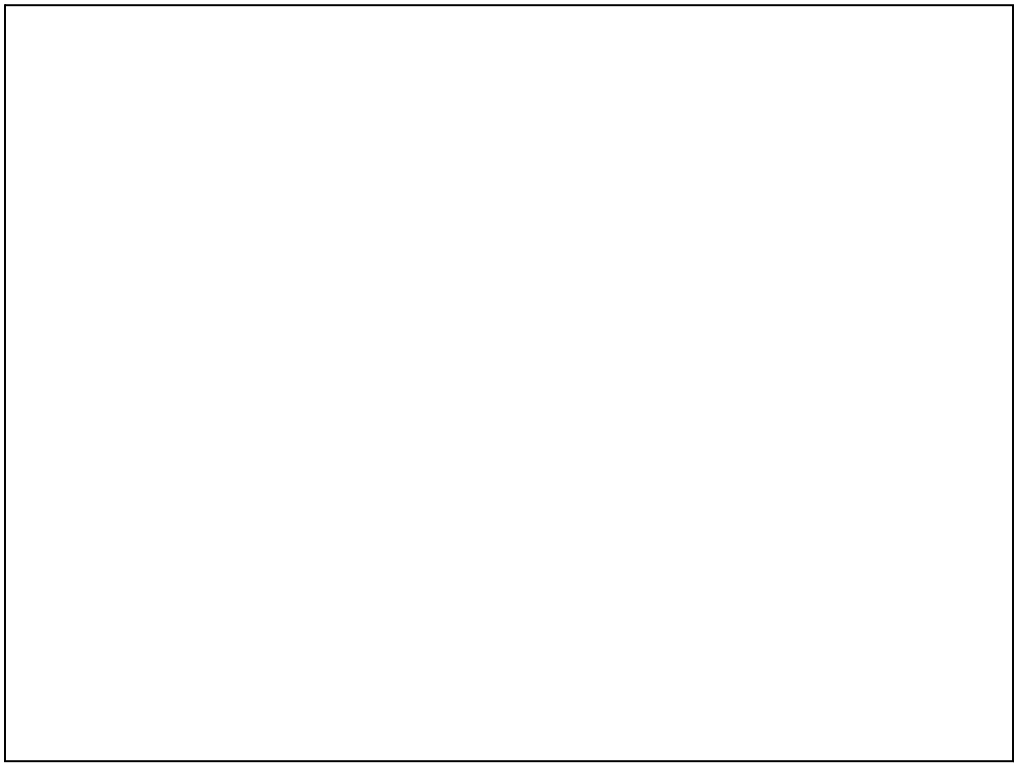


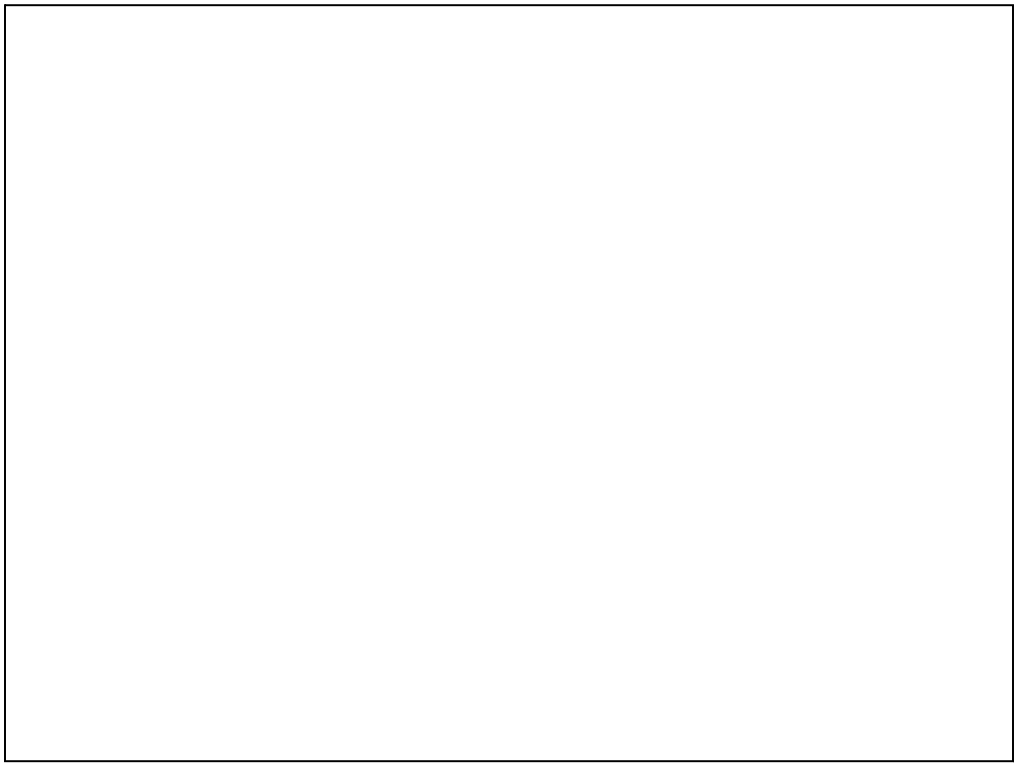


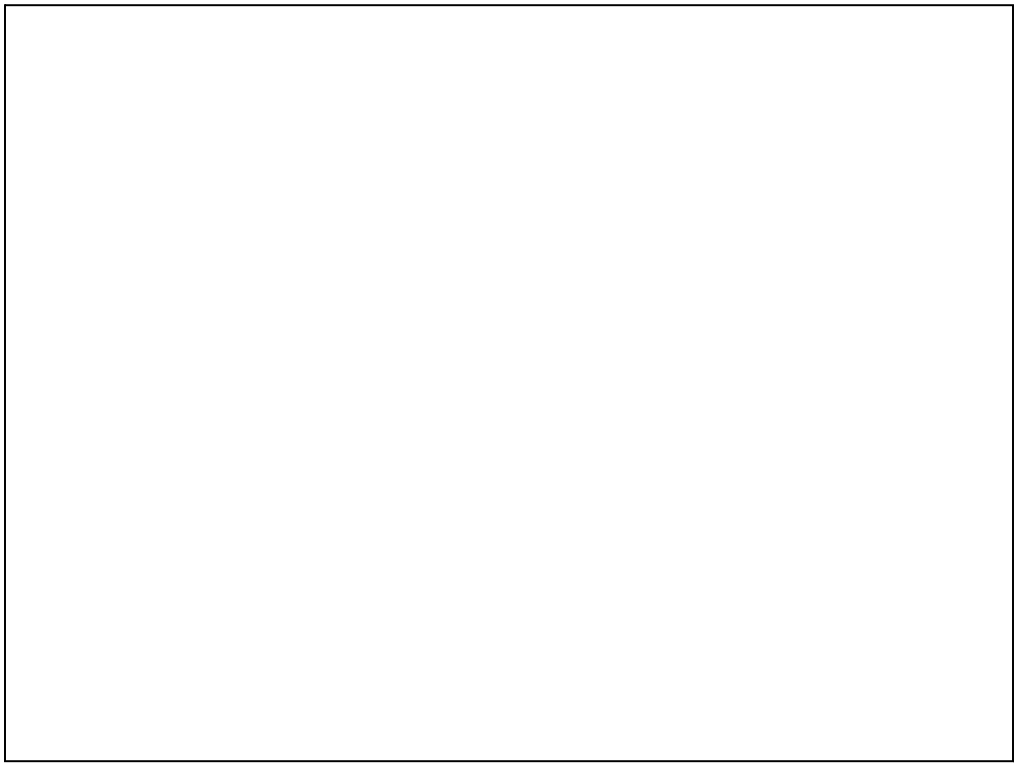


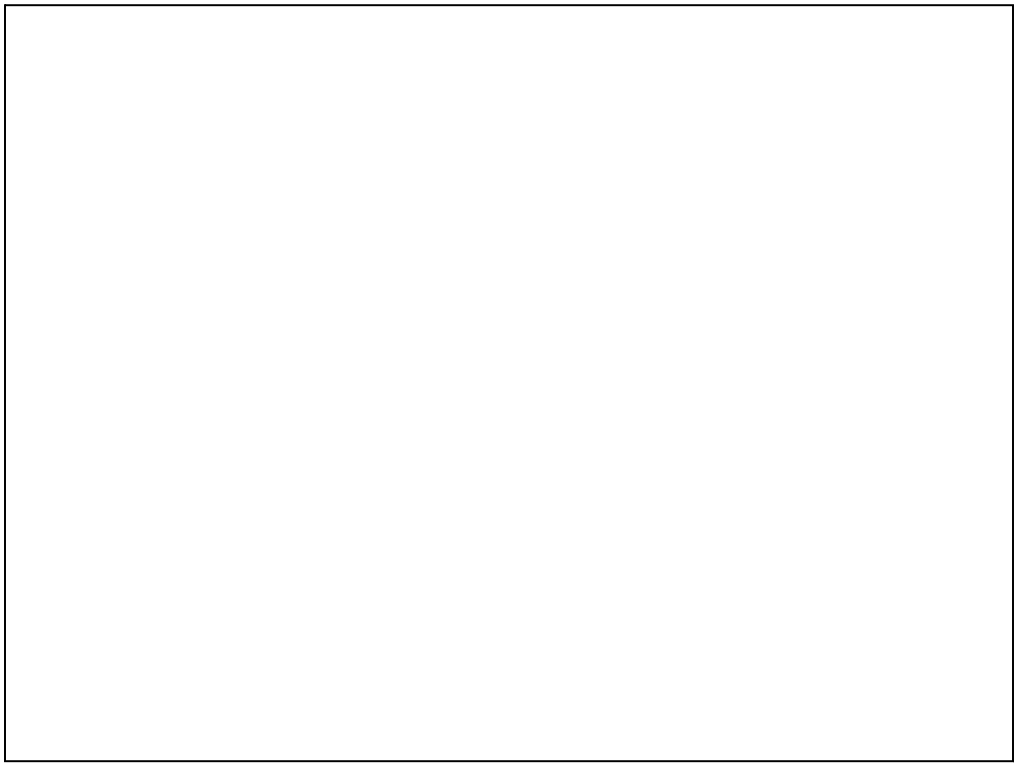








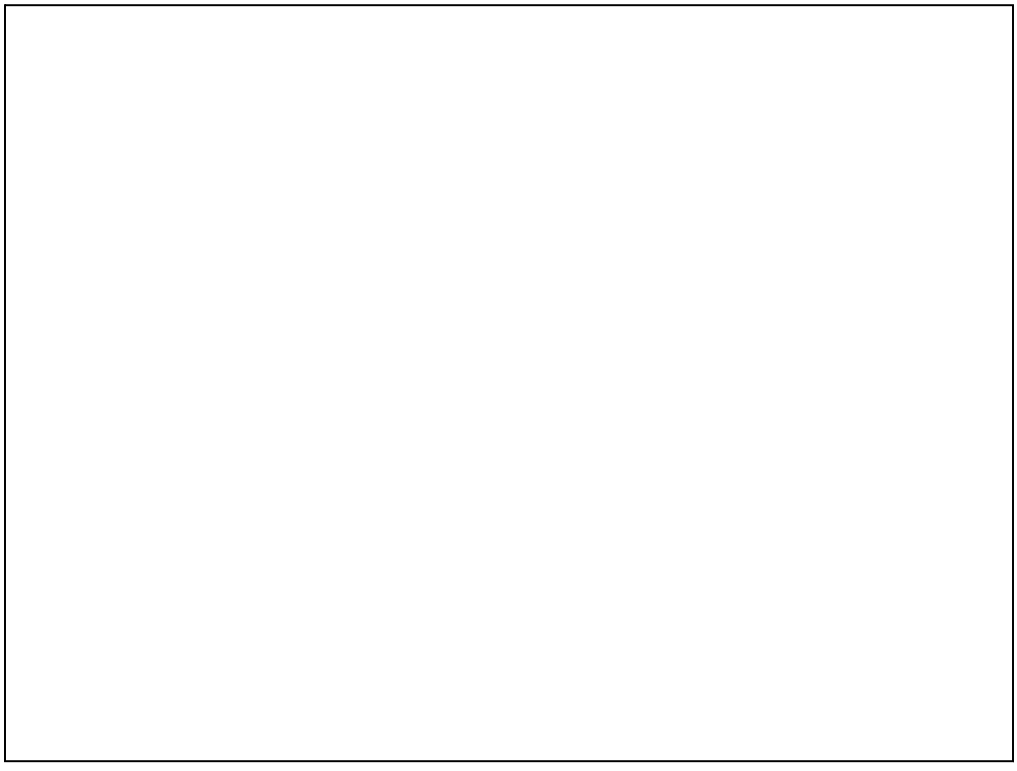


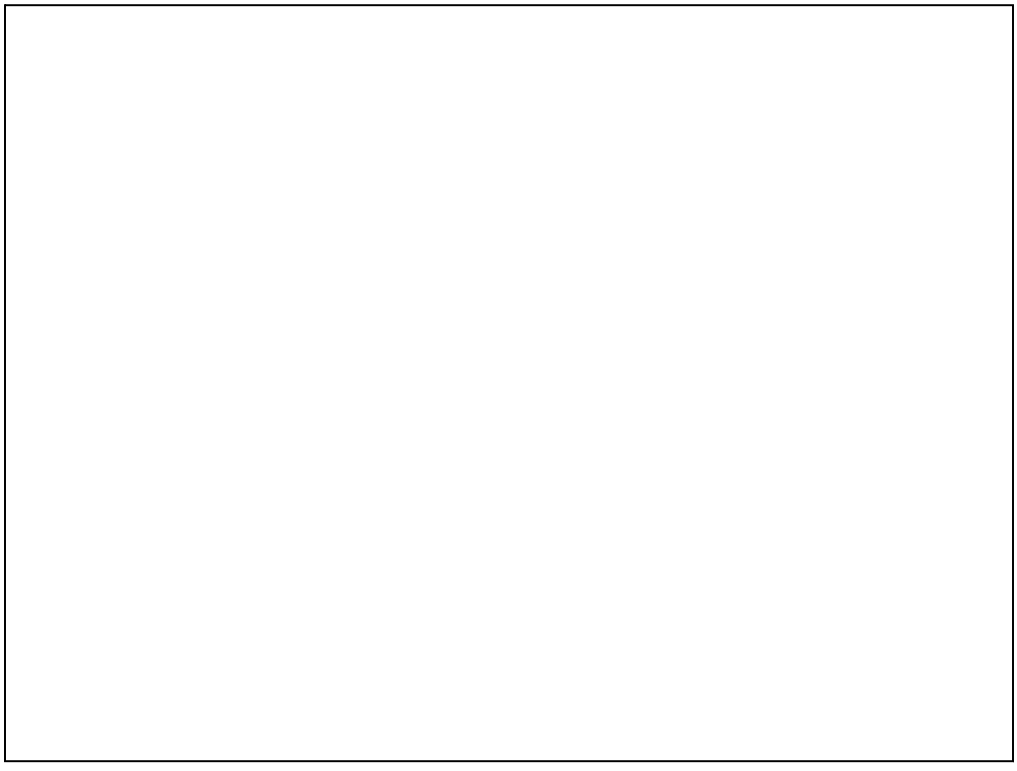


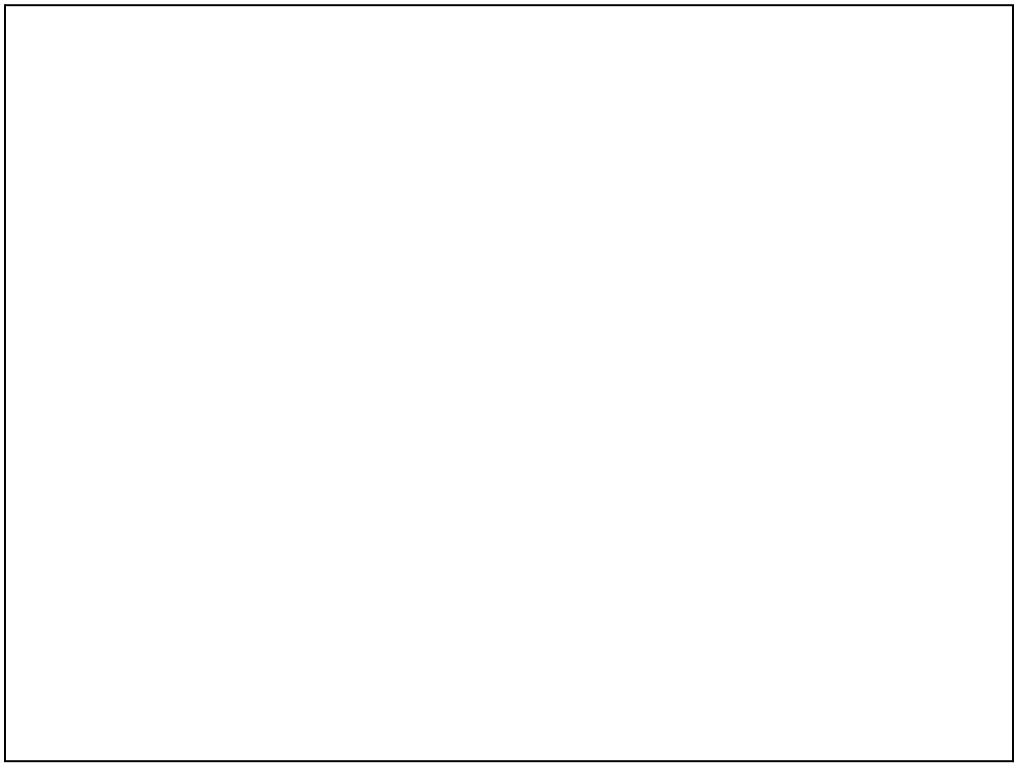
TCP Congestion Control

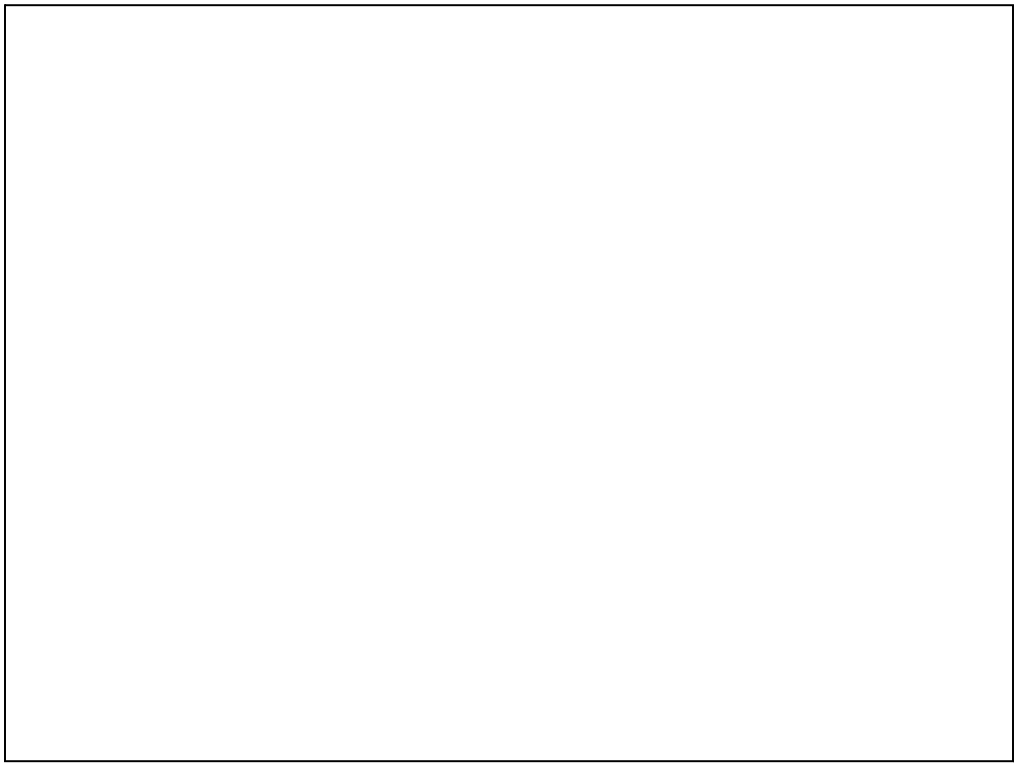
- Essential strategy
- TCP host sends packets into network without reservation and then host reacts to observable events.
  - Originally TCP assumed FIFO queuing
- Basic idea Each source determines how much capacity is available to a flow
- ACKs are used to ***'pace'*** transmission of packets such that TCP is **"self-clocking"**

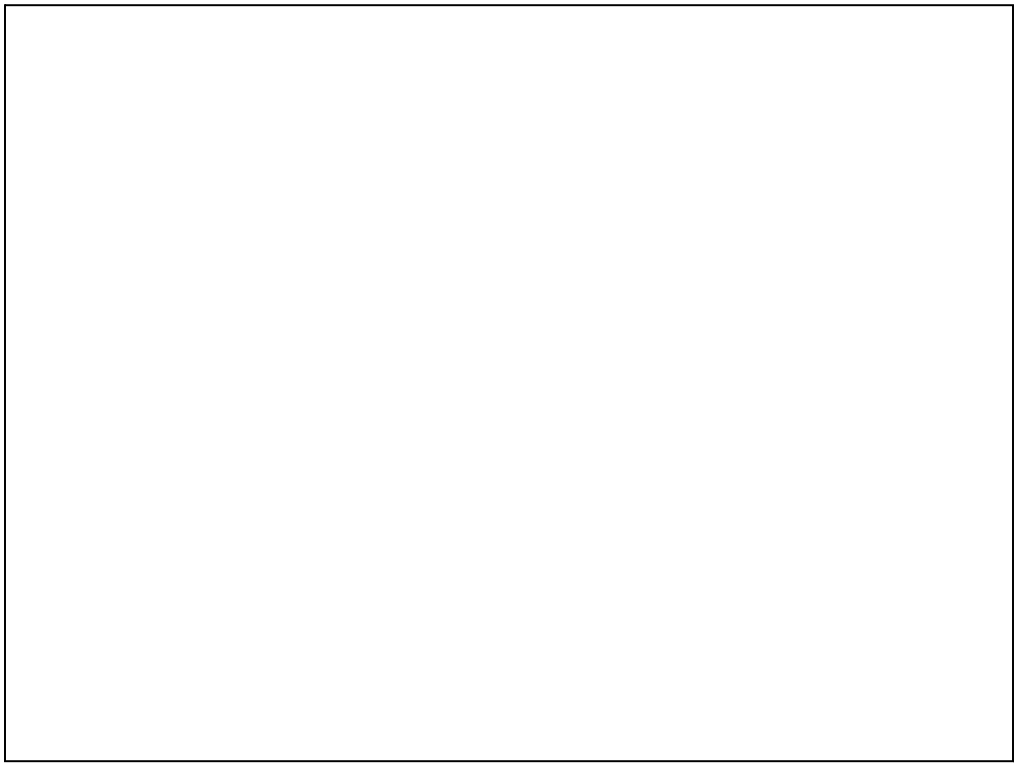
39

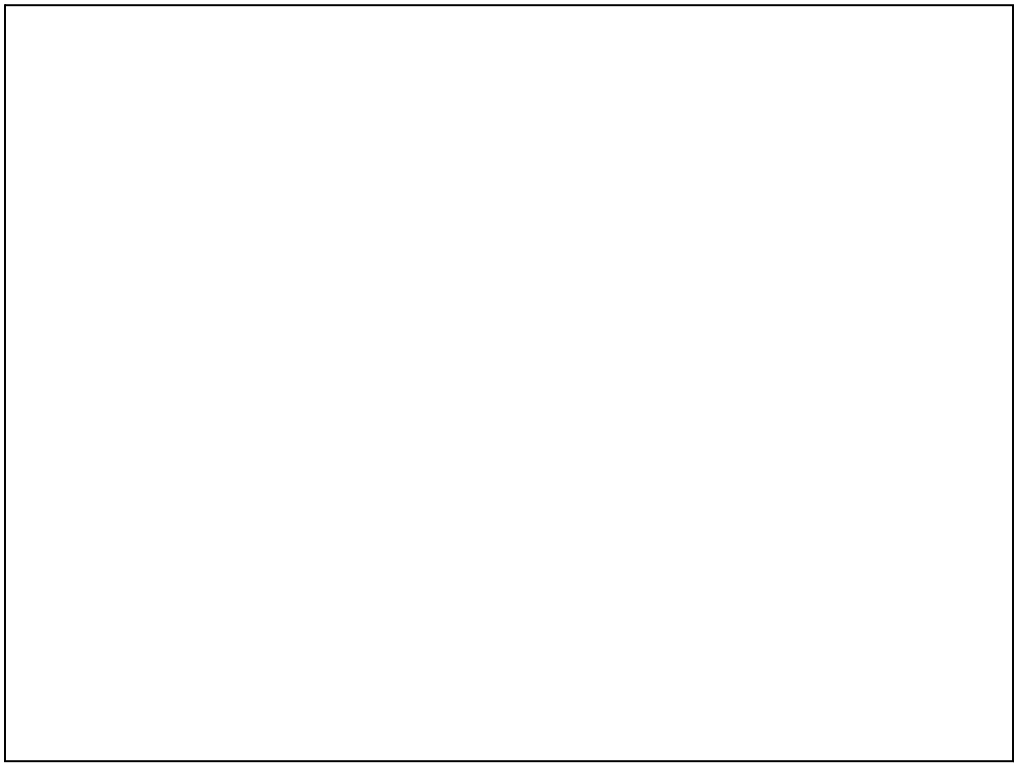


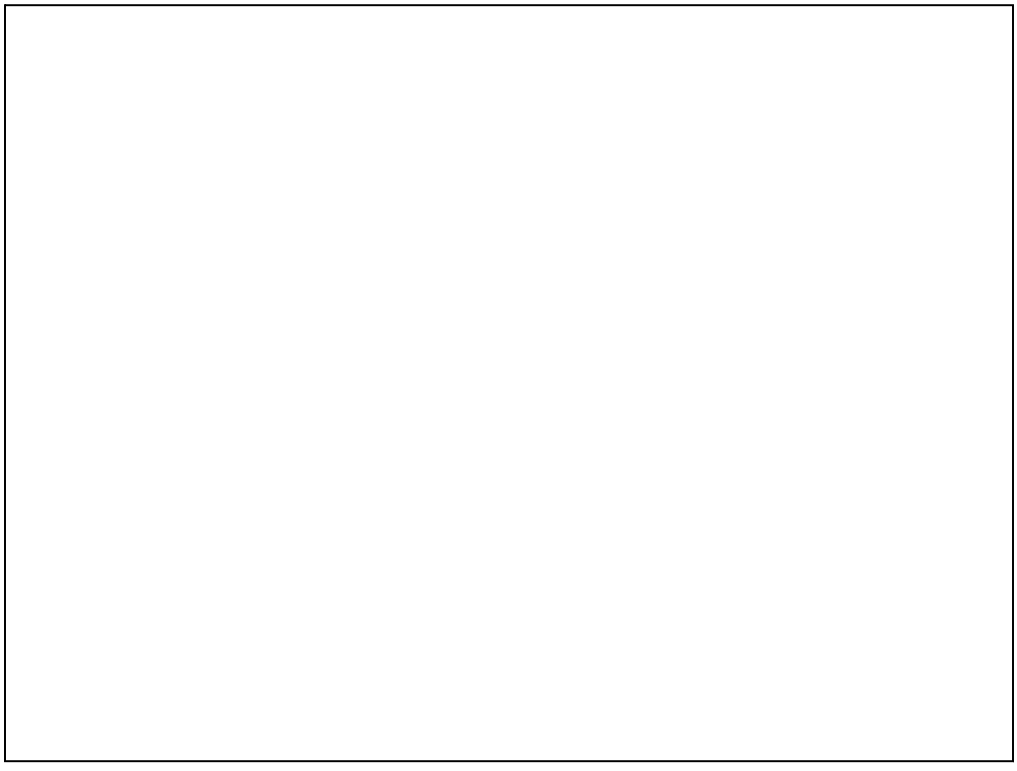


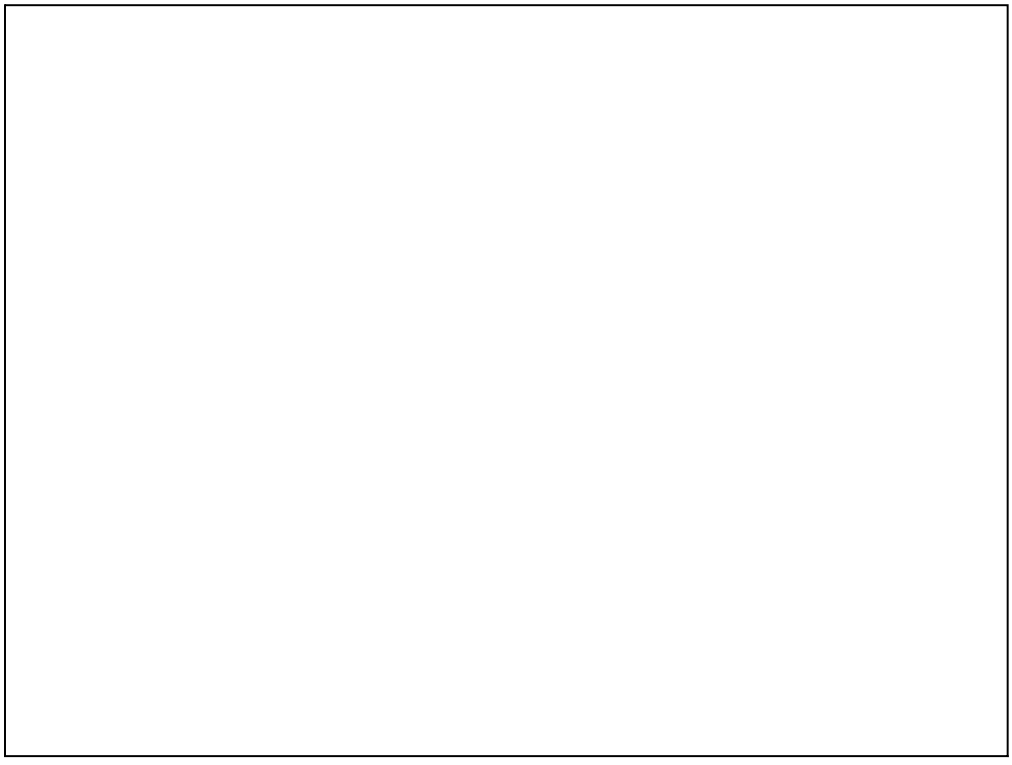


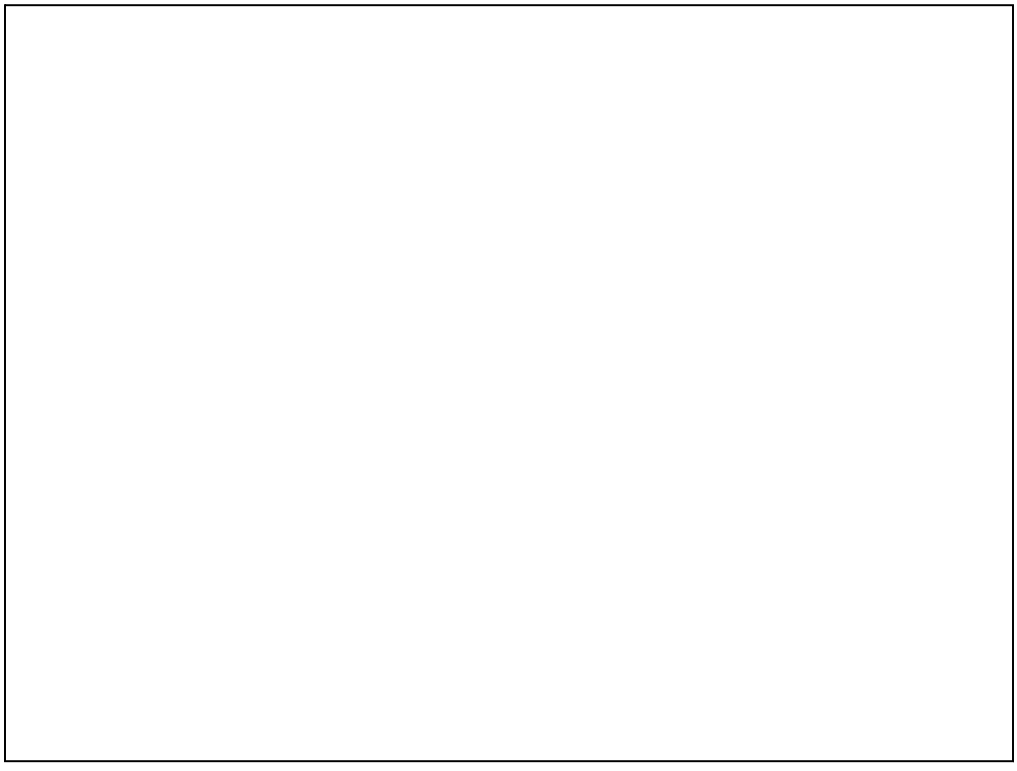


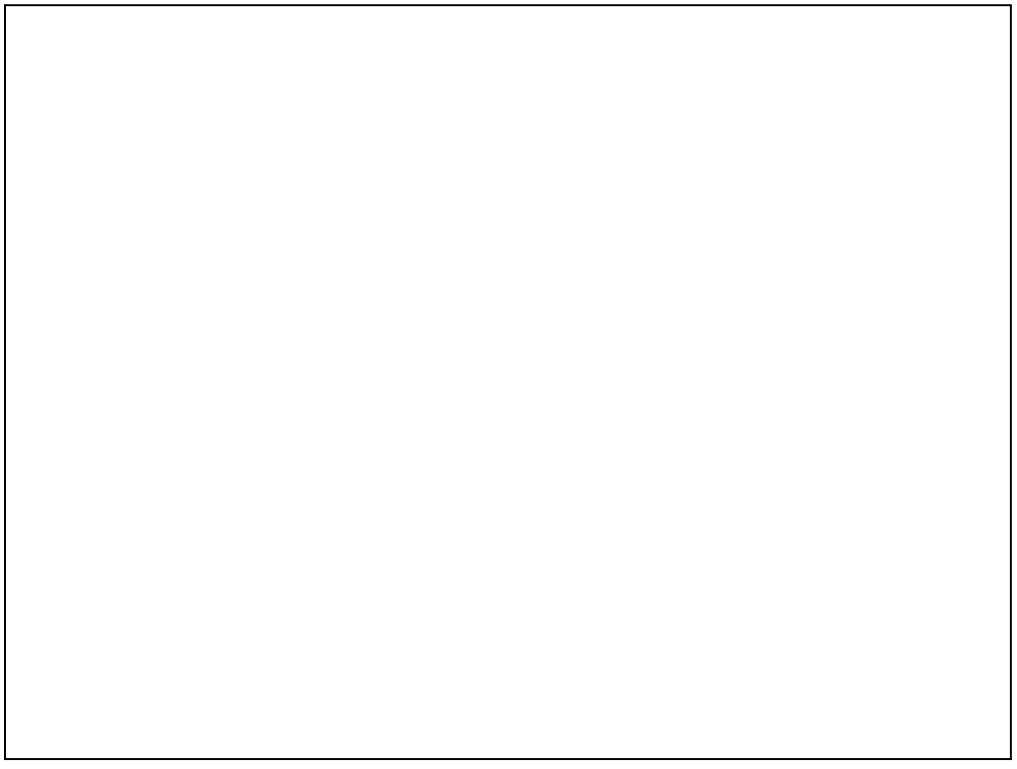


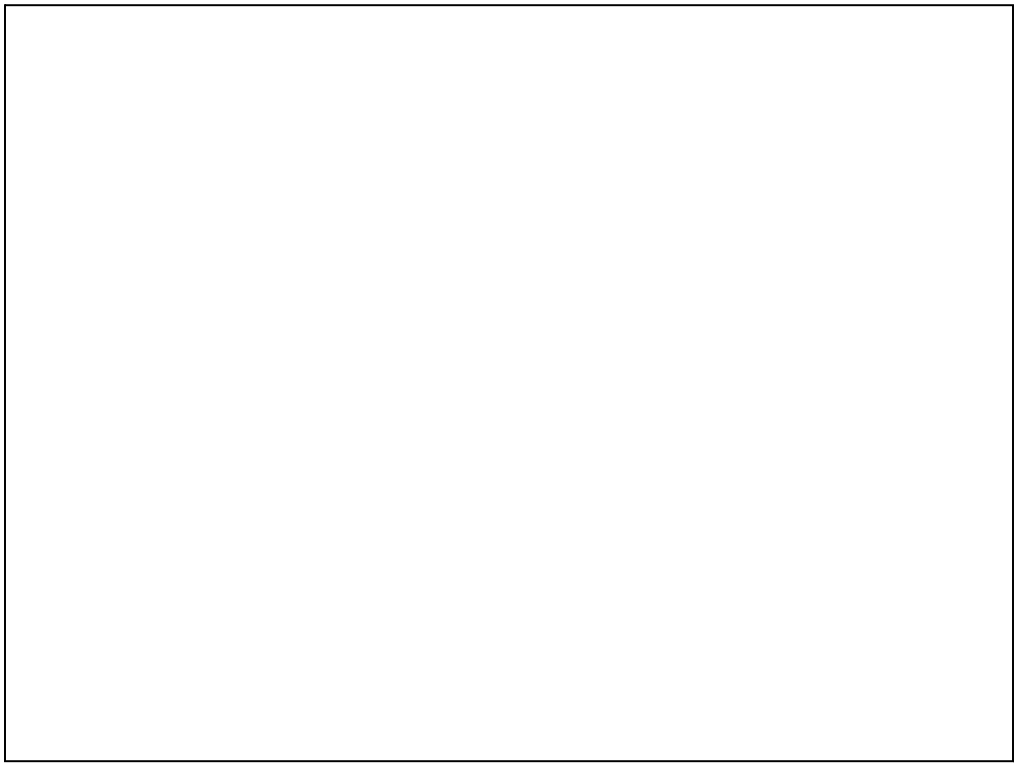


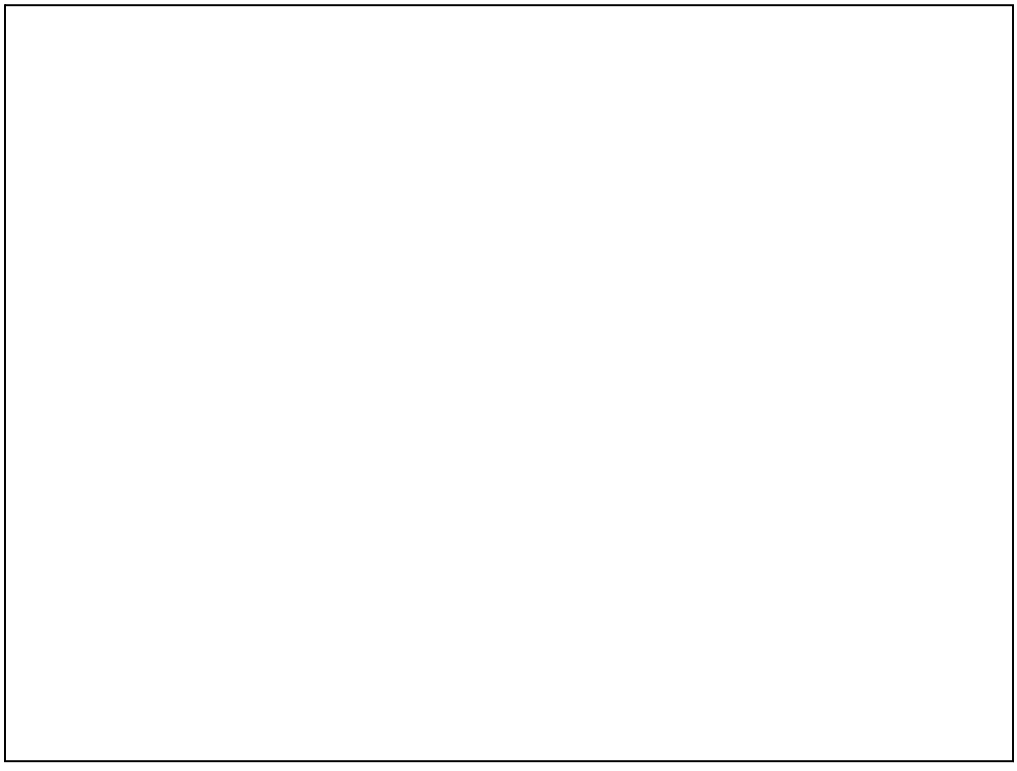








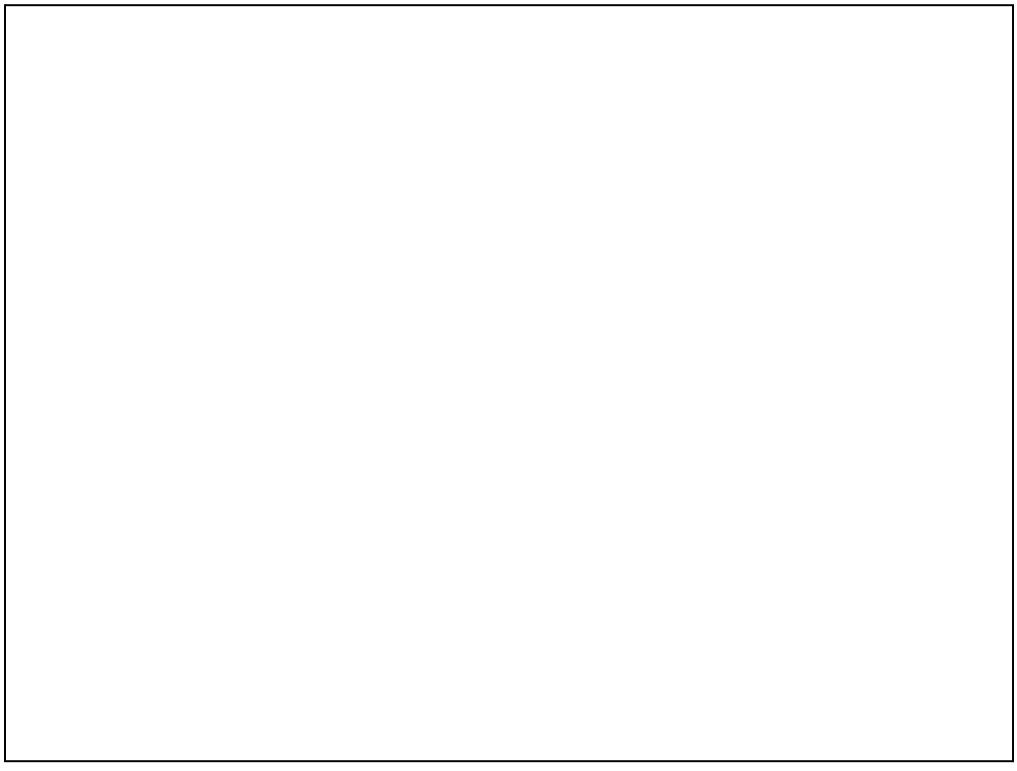




Many TCP ‘flavors’

- **TCP New Reno** – Variation of Reno,
- **TCP Vegas**
  - Adjusts window size based on difference between expected and actual RTT
- **TCP Binary Increase Congestion** (**BIC)**
  - Uses optimized congestion control algorithm for high speed networks with high latency
- **TCP Cubic**
  - TCP Cubic allows very fast window expansion however, it also makes attempts to slow the growth of cwnd sharply as cwnd approaches the current network ceiling, it is the default in Linux

Nice overview of newer TCP Congestion Algorithms
http://intronetworks.cs.luc.edu/current/html/newtcps.html#tcp-cubic



Summary

- TCP interacts with routers and reacts to implicit congestion notification,
  - Packet drop
- By reducing TCP sender's congestion window
- TCP increases congestion window using slow start or congestion avoidance
- Does this for every user in the network

Currently, there are a variety of newer TCP algorithms for congestion control

Each has slightly different attributes for optimal congestion control