# Lesson 10: Congestion control

Van-Linh Nguyen

Fall 2024

國立中正大學 Cybersecurity Lab

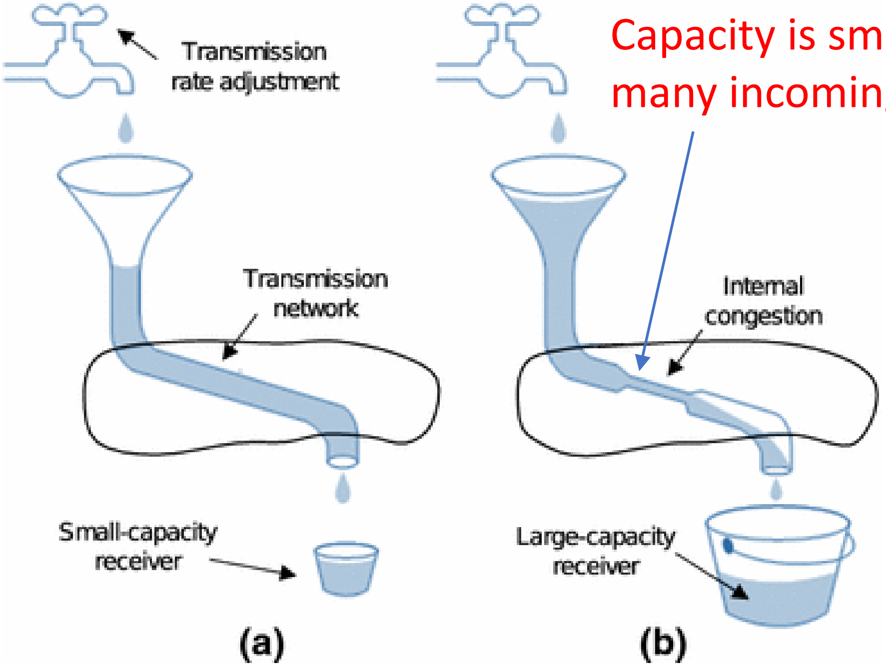
# Outline

- Congestion control
  1. Congestion control mechanisms and algorithms
  2. Quality of service

# Congestion issue

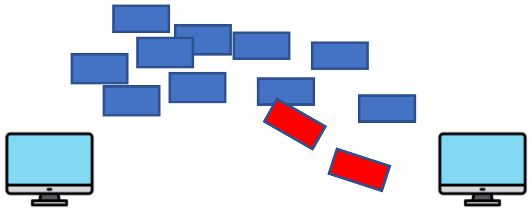
- Congestion in computer networks occur when there's just not enough bandwidth to handle the existing amount of traffic

A limited number of road lanes but too many vehicles




Capacity is small but too many incoming packets

# Congestion issue

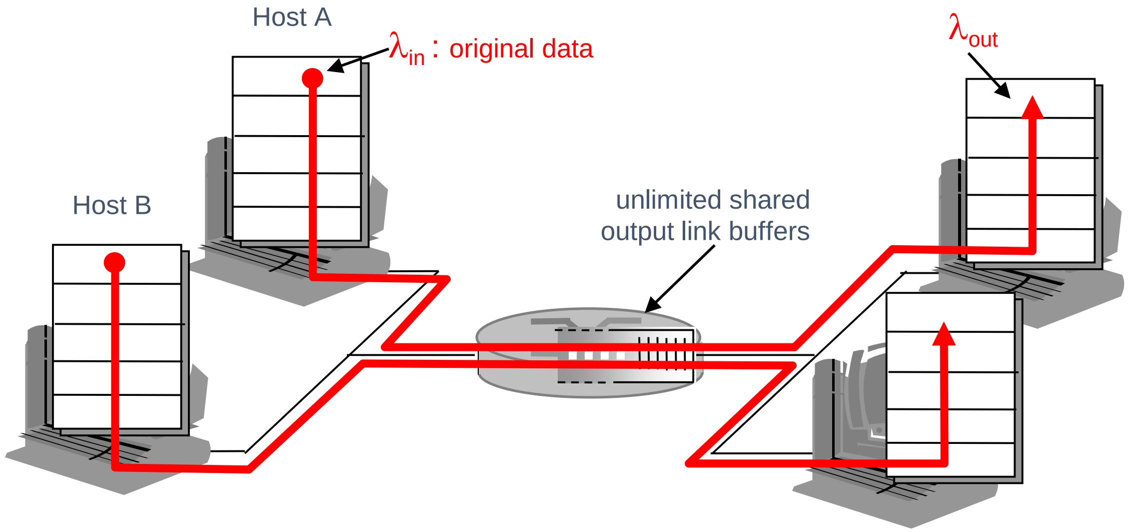


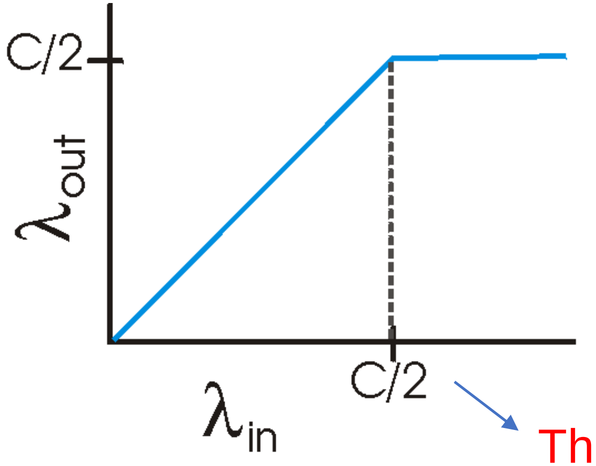
- Too many sources sending too much data for *network* to handle
- Consequences: lost packets, long delay
- The hosts would time out and retransmit their packets, resulting in even more congestion

  - **Sender A**: Hey, I am sending 100 gifts to my darling…
  - **Channel**: I am helping you to forward but just 20 per second
  - **Sender A** (*after 2 seconds*): Hm, why there is no response from my darling, let me transmit again
  - **Channel**: Wow, I have not yet finished transferring your previous gift,….. Omg, new other 100?

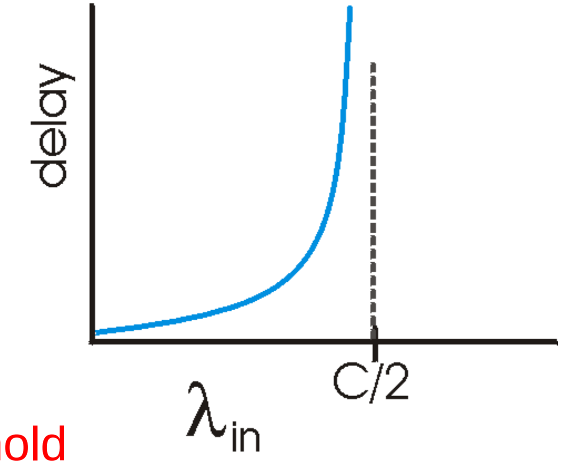
# Causes/costs of congestion: scenario 1

- two senders, two receivers
- one router, infinite buffers
- no retransmission



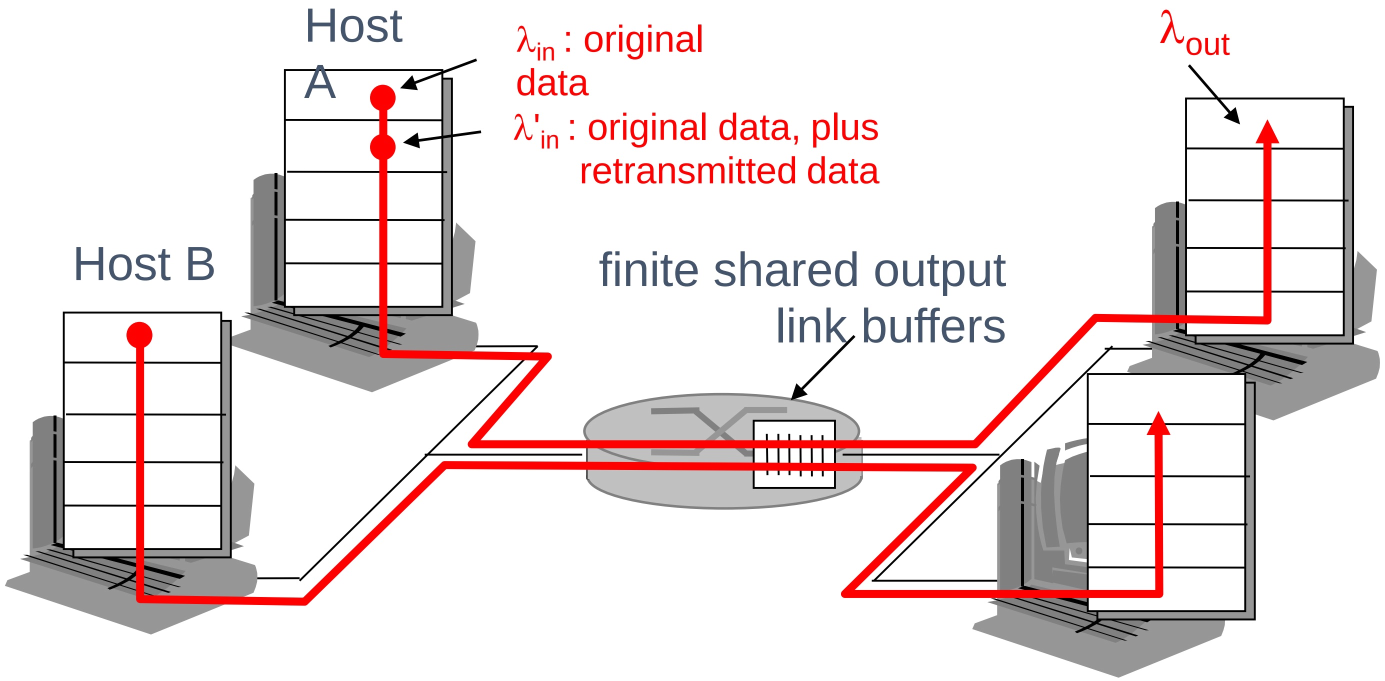

Threshold




- large delays when congested
- maximum achievable throughput

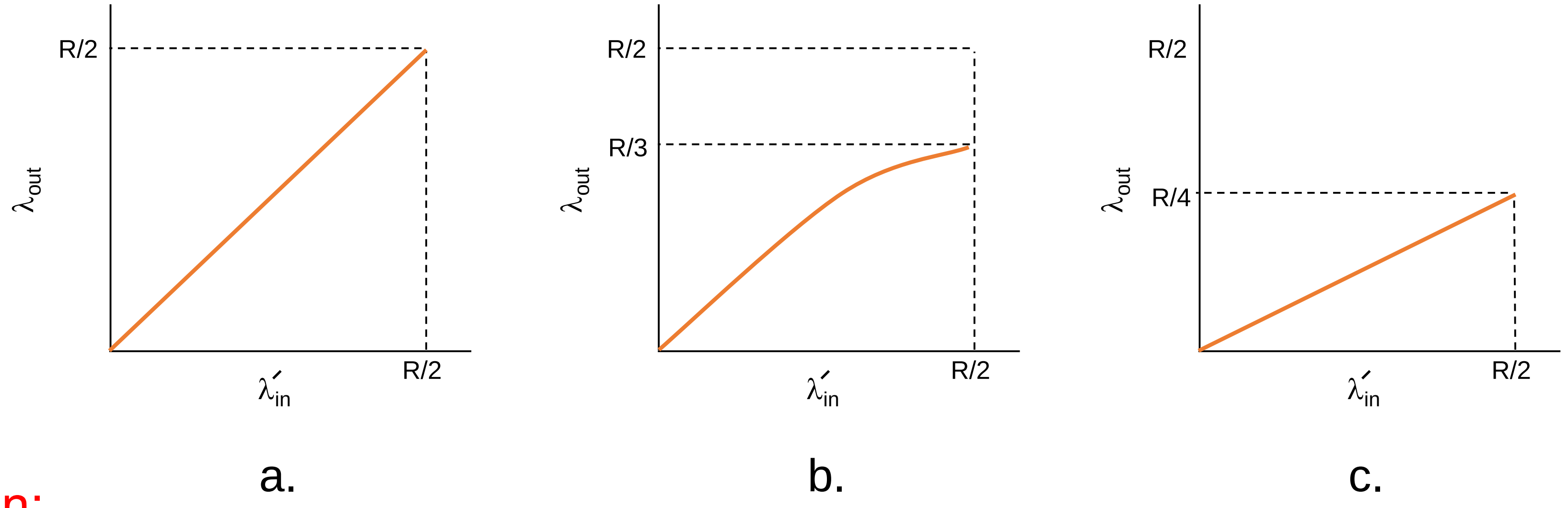
# Causes/costs of congestion: scenario 2

- one router, *finite* buffers
- sender retransmission of lost packet



# Causes/costs of congestion: scenario 2

- always: $\lambda_{in} = \lambda_{out}$ (goodput)
- "perfect" retransmission only when loss: $\lambda'_{in} > \lambda_{out}$
- retransmission of delayed (not lost) packet makes $\lambda_{in}$ larger (than perfect case) for same $\lambda_{out}$



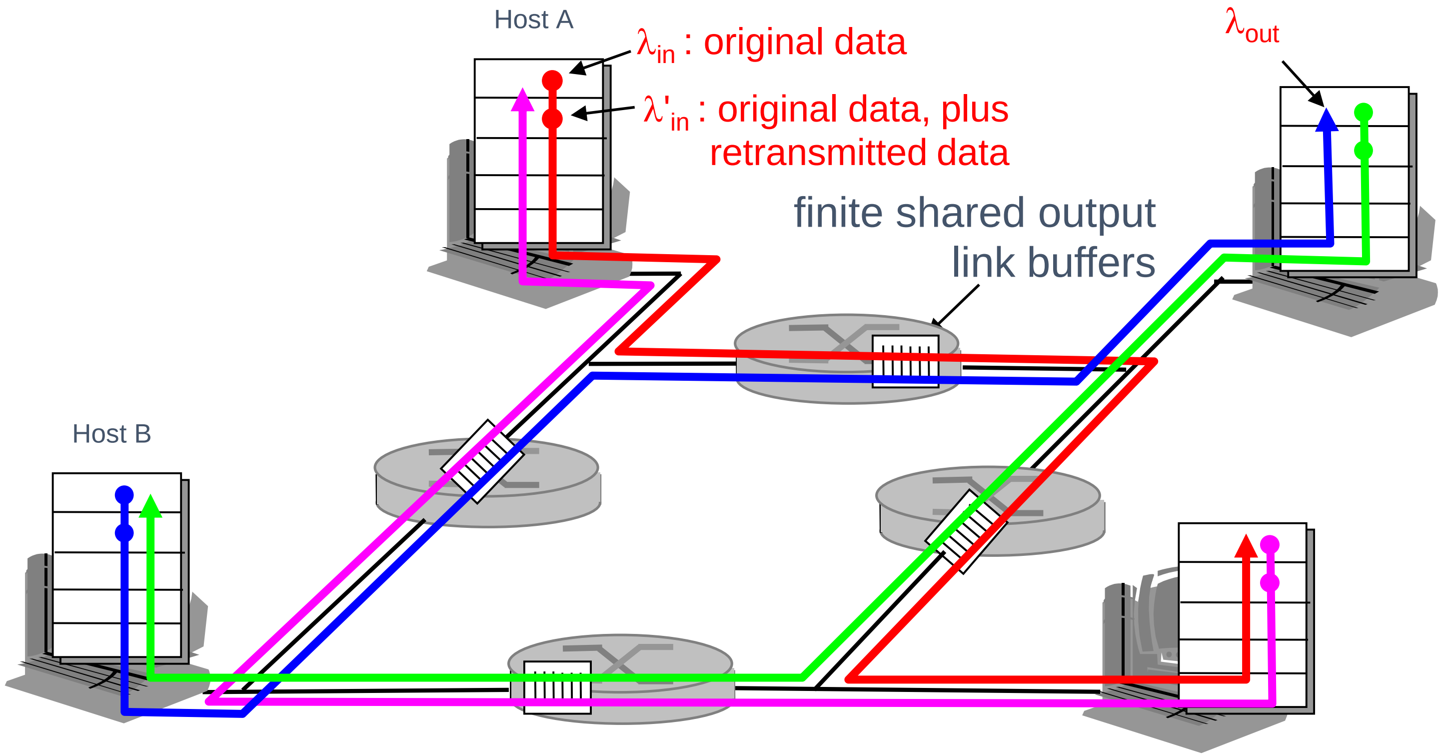

"costs" of congestion:

- more work (retrans) for given "goodput"
- unneeded retransmissions: link carries multiple copies of pkt

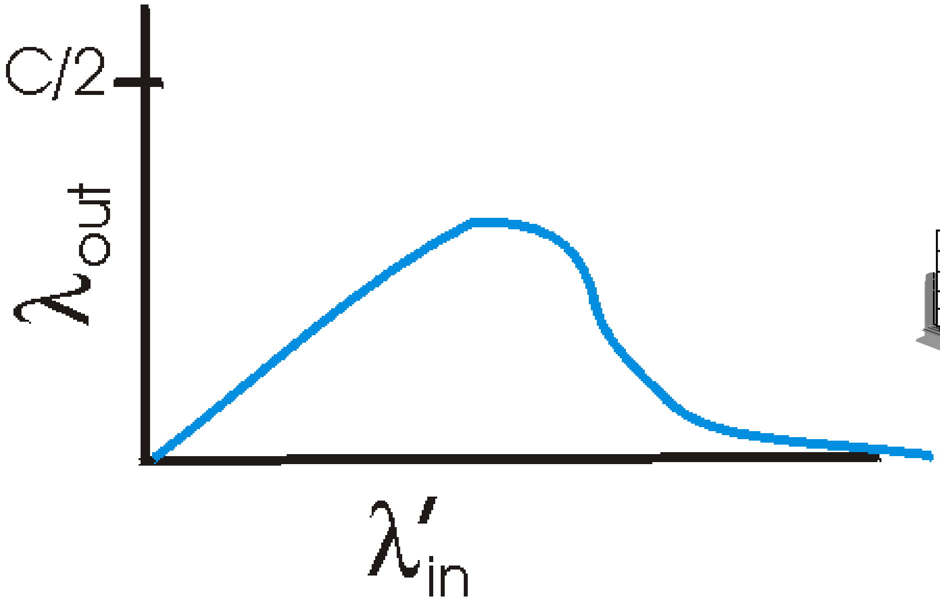
# Causes/costs of congestion: scenario 3

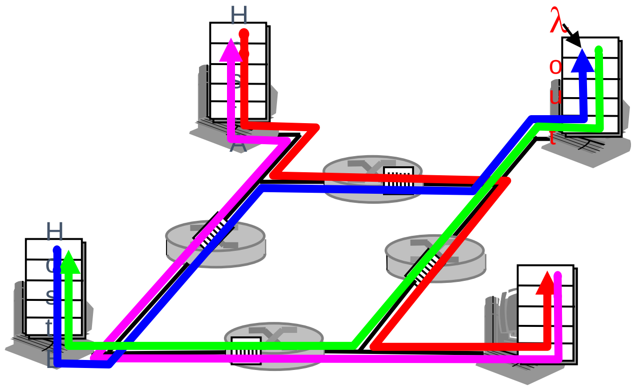
- four senders
- multihop paths
- timeout/retransmit

Q: what happens as $\lambda_{in}$ and $\lambda'_{in}$ increase ?



# Causes/costs of congestion: scenario 3



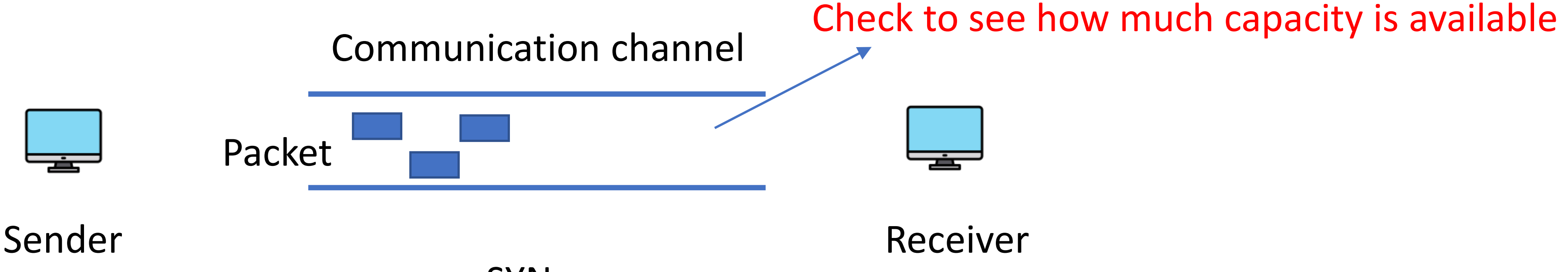



Another "cost" of congestion:

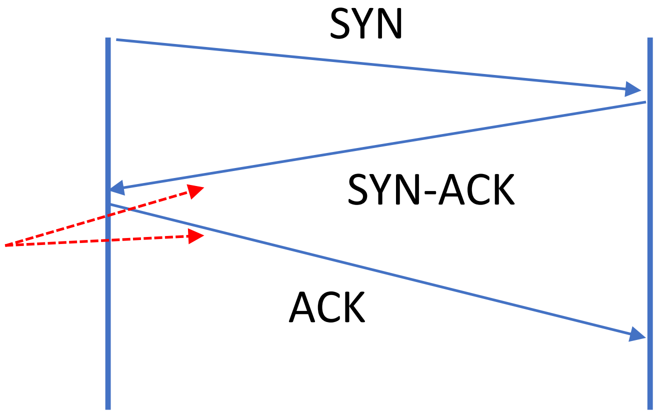
- when packet dropped, any "upstream transmission capacity used for that packet was wasted!

# TCP Congestion Control

- The idea of TCP congestion control is for each source to determine how much capacity is available in the network, so that it knows how many packets it can safely have in transit.




The hosts can use the arrival of an ACK as a signal that one of its packets has left the network, and that it is therefore safe to insert a new packet into the network without adding to the level of congestion.




Three-way handshake

By using ACKs to pace the transmission of packets, TCP is said to be *self-clocking.*

# Approaches towards congestion control

Two approaches towards congestion control

**End-end congestion control:**

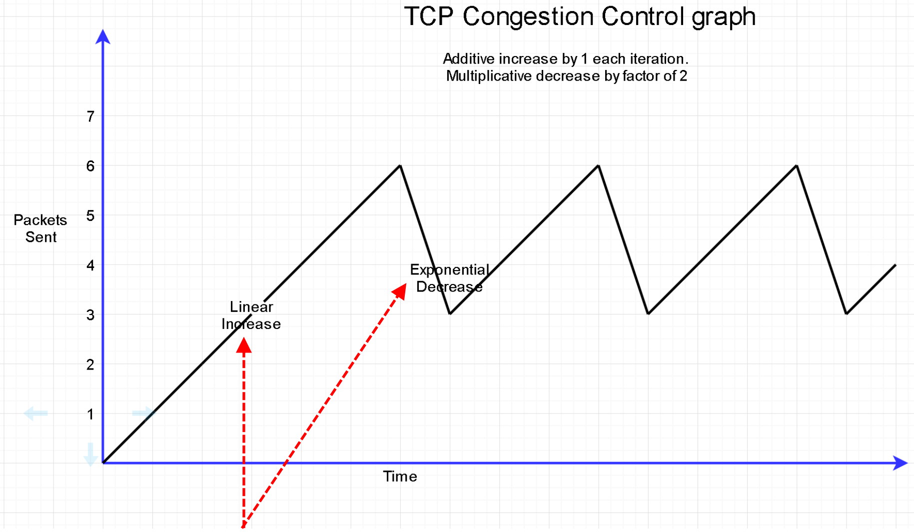
- no explicit feedback from network
- congestion inferred from end-system observed loss, delay
- approach taken by TCP

**Network-assisted congestion control:**

- routers provide feedback to end systems
  - single bit indicating congestion (SNA, DECbit, TCP/IP ECN, ATM)
  - explicit rate sender should send at

# Case study: **Additive Increase Multiplicative Decrease**

- **Additive Increase Multiplicative Decrease (AIMD)**
  - ✓ Solution: Use *CongestionWindow* variable → to limit how much data allowed to have in transit at a given time.
  - ✓ TCP is modified → the maximum number of bytes of unacknowledged data allowed = the minimum of the congestion window and the advertised window
  - ✓ How TCP comes to learn an appropriate value for CongestionWindow?

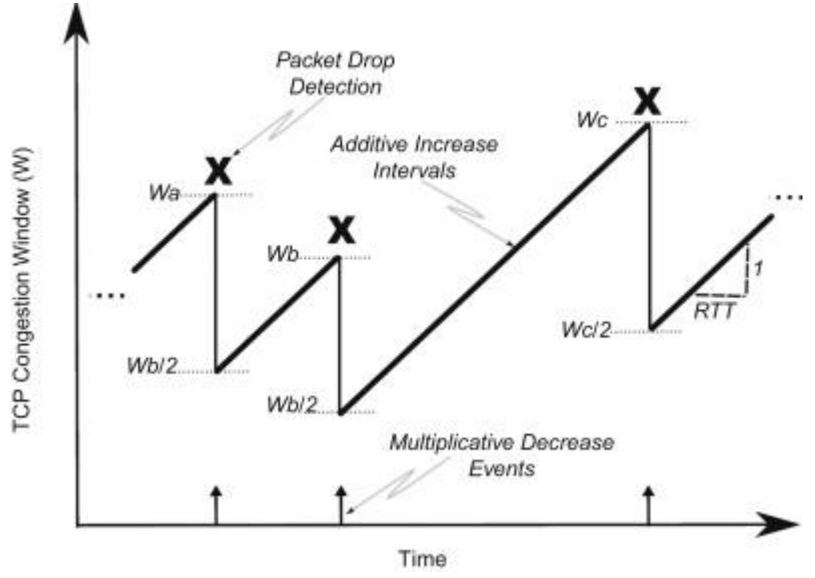



Increase/Decrease events

**Answer**: TCP source sets the CongestionWindow based on the level of congestion it perceives to exist in the network.

# Case study: **Additive Increase Multiplicative Decrease**

- **Additive Increase Multiplicative Decrease**
  - TCP's effective window is revised as follows:
    - MaxWindow = MIN(CongestionWindow, AdvertisedWindow)
    - EffectiveWindow = MaxWindow − (LastByteSent − LastByteAcked).
  - MaxWindow replaces AdvertisedWindow in the calculation of EffectiveWindow.
  - a TCP source is allowed to send no faster than the slowest component—the network or the destination host—can accommodate.



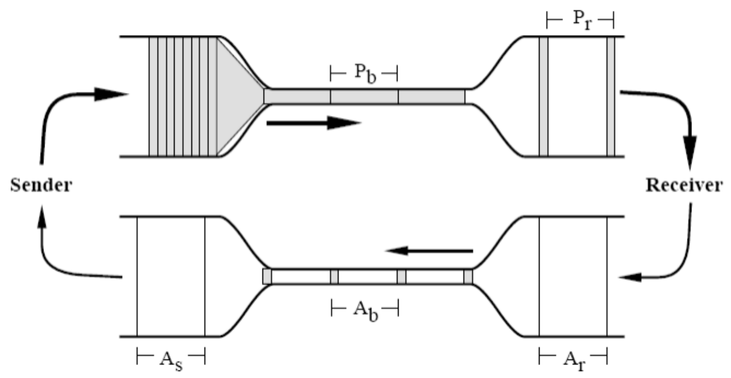

maximum segment size

Increment = MSS × (MSS/CongestionWindow)

CongestionWindow+= Increment

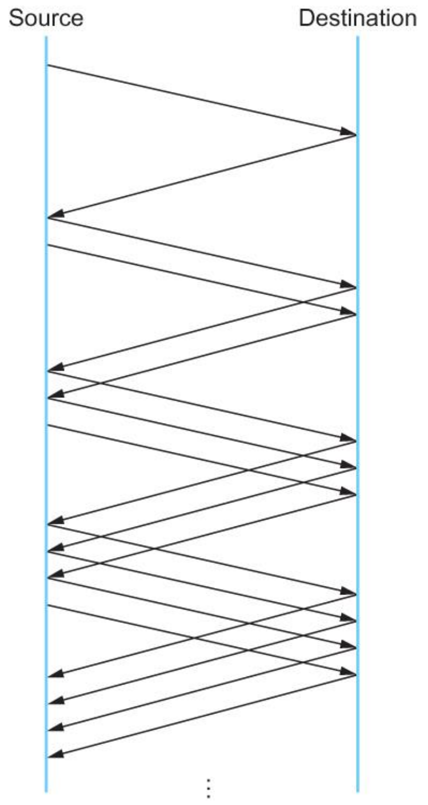
# Case study: **Additive Increase Multiplicative Decrease**

- **Additive Increase Multiplicative Decrease (AIMD)**
  - How does the source determine that the network is congested and that it should decrease the congestion window?
    - ✓ **The answer**: based on several packets are not delivered or timeo results → a packet was dropped due to congestion.
    - ✓ Therefore, TCP interprets timeouts as a sign of congestion and reduces the rate at which it is transmitting.
    - ✓ Each time a timeout occurs, the source sets CongestionWindow t half of its previous value → this is the meaning of "multiplicative decrease"
    - ✓ CongestionWindow is not allowed to fall below the size of a single packet, or in TCP terminology, the *maximum segment size (MSS)*



# Case study: **Additive Increase Multiplicative Decrease**

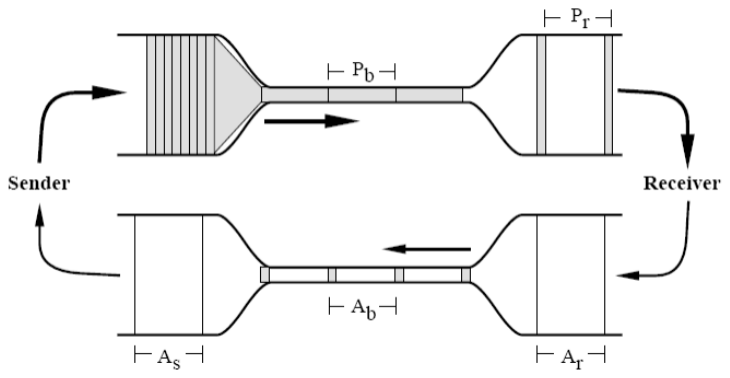
- **Additive Increase Multiplicative Decrease (AIMD)**
  - How does the source determine that the network is congested and that it should decrease the congestion window?
    - ✓ **The answer**: based on several packets are not delivered or timeout results → a packet was dropped due to congestion.
    - For example, suppose the CongestionWindow is currently set to 16 packets. If a loss is detected, CongestionWindow is set to 8.
    - Additional losses cause CongestionWindow to be reduced to 4, then 2, and finally to 1 packet.




Packets in transit during additive increase, with one packet being added each RTT.

# Case study: **Additive Increase Multiplicative Decrease**

- **Additive Increase Multiplicative Decrease (AIMD)**
  - A congestion-control strategy that only decreases the window size is obviously too conservative.
  - We also need to be able to increase the congestion window to take advantage of newly available capacity in the network.




✓ The "additive increase" part of AIMD works as follows.

✓ Every time the source successfully sends a CongestionWindow's worth of packets—that is, each packet sent out during the last RTT has been ACKed—it adds the equivalent of 1 packet to CongestionWindow.

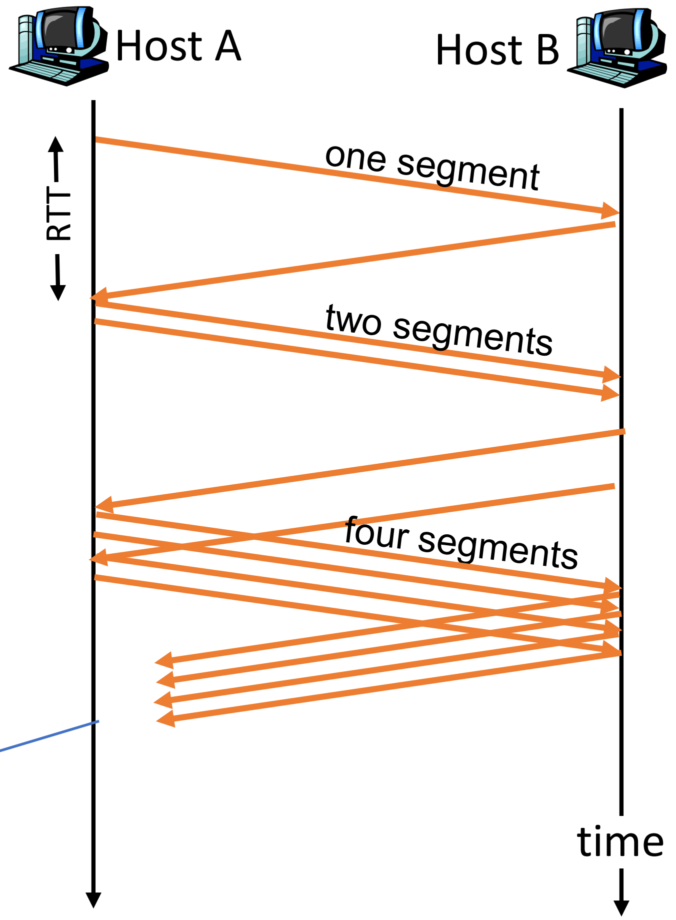
# Case study: TCP Slow Start

- The additive increase mechanism → best usage when the source is operating close to the available capacity of the network, but it takes too long to ramp up a connection when it is starting from scratch.
- TCP therefore provides a second mechanism, ironically called *slow start*, that is used to increase the congestion window rapidly from a cold start.
- Slow start effectively increases the congestion window exponentially, rather than linearly.

- When connection begins, `CongWin` = 1 MSS
  - Example: MSS = 500 bytes & RTT = 200 msec
  - initial rate = 20 kbps
- available bandwidth may be >> MSS/RTT
  - desirable to quickly ramp up to respectable rate
- When connection begins, increase rate exponentially fast until first loss event

# Case study: 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




We increase into a given threshold only

i.e., CongestionThreshold

# Case study: TCP Slow Start

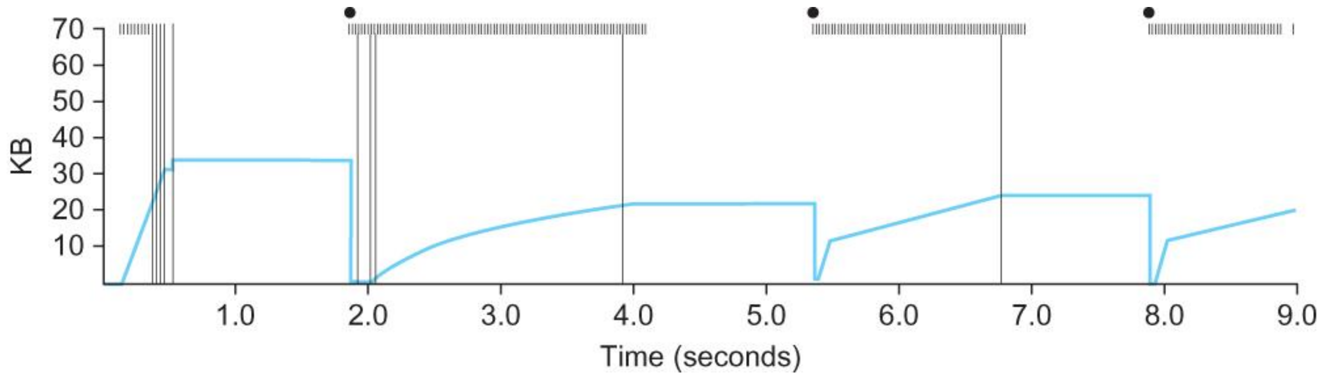
- TCP increases the congestion window as defined by the following code fragment:

```
{
        u_int cw = state->CongestionWindow;
        u_int incr = state->maxseg;
        if (cw > state->CongestionThreshold)
                incr = incr * incr / cw;
        state->CongestionWindow = MIN(cw + incr,  TCP_MAXWIN);
}
```

where state represents the state of a particular TCP connection and TCP MAXWIN defines an upper bound on how large the congestion window is allowed to grow.

# Case study: TCP Slow Start

- Slow Start




Behavior of TCP congestion control

✓ Colored line = value of CongestionWindow over time;

✓ Solid bullets at top of graph = timeouts

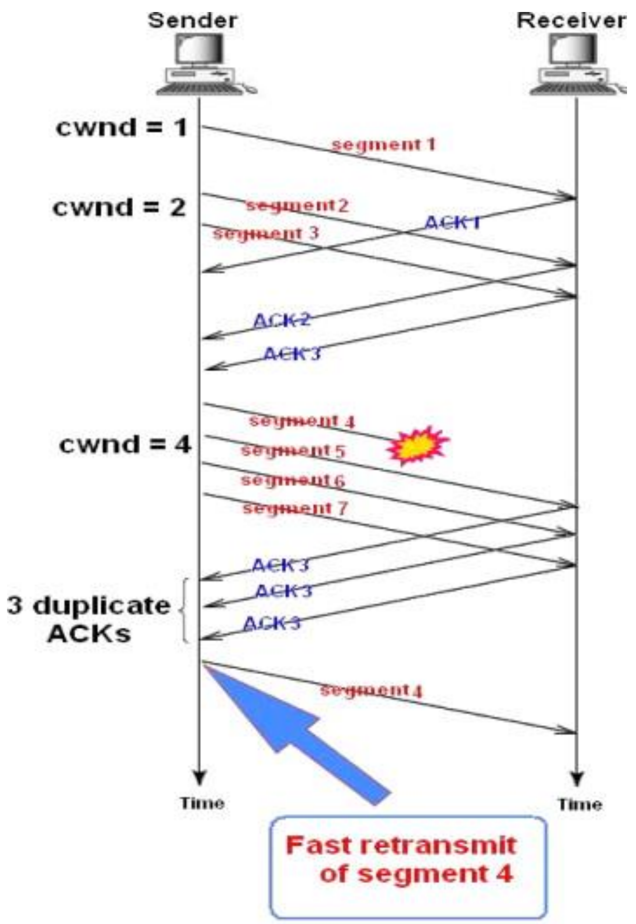
✓ Hash marks at top of graph = time when each packet is transmitted

✓ Vertical bars = time when a packet that was eventually retransmitted was first transmitted.

Cybersecurity Lab

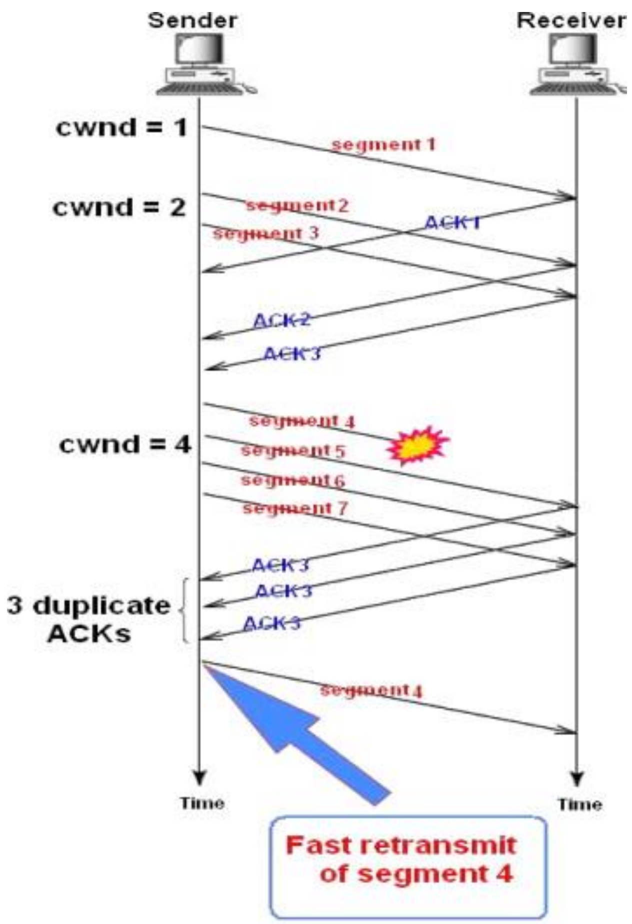
# Case study: Fast Retransmit and Fast Recovery

- TCP timeouts led to long periods of time during which the connection went dead while waiting for a timer to expire.
- Because of this, a new mechanism called *fast retransmit* was added to TCP.
- Fast retransmit is a heuristic that sometimes triggers the retransmission of a dropped packet sooner than the regular timeout mechanism.



# Case study: Fast Retransmit and Fast Recovery

- Every time a data packet arrives at the receiving side, the receiver responds with an acknowledgment → even if this sequence number has already been acknowledged.
- When a packet arrives out of order— that is, TCP cannot yet acknowledge the data the packet contains because earlier data has not yet arrived—TCP resends the same acknowledgment it sent the last time.



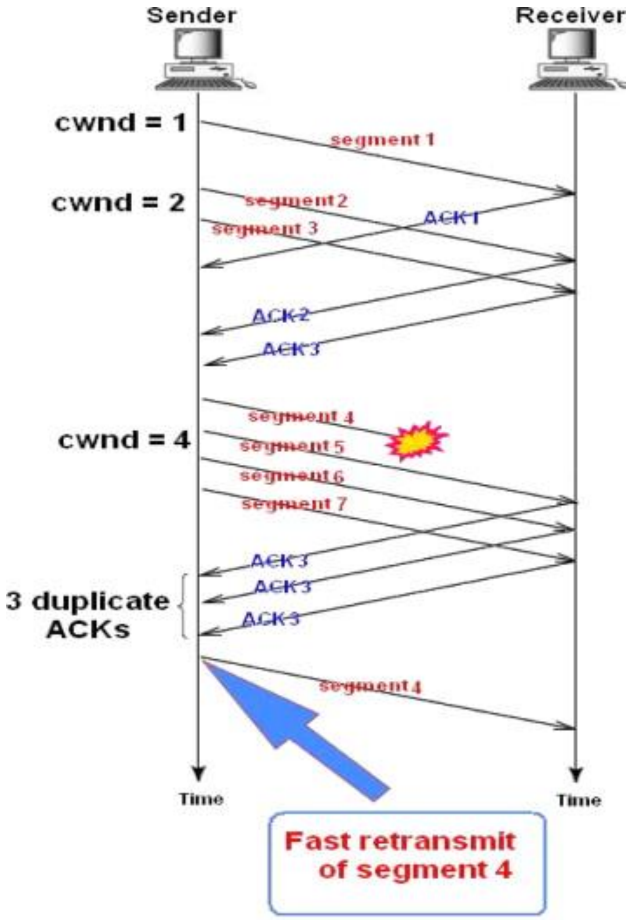

國立中正大學 Cybersecurity Lab

# Case study: Fast Retransmit and Fast Recovery

- This second transmission of the same acknowledgment is called a *duplicate ACK.*

- When the sending side sees a duplicate ACK, it knows that the other side must have received a packet out of order, which suggests that an earlier packet might have been lost.

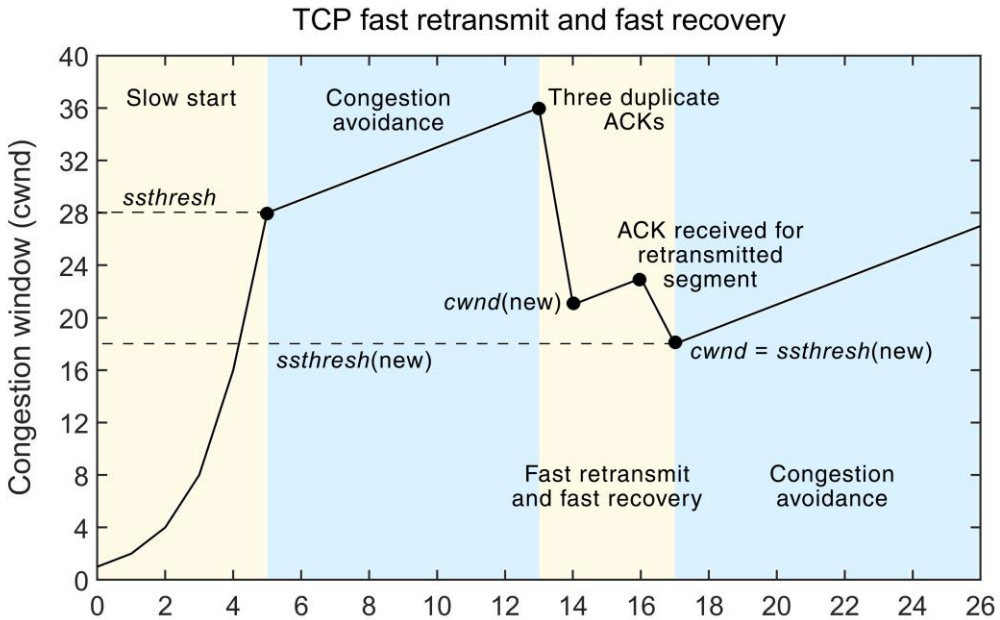
- Since it is also possible that the earlier packet has only been delayed rather than lost, the sender waits until it sees some number of duplicate ACKs and then retransmits the missing packet. In practice, TCP waits until it has seen three duplicate ACKs before retransmitting the packet.



# Case study: Fast Retransmit and Fast Recovery

- When the fast retransmit mechanism signals congestion, rather than drop the congestion window all the way back to one packet and run slow start, it is possible to use the ACKs that are still in the pipe to clock the sending of packets.

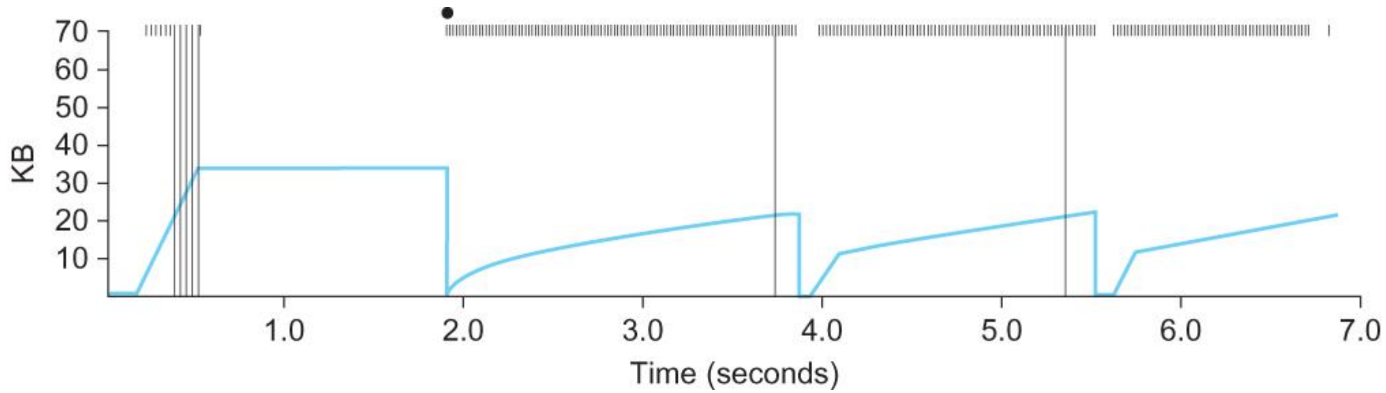
- This mechanism, which is called *fast recovery, effectively* removes the slow start phase that happens between when fast retransmit detects a lost packet and additive increase begins.




國立中正大學 Cybersecurity Lab

# Case study: Fast Retransmit and Fast Recovery

- Illustration




Trace of TCP with fast retransmit.

✓ Colored line = CongestionWindow;

✓ Solid bullet = timeout

✓ Hash marks = time when each packet is transmitted

✓ Vertical bars = time when a packet that was eventually retransmitted was first transmitted.

# 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.

# TCP sender congestion control

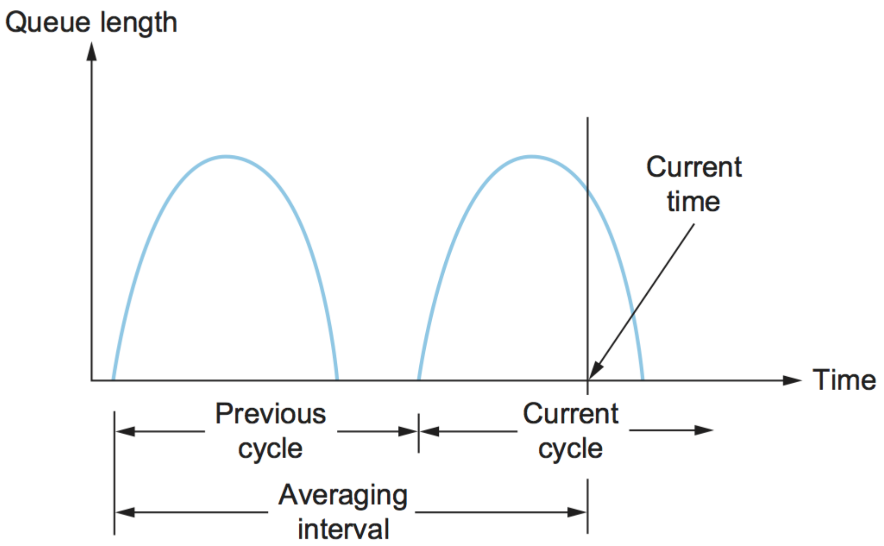
| Event | State | TCP Sender Action | Commentary |
|---|---|---|---|
| ACK receipt for previously unacked data | Slow Start (SS) | CongWin = CongWin + MSS, If (CongWin > Threshold) set state to “Congestion Avoidance” | Resulting in a doubling of CongWin every RTT |
| ACK receipt for previously unacked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

# Congestion avoidance mechanisms

- TCP's strategy
  - ✓ Control congestion once it happens
  - ✓ Repeatedly increase load to find the point where the congestion occurs, and back off
- Alternative strategy
  - ✓ Predict when congestion is about to happen
  - ✓ R educe the rate at which hosts send data just before packets start being discarded
  - ✓ This strategy is so-called *congestion avoidance*, instead of *congestion control*
- Two methods
  - ✓ Router-center: Dec-bit, RED Gateways
  - ✓ Host-center: TCP Vegas

# Congestion Avoidance Mechanism

- **DEC Bit**
  - Monitor router queue length over last busy-idle cycle plus current cycle
  - Set congestion bit if the average queue length > 1
  - Balance throughput against delays
    - ✓ The destination host then copies this congestion bit into the ACK it sends back to the source.
    - ✓ The source records how many packets results in set bit
      - ➢ less than 50% of last window worth have bit set → increase *CongWin* by 1 packet
      - ➢ 50% or more of last window worth have bit set →decrease *CongWin* by 0.875 times

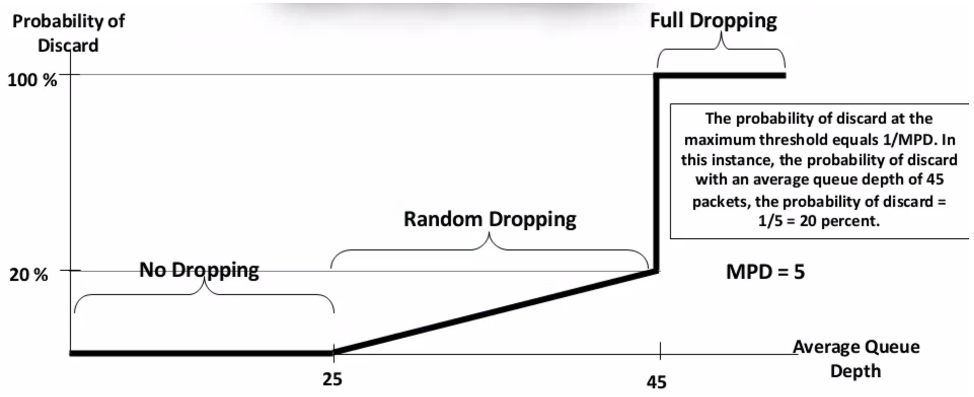



Computing average queue length at a router

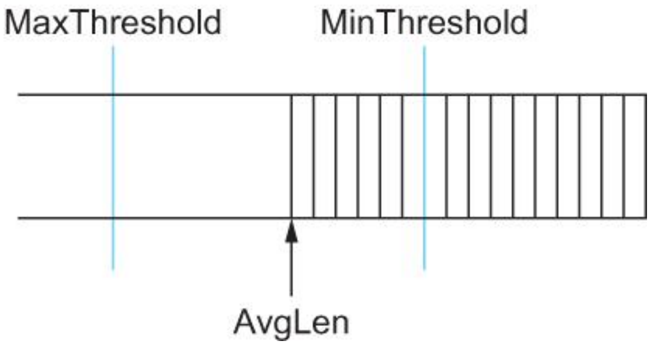
The "increase by 1, decrease by 0.875" rule was selected because additive increase/multiplicative decrease makes the mechanism stable

# Congestion Avoidance Mechanism

- **Random Early Detection (RED)**
  - Similar to the DECbit scheme in that each router is programmed to monitor its own queue length
  - When router detects that congestion is imminent
    - ✓ Notify the source to adjust its congestion window.
  - Notification is implicit
    - ✓ Just drop the packet (TCP will timeout)
    - ✓ Could make explicit by marking the packet
  - Early random drop
    - ✓ Rather wait for queue become full, drop each arriving packet with some drop probability whenever the queue length exceeds drop level
    - ✓ Target is to cause the source to slow down Early to avoid congestion



# Congestion Avoidance Mechanism




- **Random Early Detection (RED)**
  - Compute average queue length

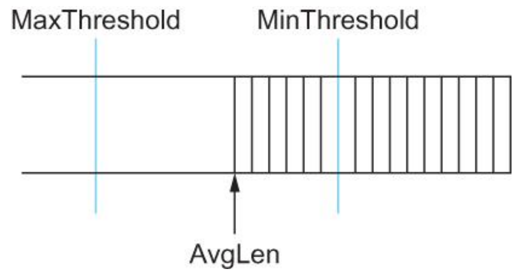
    $$AvgLen = (1 - Weight) \times AvgLen + Weight \times SampleLen$$

    where 0 < Weight < 1 and SampleLen is the length of the queue when a sample measurement is made.

  - In most software implementations, the queue length is measured every time a new packet arrives at the gateway.
  - In hardware, it might be calculated at some fixed sampling interval.

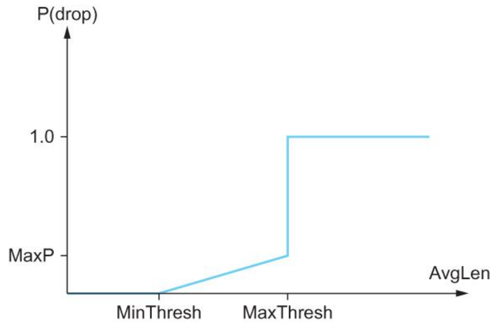
# Congestion Avoidance Mechanism

- **Random Early Detection (RED)**
  - Two queue length threshold that trigger certain activities: MinThreshold and MaxThreshold.
  - When a packet arrives at the gateway, RED compares the current AvgLen with these two thresholds, according to the following rules:
    - ✓ if AvgLen $\leq$ MinThreshold
      - → queue the packet
    - ✓ if MinThreshold < AvgLen < MaxThreshold
      - → calculate probability P
      - → drop the arriving packet with probability P
    - ✓ if MaxThreshold $\leq$ AvgLen
      - → drop the arriving packet



RED thresholds on a FIFO queue

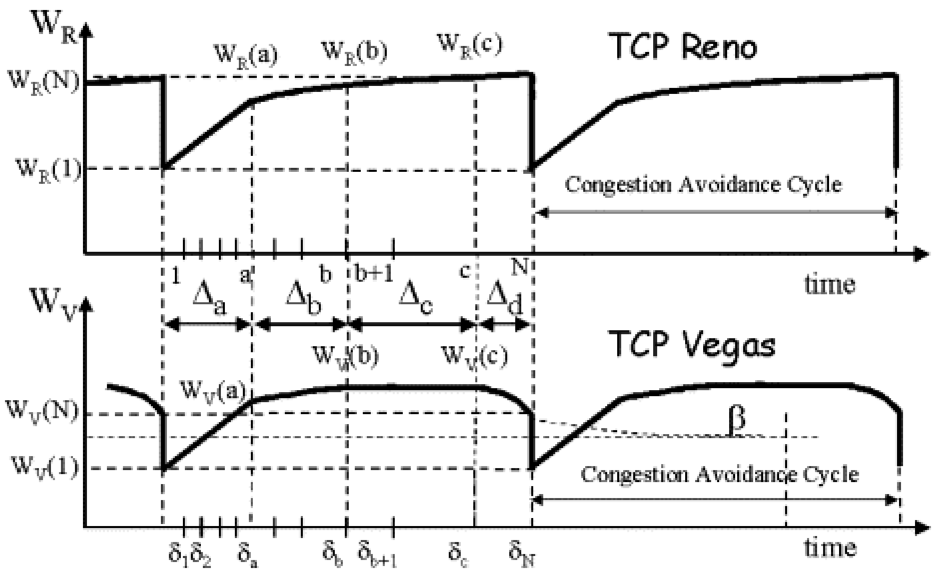
  - Compute Probability P
    - ✓ TempP = MaxP × (AvgLen − MinThreshold)/(MaxThreshold − MinThreshold)
    - ✓ P = TempP/(1 − count × TempP)



Drop probability function for RED

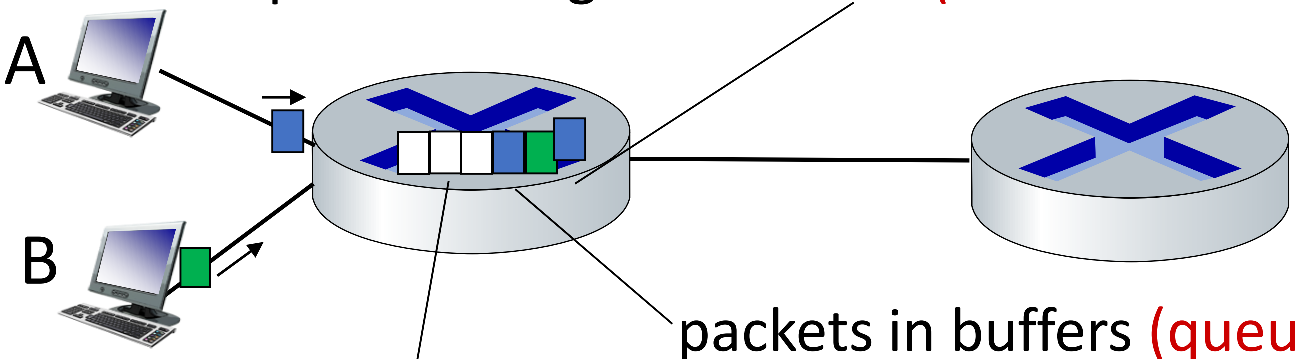
# Congestion avoidance mechanism

- TCP Reno/Vegas
  - Not much in use but continues to be studied
  - Goals
    - Detect congestion in routers between source and destination before packet loos occurs
    - Lower the rate linearly when this imminent packet loss is detected
      - Predicted by observing the RTTs
      - Longer RTT than expected == greater congestion in routers

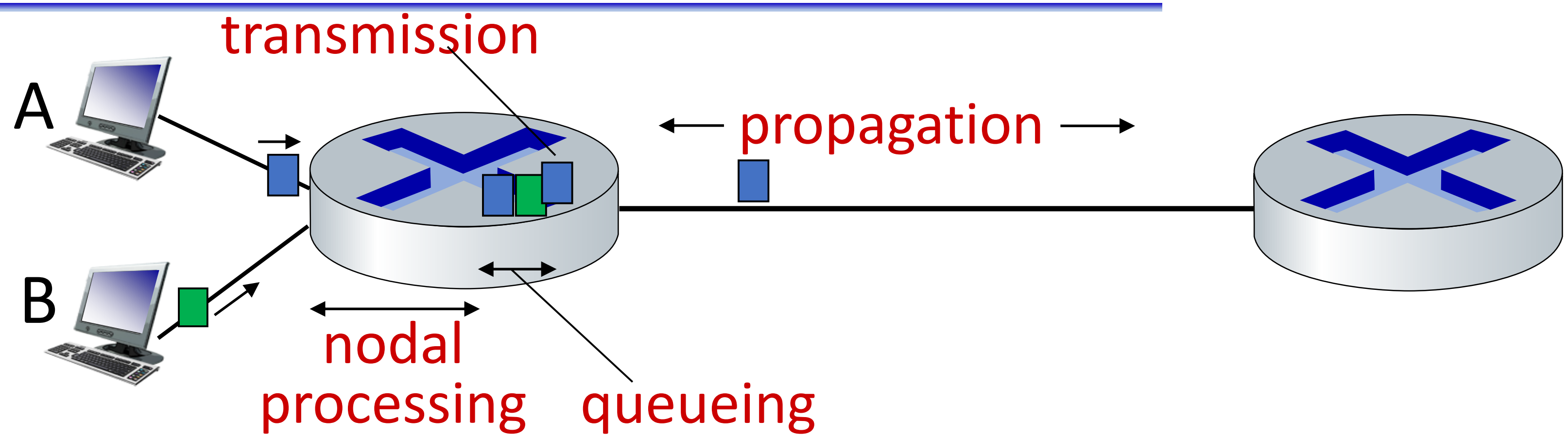


# How do packet delay and loss occur?

- packets *queue* in router buffers, waiting for turn for transmission
  - queue length grows when arrival rate to link (temporarily) exceeds output link capacity
- packet *loss* occurs when memory to hold queued packets fills up



# Packet delay: four sources




$$d_{\text{nodal}} = d_{\text{proc}} + d_{\text{queue}} + d_{\text{trans}} + d_{\text{prop}}$$

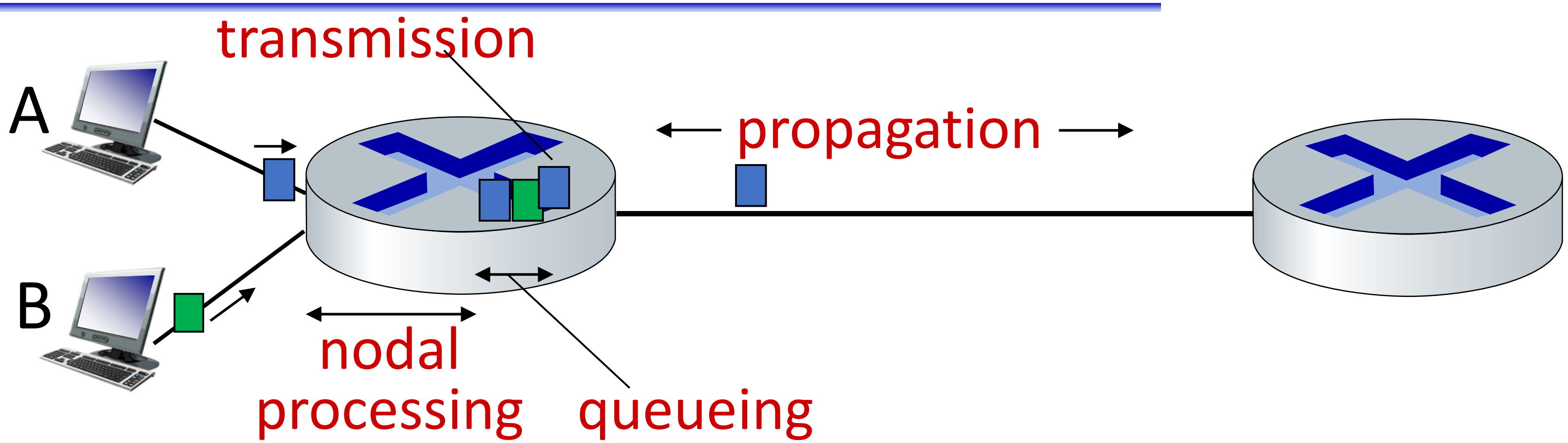
$d_{\text{proc}}$: nodal processing

- check bit errors
- determine output link
- typically < microsecs

$d_{\text{queue}}$: queueing delay

- time waiting at output link for transmission
- depends on congestion level of router

# Packet delay: four sources




$$d_{nodal} = d_{proc} + d_{queue} + d_{trans} + d_{prop}$$

## $d_{trans}$: transmission delay:

- $L$: packet length (bits)
- $R$: link *transmission rate (bps)*
- $d_{trans} = L/R$

## $d_{prop}$: propagation delay:

- $d$: length of physical link
- $s$: propagation speed (~2x$10^8$ m/sec)
- $d_{prop} = d/s$

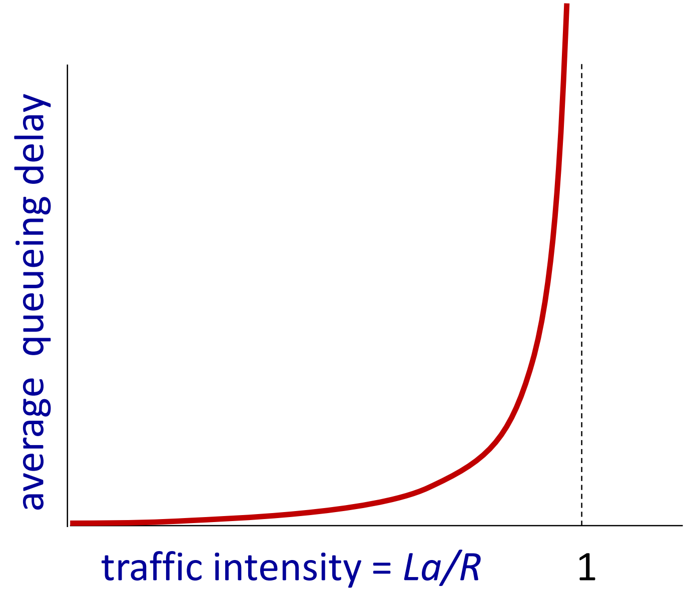
$d_{trans}$ and $d_{prop}$
*very* different

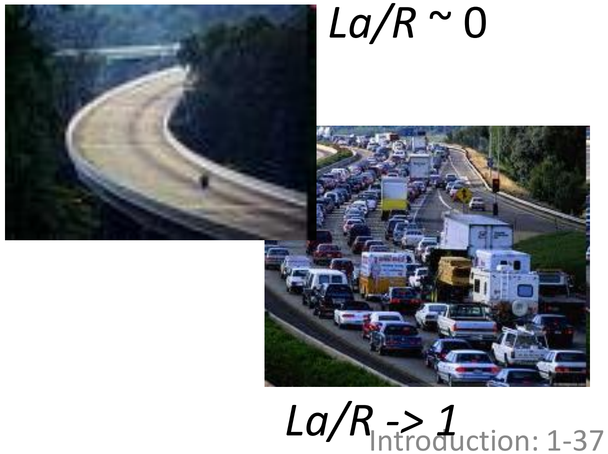
# Packet queueing delay

- *a:* average packet arrival rate
- *L:* packet length (bits)
- *R:* link bandwidth (bit transmission rate)

$$\frac{L \cdot a}{R} : \quad \frac{\text{arrival rate of bits}}{\text{service rate of bits}}$$

*"traffic intensity"*

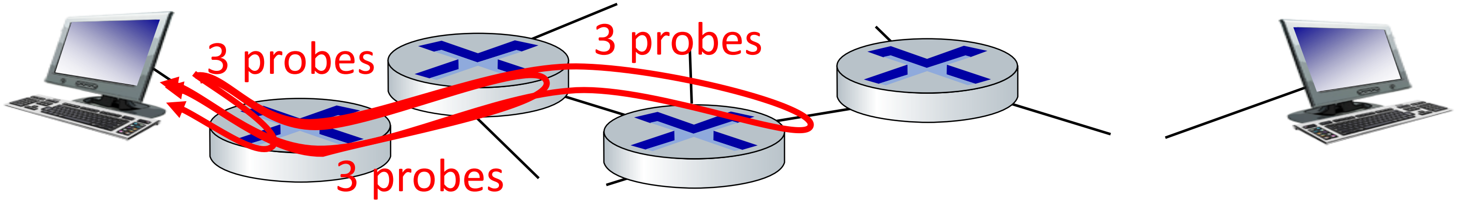



- *La/R* ~ 0: avg. queueing delay small
- *La/R* -> 1: avg. queueing delay large
- *La/R* > 1: more "work" arriving is more than can be serviced - average delay infinite!

# "Real" Internet delays and routes

- what do "real" Internet delay & loss look like?
- **traceroute** program: provides delay measurement from source to router along end-end Internet path towards destination. For all *i:*
  - sends three packets that will reach router *i* on path towards destination (with time-to-live field value of *i*)
  - router *i* will return packets to sender
  - sender measures time interval between transmission and reply




國立中正大學 Cybersecurity Lab

# Real Internet delays and routes

## traceroute: gaia.cs.umass.edu to www.eurecom.fr

3 delay measurements from
gaia.cs.umass.edu to cs-gw.cs.umass.edu

3 delay measurements
to border1-rt-fa5-1-0.gw.umass.edu

```
1  cs-gw (128.119.240.254)  1 ms  1 ms  2 ms
2  border1-rt-fa5-1-0.gw.umass.edu (128.119.3.145)  1 ms  1 ms  2 ms
3  cht-vbns.gw.umass.edu (128.119.3.130)  6 ms 5 ms 5 ms
4  jn1-at1-0-0-19.wor.vbns.net (204.147.132.129)  16 ms 11 ms 13 ms
5  jn1-so7-0-0-0.wae.vbns.net (204.147.136.136)  21 ms 18 ms 18 ms
6  abilene-vbns.abilene.ucaid.edu (198.32.11.9)  22 ms  18 ms  22 ms
7  nycm-wash.abilene.ucaid.edu (198.32.8.46)  22 ms  22 ms  22 ms
8  62.40.103.253 (62.40.103.253)  104 ms 109 ms 106 ms
9  de2-1.de1.de.geant.net (62.40.96.129)  109 ms 102 ms 104 ms
10  de.fr1.fr.geant.net (62.40.96.50)  113 ms 121 ms 114 ms
11  renater-gw.fr1.fr.geant.net (62.40.103.54)  112 ms  114 ms  112 ms
12  nio-n2.cssi.renater.fr (193.51.206.13)  111 ms  114 ms  116 ms
13  nice.cssi.renater.fr (195.220.98.102)  123 ms  125 ms  124 ms
14  r3t2-nice.cssi.renater.fr (195.220.98.110)  126 ms  126 ms  124 ms
15  eurecom-valbonne.r3t2.ft.net (193.48.50.54)  135 ms  128 ms  133 ms
16  194.214.211.25 (194.214.211.25)  126 ms  128 ms  126 ms
17  * * *
18  * * *
19  fantasia.eurecom.fr (193.55.113.142)  132 ms  128 ms  136 ms
```

trans-oceanic link

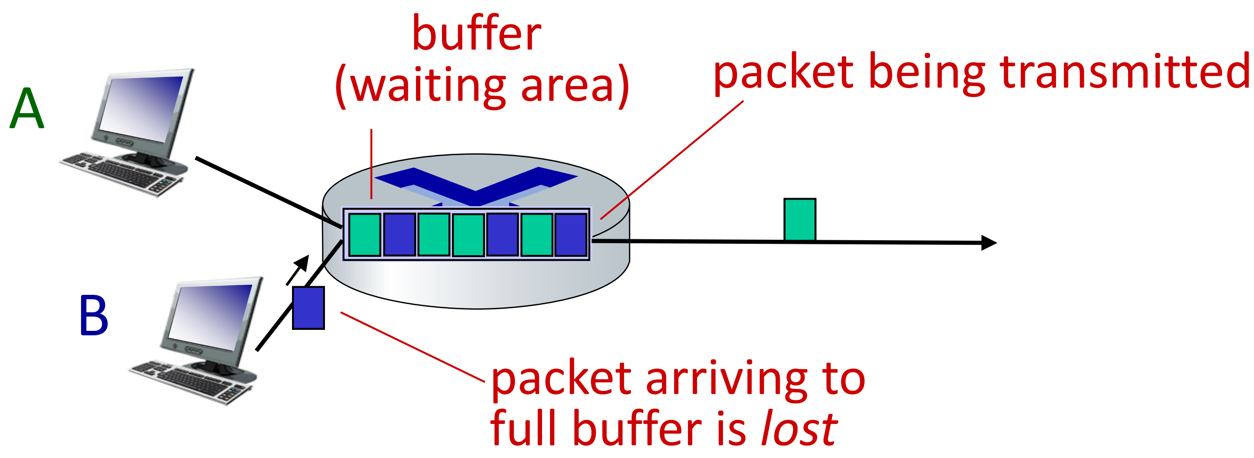
looks like delays *decrease*! Why?

* means no response (probe lost, router not replying)

* Do some traceroutes from exotic countries at www.traceroute.org

# Packet loss

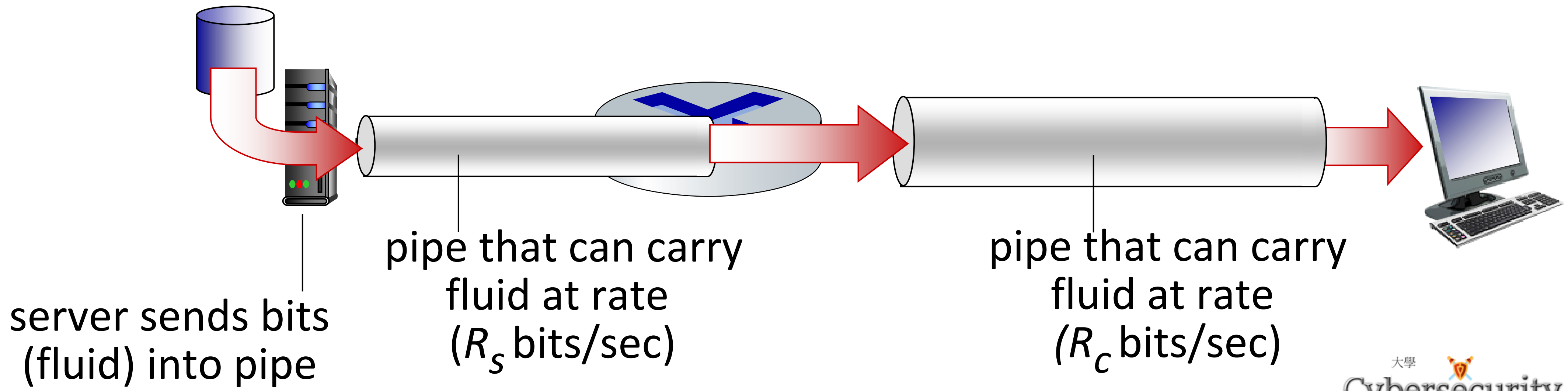
- queue (aka buffer) preceding link in buffer has finite capacity
- packet arriving to full queue dropped (aka lost)
- lost packet may be retransmitted by previous node, by source end system, or not at all




* Check out the Java applet for an interactive animation (on publisher's website) of queuing and loss

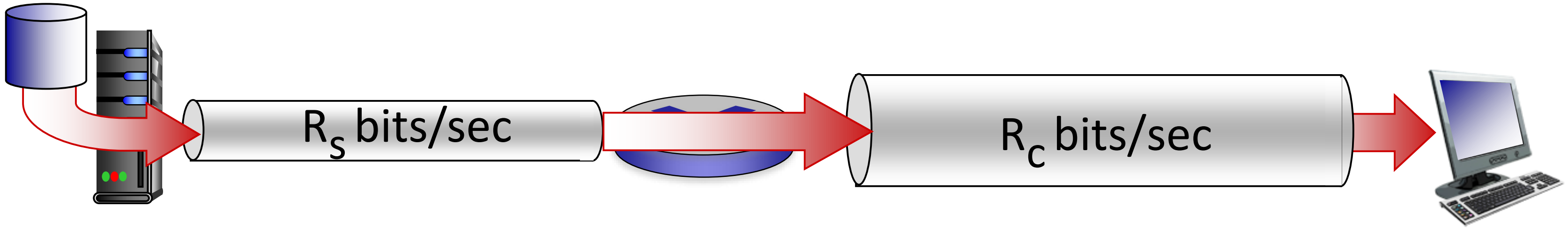
# Throughput

- *throughput:* rate (bits/time unit) at which bits are being sent from sender to receiver
  - *instantaneous:* rate at given point in time
  - *average:* rate over longer period of time

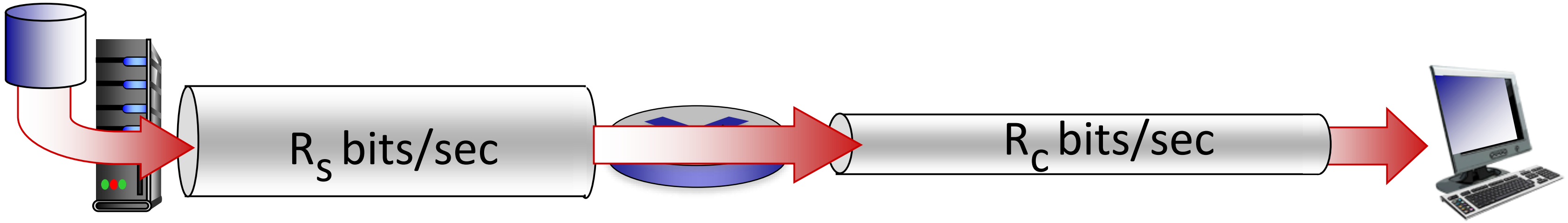


# Throughput

$R_s < R_c$ What is average end-end throughput?



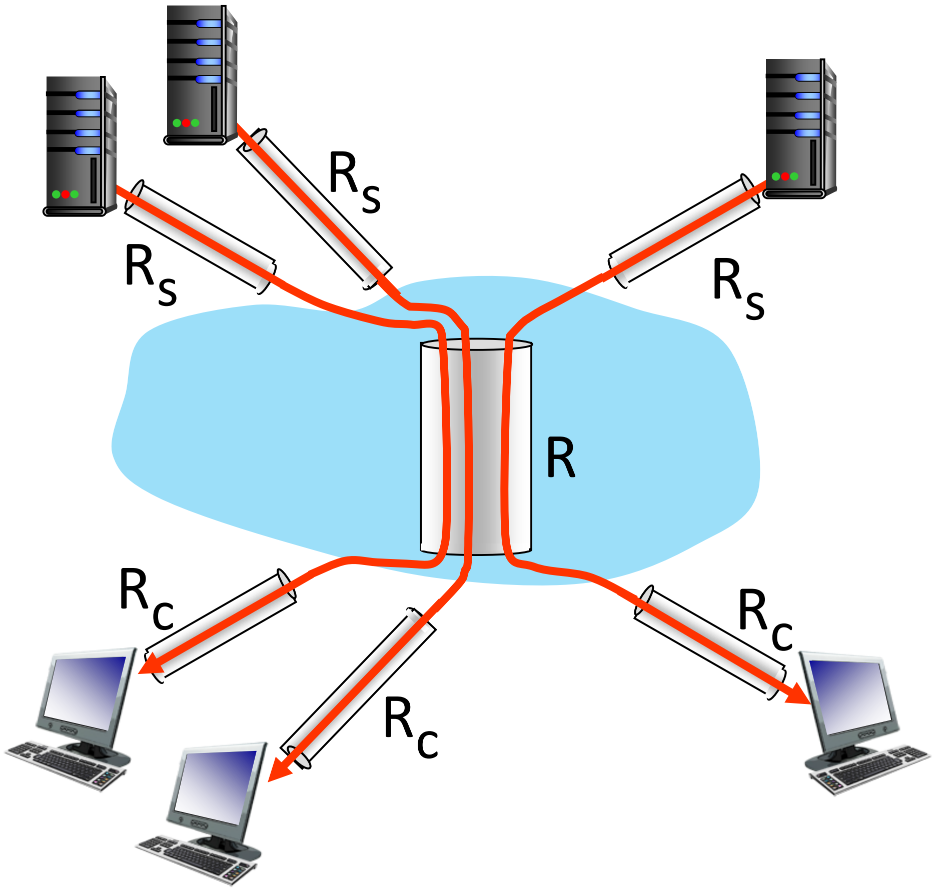

$R_s > R_c$ What is average end-end throughput?




*bottleneck link*

link on end-end path that constrains end-end throughput

# Throughput: network scenario




10 connections (fairly) share backbone bottleneck link *R* bits/sec

- per-connection end-end throughput: $min(R_c,R_s,R/10)$
- in practice: $R_c$ or $R_s$ is often bottleneck

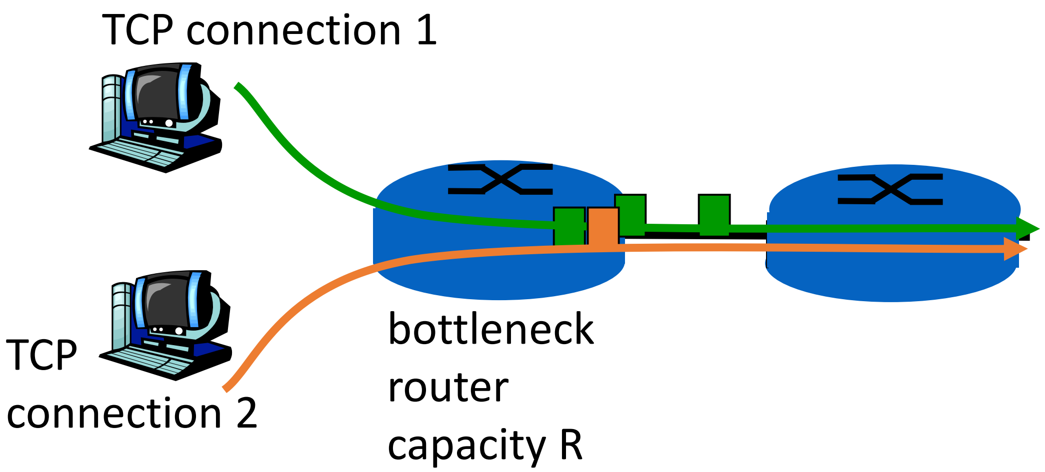
* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/

# TCP throughput

- What’s the average throughout ot TCP as a function of window size and RTT?
  - Ignore slow start
- Let W be the window size when loss occurs.
- When window is W, throughput is W/RTT
- Just after loss, window drops to W/2, throughput to W/2RTT.
- Average throughout: .75 W/RTT

# TCP Fairness

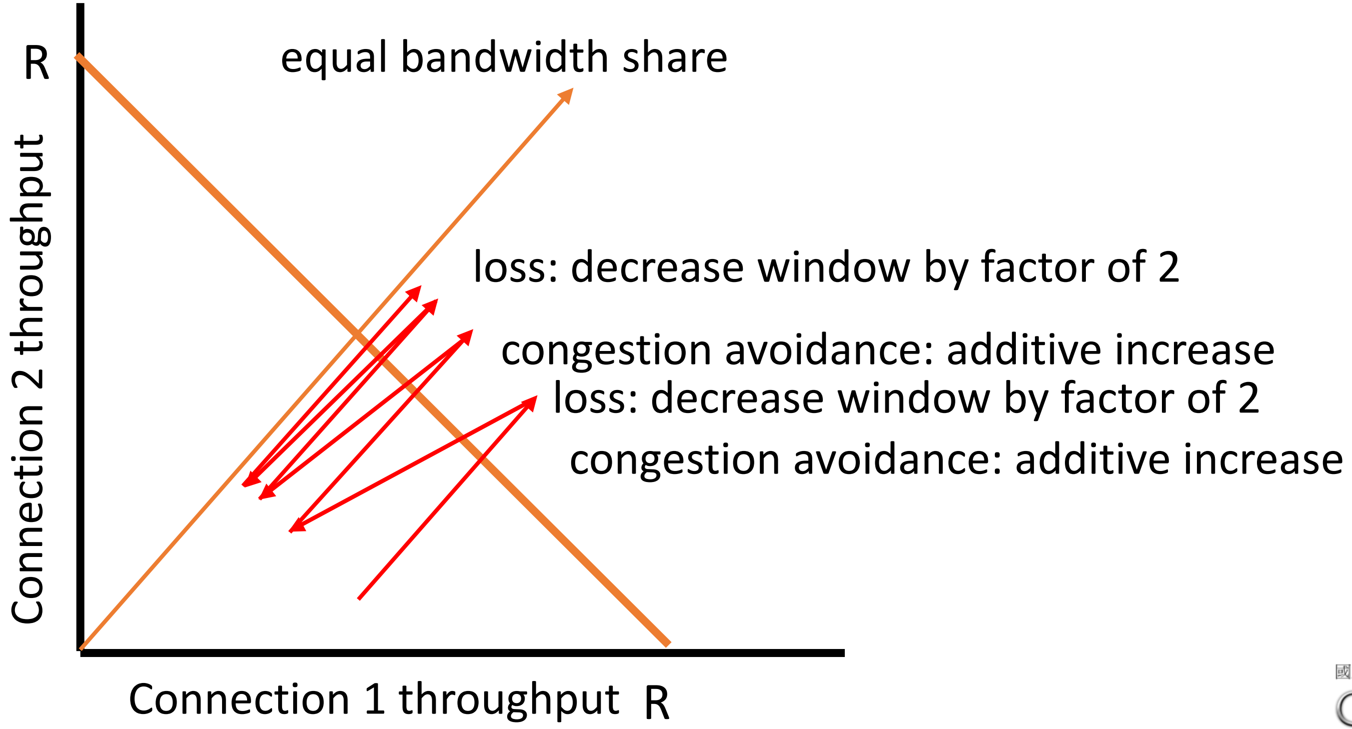
Fairness goal: if K TCP sessions share same bottleneck link of bandwidth R, each should have average rate of R/K



# Why is TCP fair?

Two competing sessions:

- Additive increase gives slope of 1, as throughout increases
- multiplicative decrease decreases throughput proportionally




國立中正大學 Cybersecurity Lab

# Fairness (more)

## Fairness and UDP

- Multimedia apps often do not use TCP
  - do not want rate throttled by congestion control
- Instead use UDP:
  - pump audio/video at constant rate, tolerate packet loss
- Research area: TCP friendly

## Fairness and parallel TCP connections

- nothing prevents app from opening parallel cnctions between 2 hosts.
- Web browsers do this
- Example: link of rate R supporting 9 cnctions;
  - new app asks for 1 TCP, gets rate R/10
  - new app asks for 11 TCPs, gets R/2 !

# Delay modeling

Q: How long does it take to receive an object from a Web server after sending a request?

Ignoring congestion, delay is influenced by:

- TCP connection establishment
- data transmission delay
- slow start

Notation, assumptions:

- Assume one link between client and server of rate R
- S: MSS (bits)
- O: object size (bits)
- no retransmissions (no loss, no corruption)

Window size:

- First assume: fixed congestion window, W segments
- Then dynamic window, modeling slow start

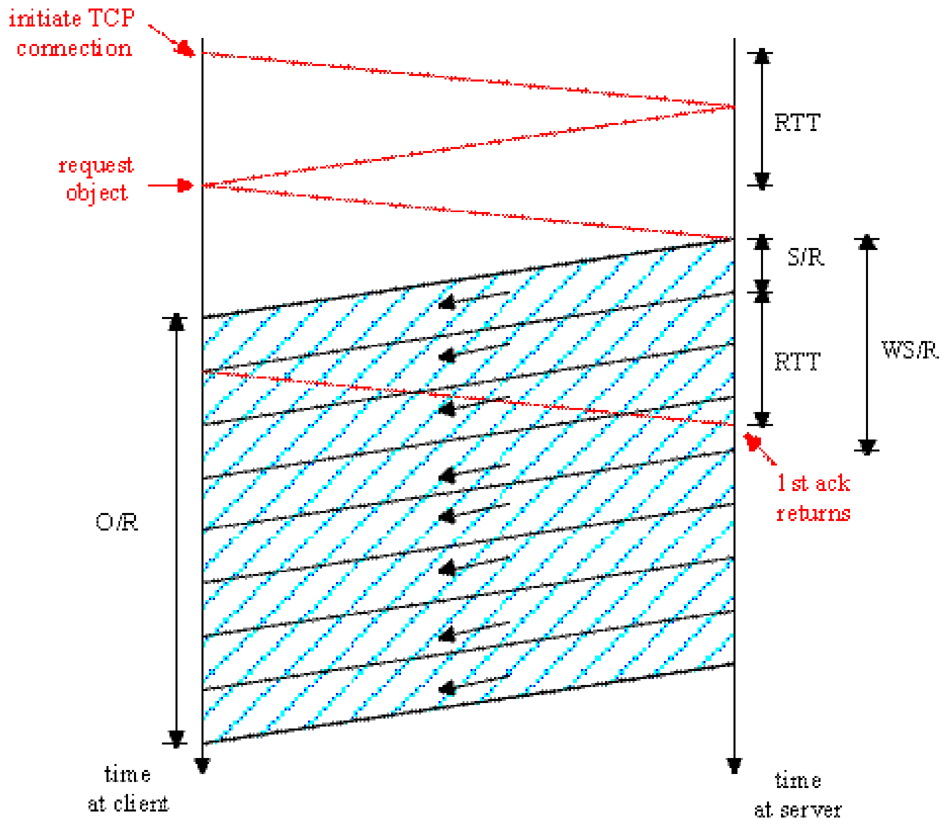
# Fixed congestion window (1)

## First case:

WS/R > RTT + S/R: ACK for first segment in window returns before window's worth of data sent

delay = 2RTT + O/R

R: Rate
S: MSS (bits)
O: object size (bits)



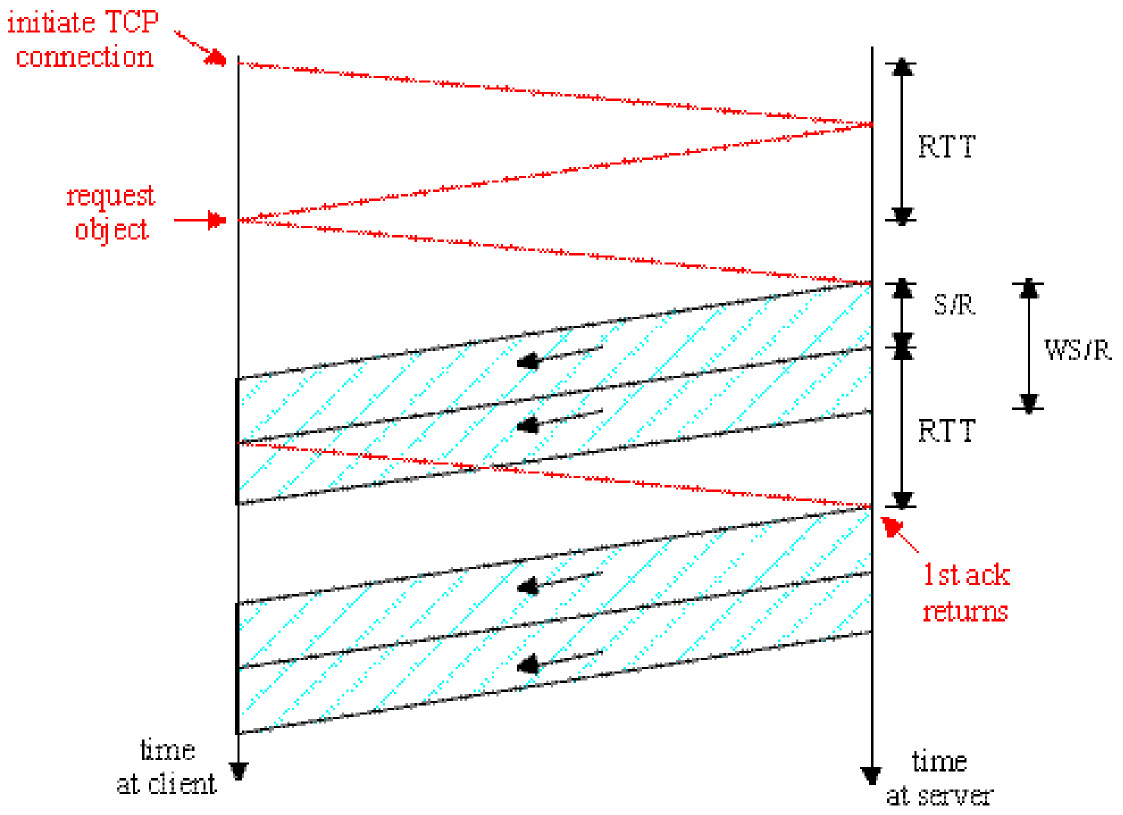
# Fixed congestion window (2)

## Second case:

- WS/R < RTT + S/R: wait for ACK after sending window's worth of data sent

delay = 2RTT + O/R
+ (K-1)[S/R + RTT - WS/R]

Where K=O/WS



# TCP Delay Modeling: Slow Start (1)

Now suppose window grows according to slow start

Will show that the delay for one object is:

$$\boxed{Latency = 2RTT + \frac{O}{R} + P\left[RTT + \frac{S}{R}\right] - (2^P - 1)\frac{S}{R}}$$

where *P* is the number of times TCP idles at server:

$$P = \min\{Q, K-1\}$$

- where Q is the number of times the server idles if the object were of infinite size.

- and K is the number of windows that cover the object.

# TCP Delay Modeling: Slow Start (2)

Delay components:

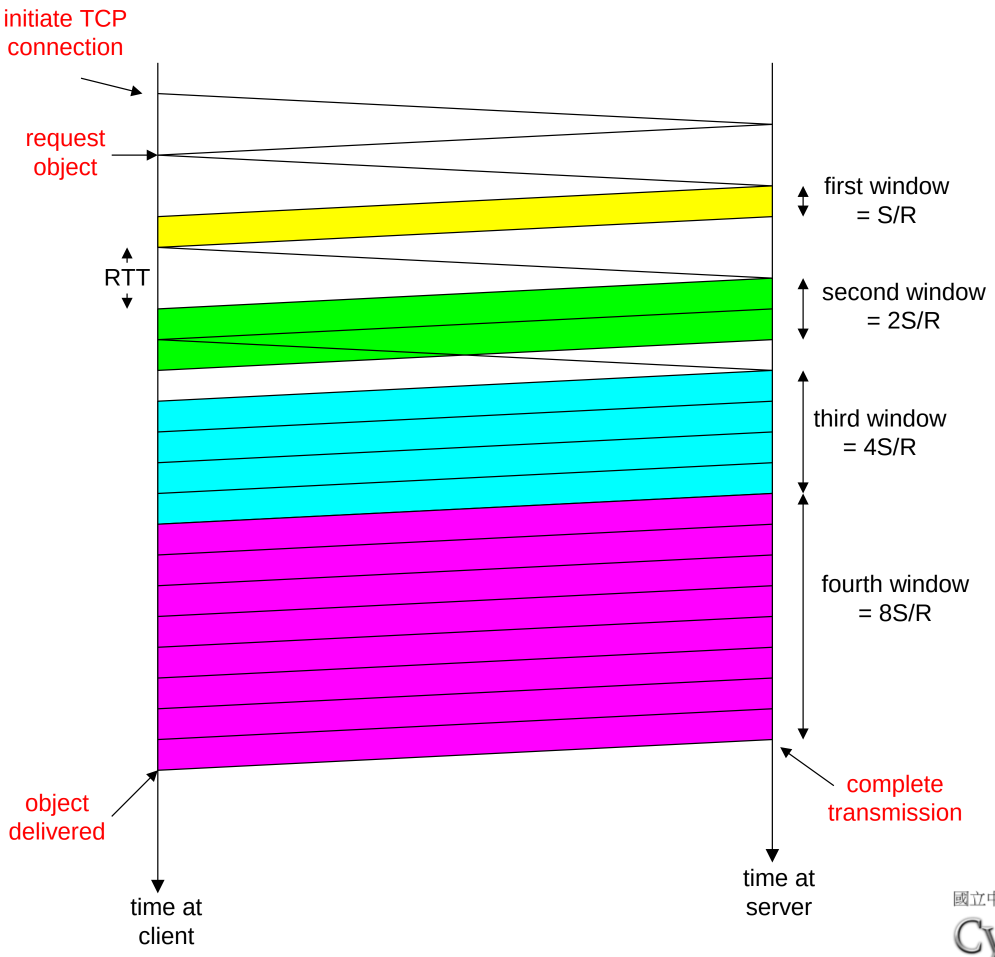
- 2 RTT for connection estab and request
- O/R to transmit object
- time server idles due to slow start

Server idles:
P = min{K-1,Q} times

Example:

- O/S = 15 segments
- K = 4 windows
- Q = 2
- P = min{K-1,Q} = 2

Server idles P=2 times



# TCP Delay Modeling (3)

$$\frac{S}{R} + RTT = \text{time from when server starts to send segment}$$

$$\text{until server receives acknowledgement}$$

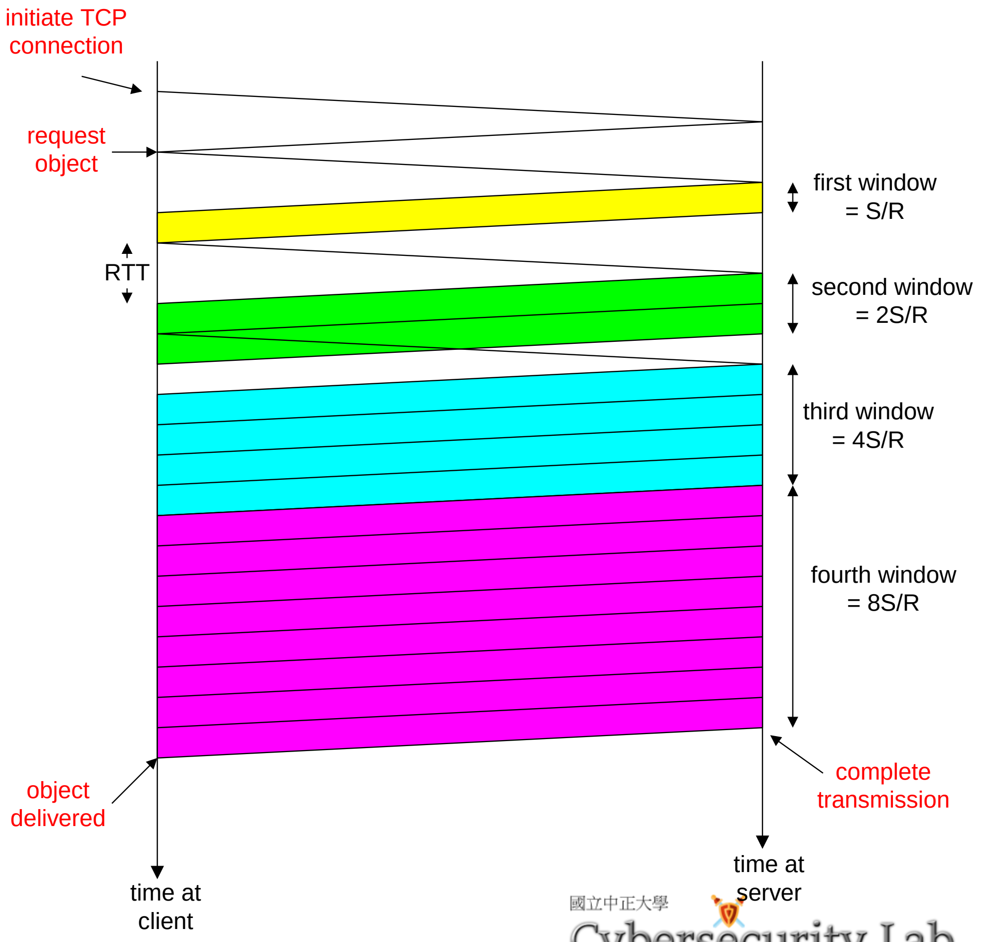
$$2^{k-1}\frac{S}{R} = \text{time to transmit the kth window}$$

$$\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]^{+} = \text{idle time after the } k\text{th window}$$

$$\text{delay} = \frac{O}{R} + 2RTT + \sum_{p=1}^{P} idleTime_p$$

$$= \frac{O}{R} + 2RTT + \sum_{k=1}^{P}\left[\frac{S}{R} + RTT - 2^{k-1}\frac{S}{R}\right]$$

$$= \frac{O}{R} + 2RTT + P\left[RTT + \frac{S}{R}\right] - (2^{P} - 1)\frac{S}{R}$$



# TCP Delay Modeling (4)

Recall K = number of windows that cover object
How do we calculate K ?

$$
\begin{aligned}
K &= \min\{k : 2^0 S + 2^1 S + \cdots + 2^{k-1} S \geq O\} \\
&= \min\{k : 2^0 + 2^1 + \cdots + 2^{k-1} \geq O/S\} \\
&= \min\{k : 2^k - 1 \geq \frac{O}{S}\} \\
&= \min\{k : k \geq \log_2(\frac{O}{S} + 1)\} \\
&= \left\lceil \log_2(\frac{O}{S} + 1) \right\rceil
\end{aligned}
$$

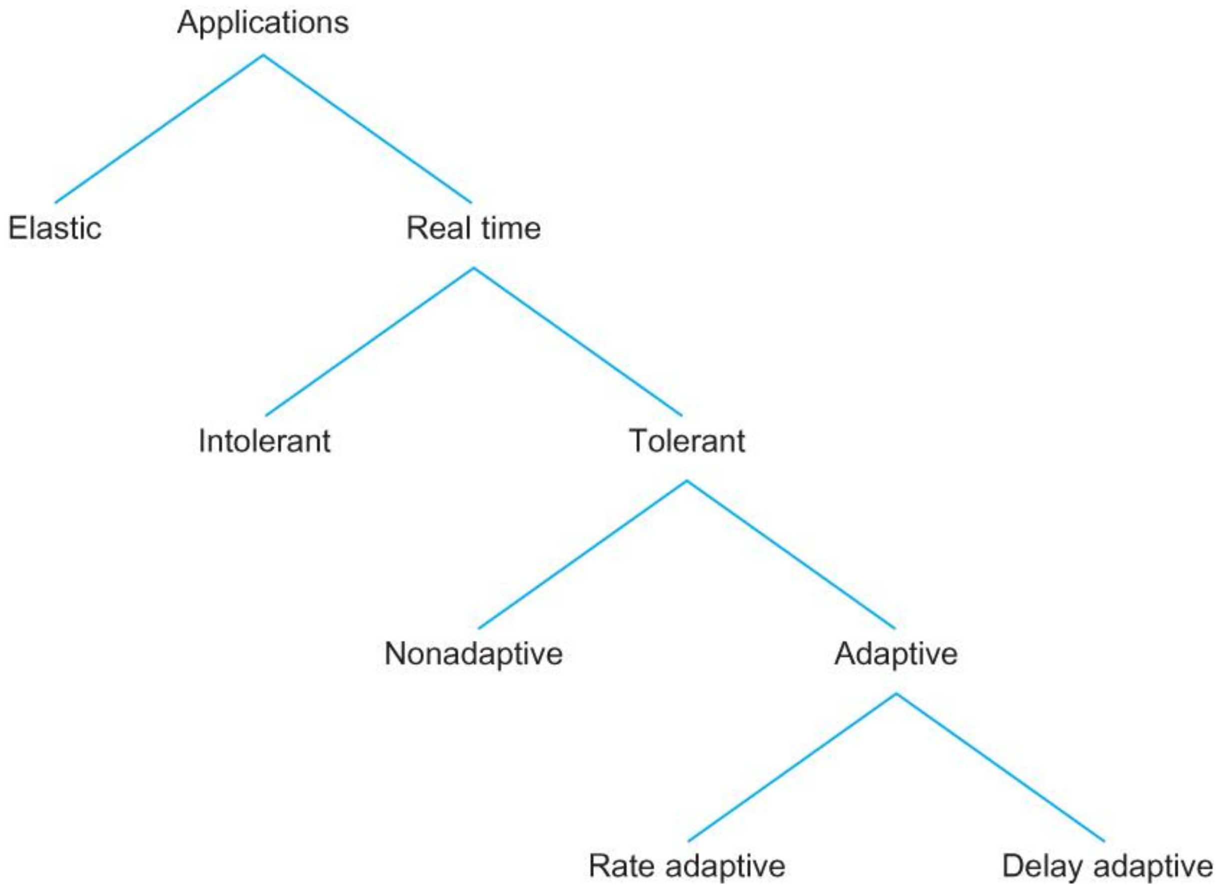
Calculation of Q, number of idles for infinite-size object, is similar (see HW).

# Quality of Service

- Definition: the use of mechanisms or technologies that work on a network to control traffic and ensure the performance of critical applications with limited network capacity
  - ✓ Voice and video applications: no delay, lag
  - ✓ File transfer applications: timeliness constraints
  - ✓ Phone: call can be heard by the partner
- Real-Time Applications
  - Data is generated by collecting samples from a microphone and digitizing them using an A →D converter
  - The digital samples are placed in packets which are transmitted across the network and received at the other end
  - At the receiving host the data must be played back at some appropriate rate

# QoS in multiple applications

- There are many apps require QoS
- Applications Can adjust their playback point *delay-adaptive applications.*
- Another class of adaptive applications are *rate adaptive*



# Best Effort vs. QoS

- Best Effort:
    - You get a link to the Internet with at most B bits/sec.
    - If you don't like it, switch to another provider.
- Quality of Service (QoS)
  - We provide you some kind of guarantees for:
    - Bandwidth
    - Latency
    - Jitter
  - I.e., network is engineered to provide some Quality beyond "Not to exceed B bits/s"

# Two Styles of QoS

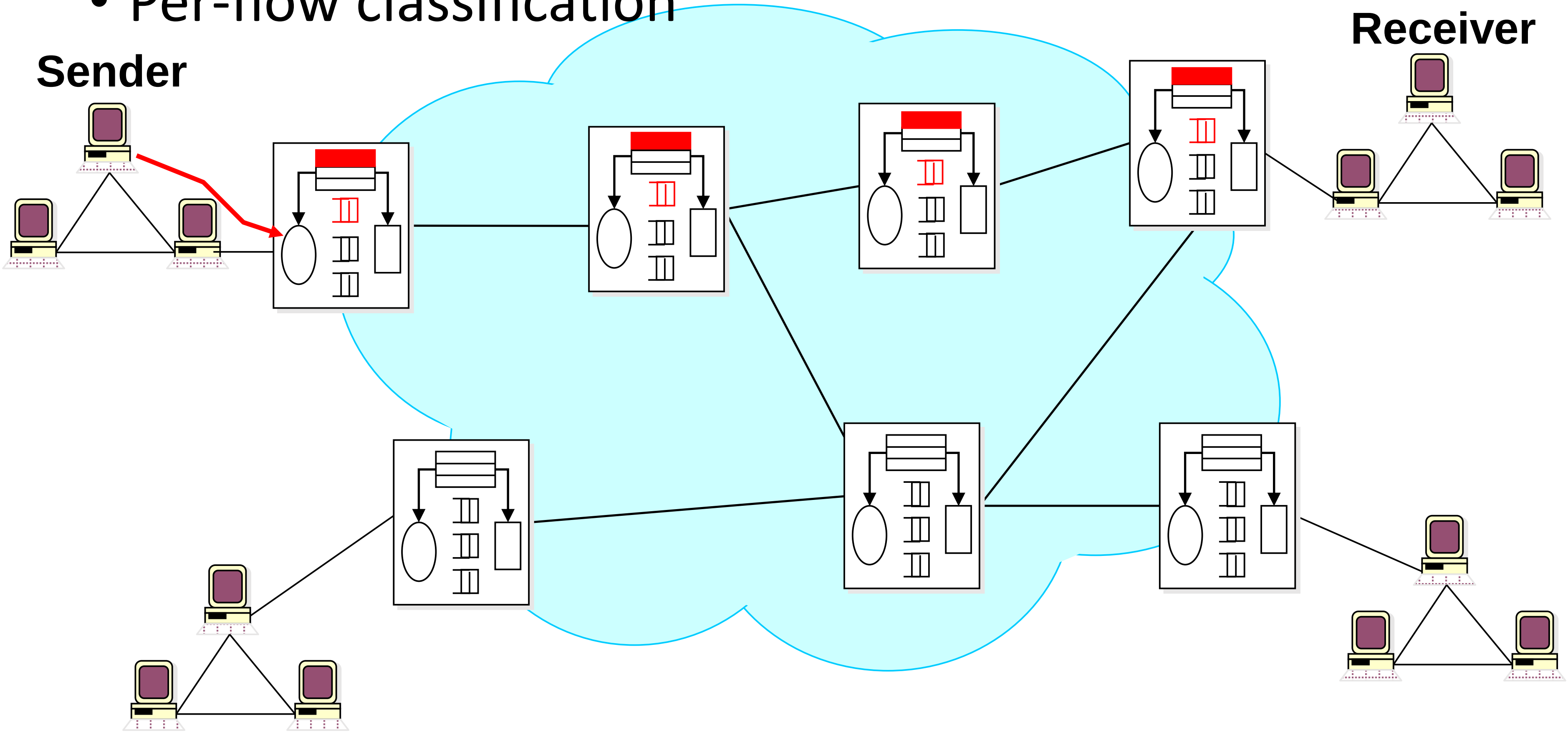
- Worst-case
  - Provide bandwidth/delay/jitter guarantee to every packet
  - E.g., “hard real time”
- Average-case
  - Provide bandwidth/delay/jitter guarantee over many packets
  - Statistical in nature
  - E.g. “Soft real time”

# Quality of service issues

- Flow specification
  - Flow spec: traffic characteristics, QoS requirements (delay, jitter,bandwidth)
- Routing
  - Routing traffic to best meet demand
- Resource reservation
  - End-host signaling to network QoS resource requirements
- Admission control
  - Limiting number of reservations
- Packet scheduling
  - Packet by packet scheduling (fairness, delay)
- Resource Reservation Protocol (RSVP) addresses reservation

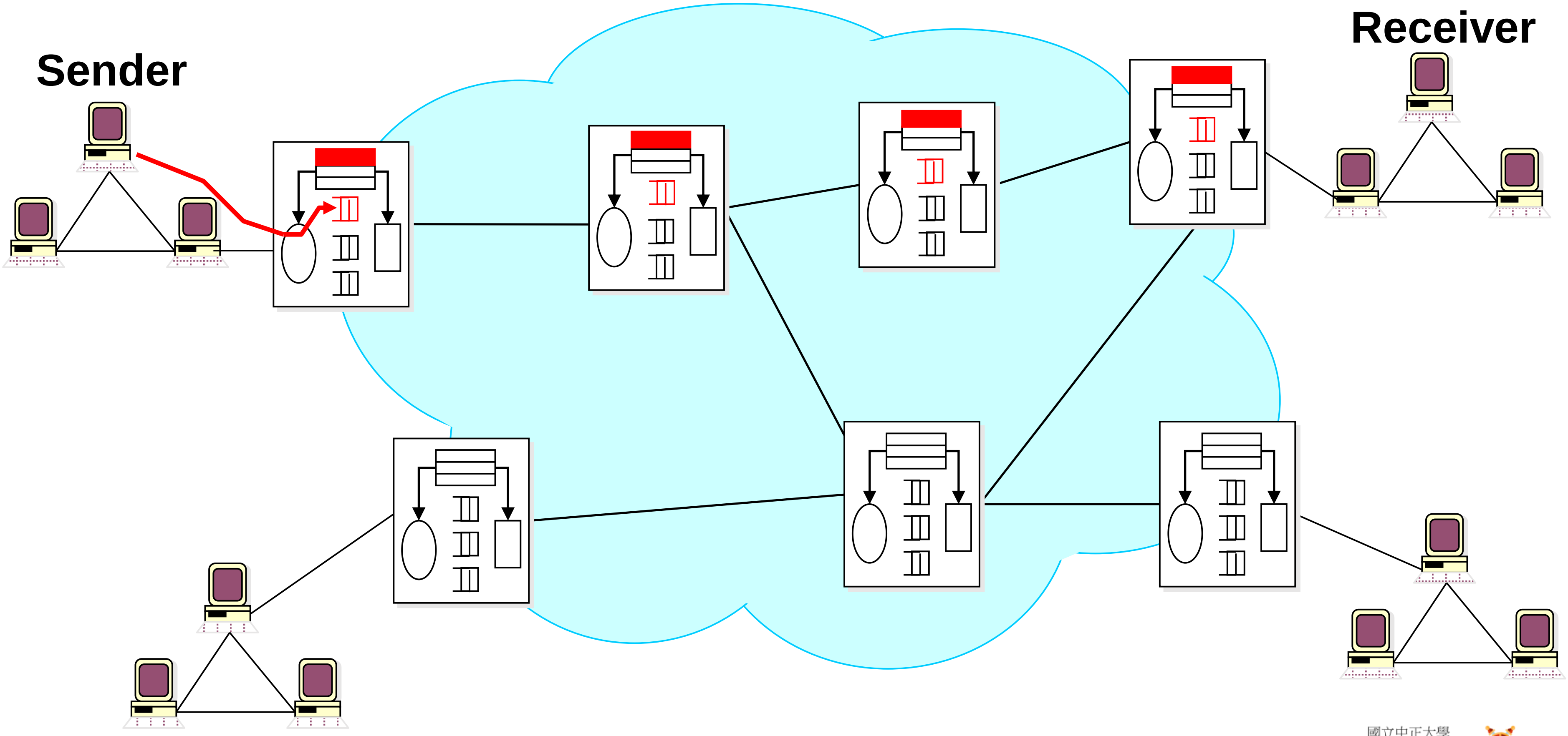
# Integrated Services Example: Data Path

- Per-flow classification



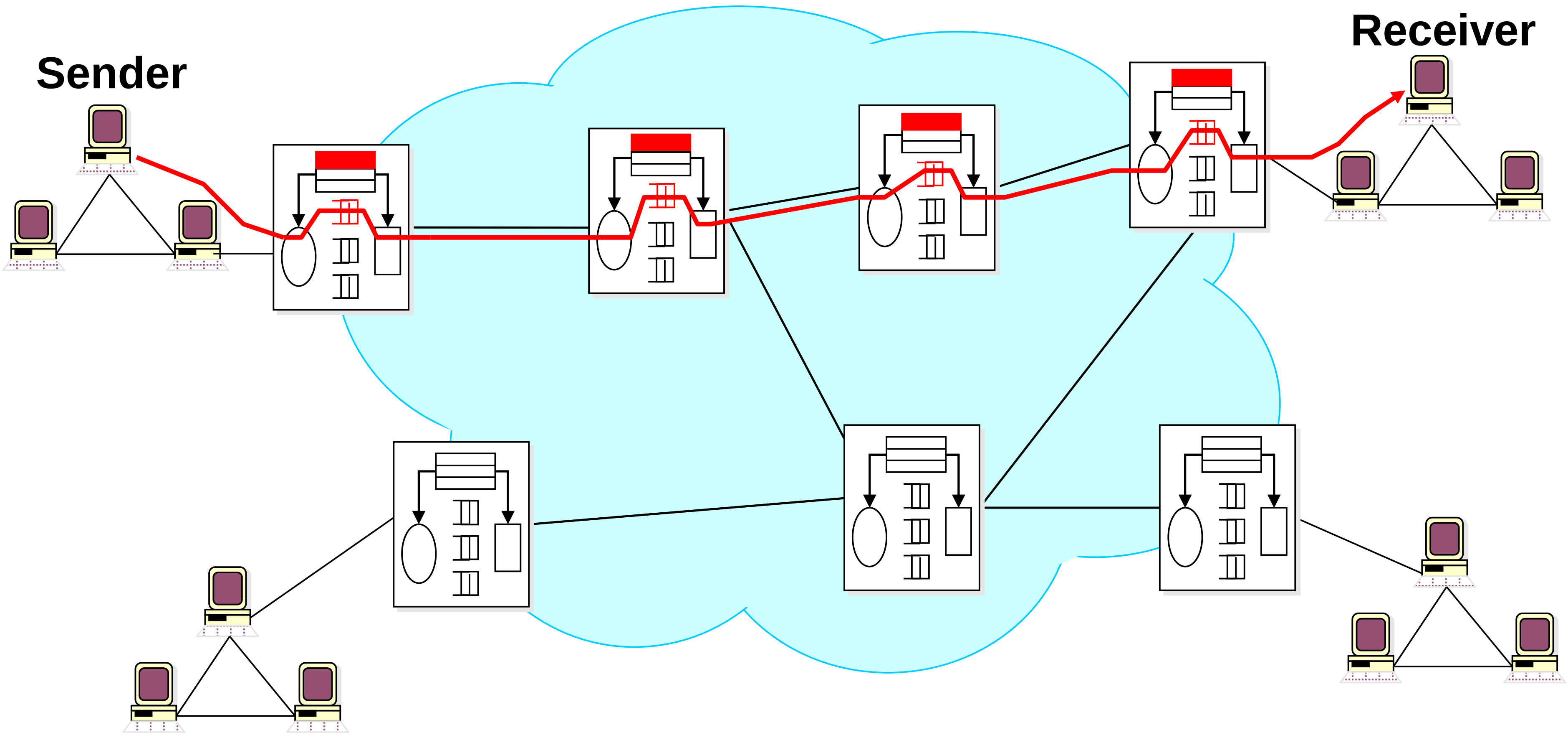
# Integrated Services Example: Data Path

- Per-flow buffer management

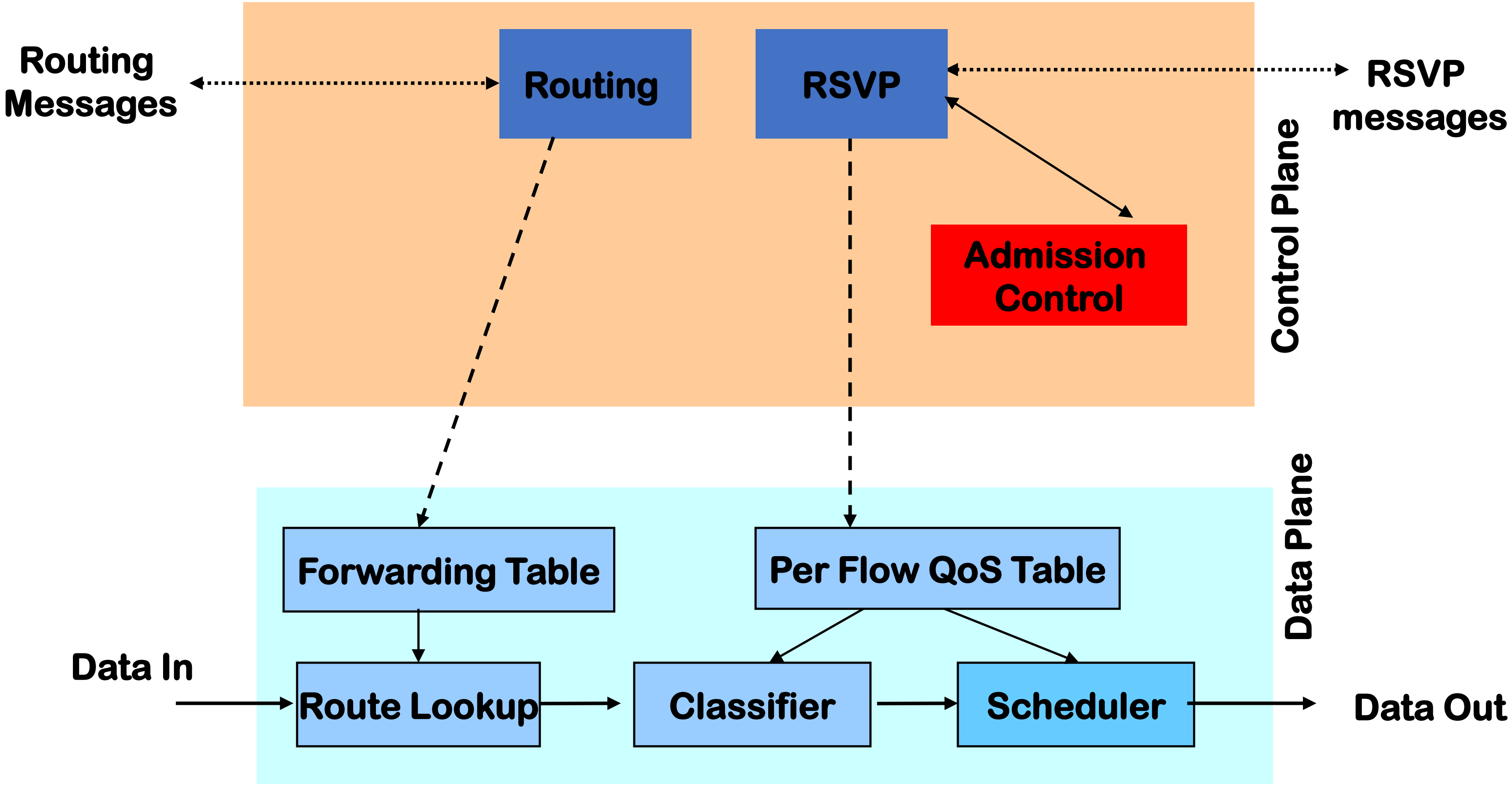


# Integrated Services Example

- Per-flow scheduling

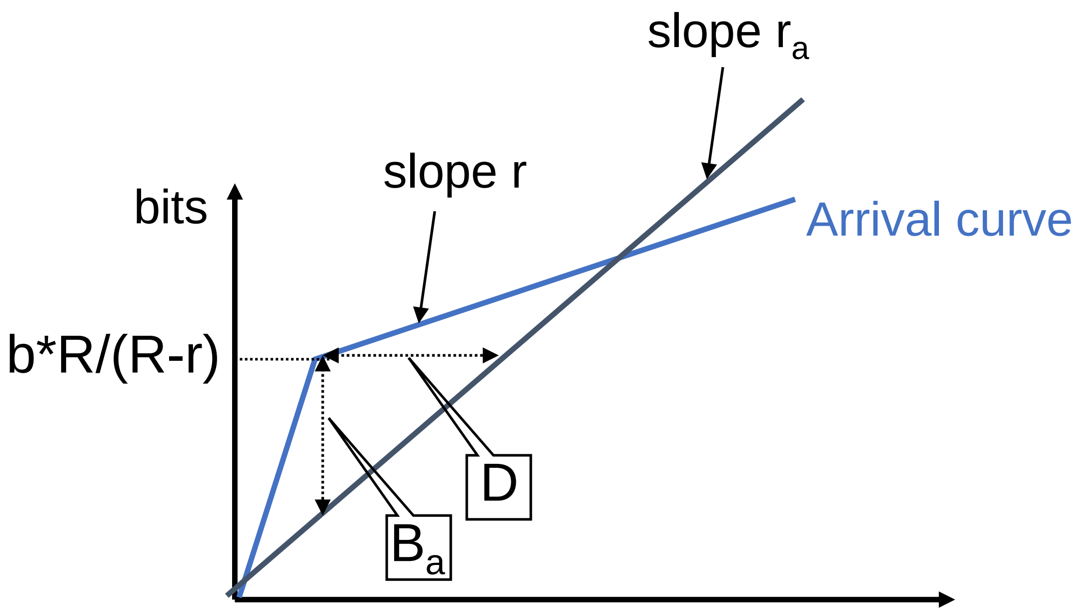


# How Things Fit Together



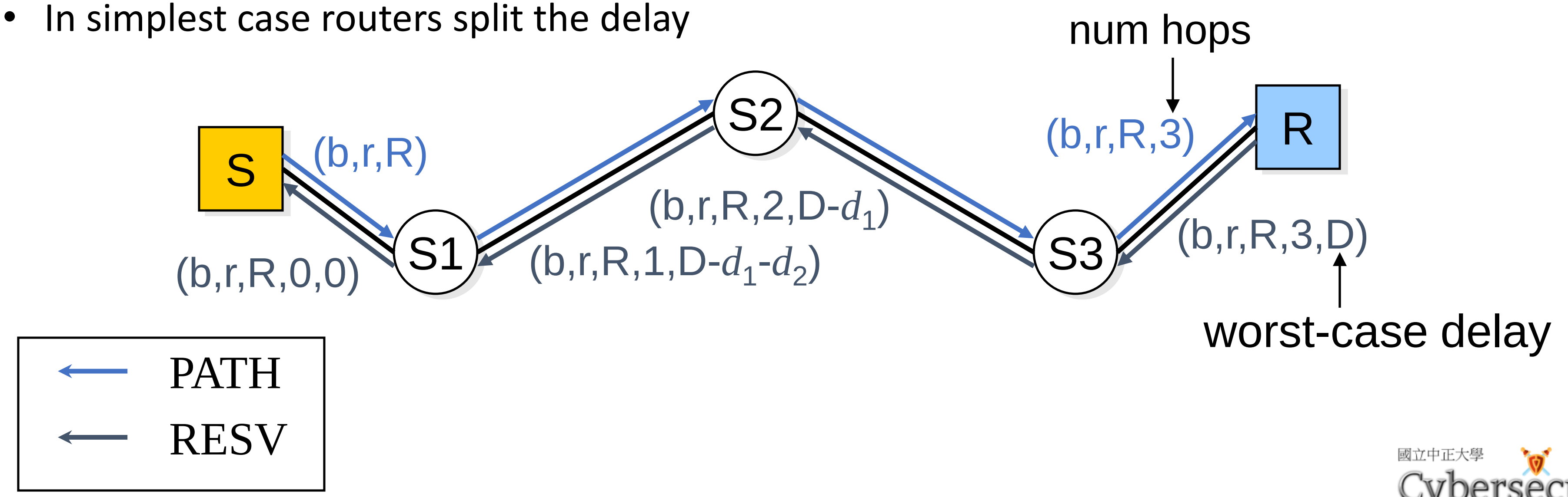
# QoS Guarantees: Per-hop Reservation

- End-host: specify
  - the arrival rate characterized by token-bucket with parameters (b,r,R)
  - the maximum maximum admissible delay D
- Router: allocate bandwidth $r_a$ and buffer space $B_a$ such that
  - no packet is dropped
  - no packet experiences a delay larger than D



# End-to-End Reservation

- When R gets PATH message it knows
  - Traffic characteristics (tspec): (r,b,R)
  - Number of hops
- R sends back this information + worst-case delay in RESV
- Each router along path provide a per-hop delay guarantee and forward RESV with updated info
  - In simplest case routers split the delay

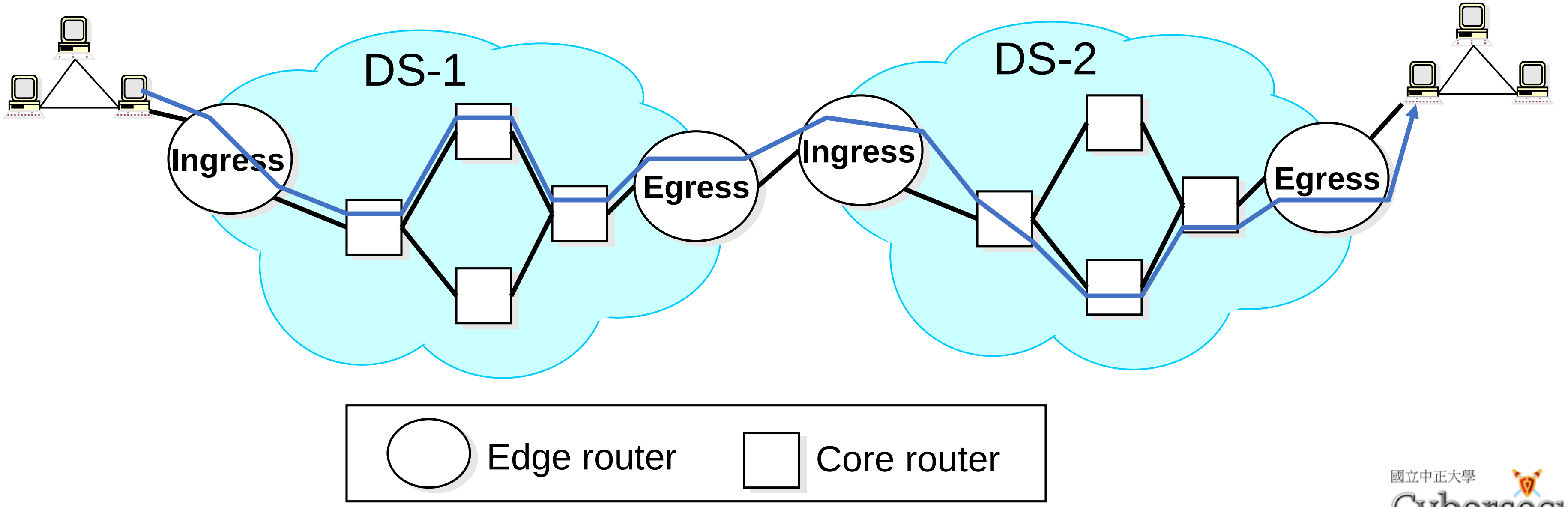


# Differentiated Services (Diffserv)

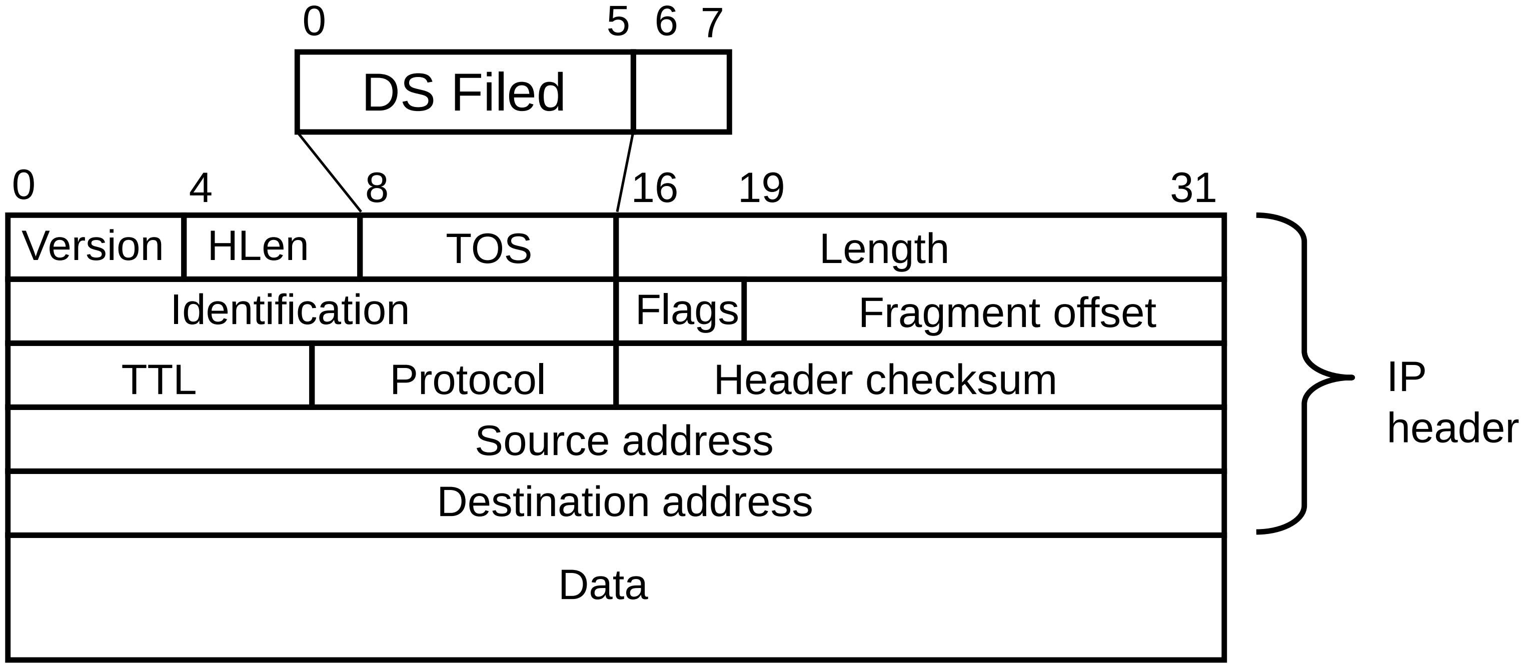
- Definition: Classifying and managing network traffic and providing quality of service on modern IP network
- Differentiate between edge and core routers
- Edge routers
  - Perform per aggregate shaping or policing
  - Mark packets with a small number of bits; each bit encoding represents a class (subclass)
- Core routers
  - Process packets based on packet marking

# Diffserv Architecture

- Ingress routers
  - Police/shape traffic
  - Set Differentiated Service Code Point (DSCP) in Diffserv (DS)  field
- Core routers
  - Implement Per Hop Behavior (PHB) for each DSCP
  - Process packets based on DSCP

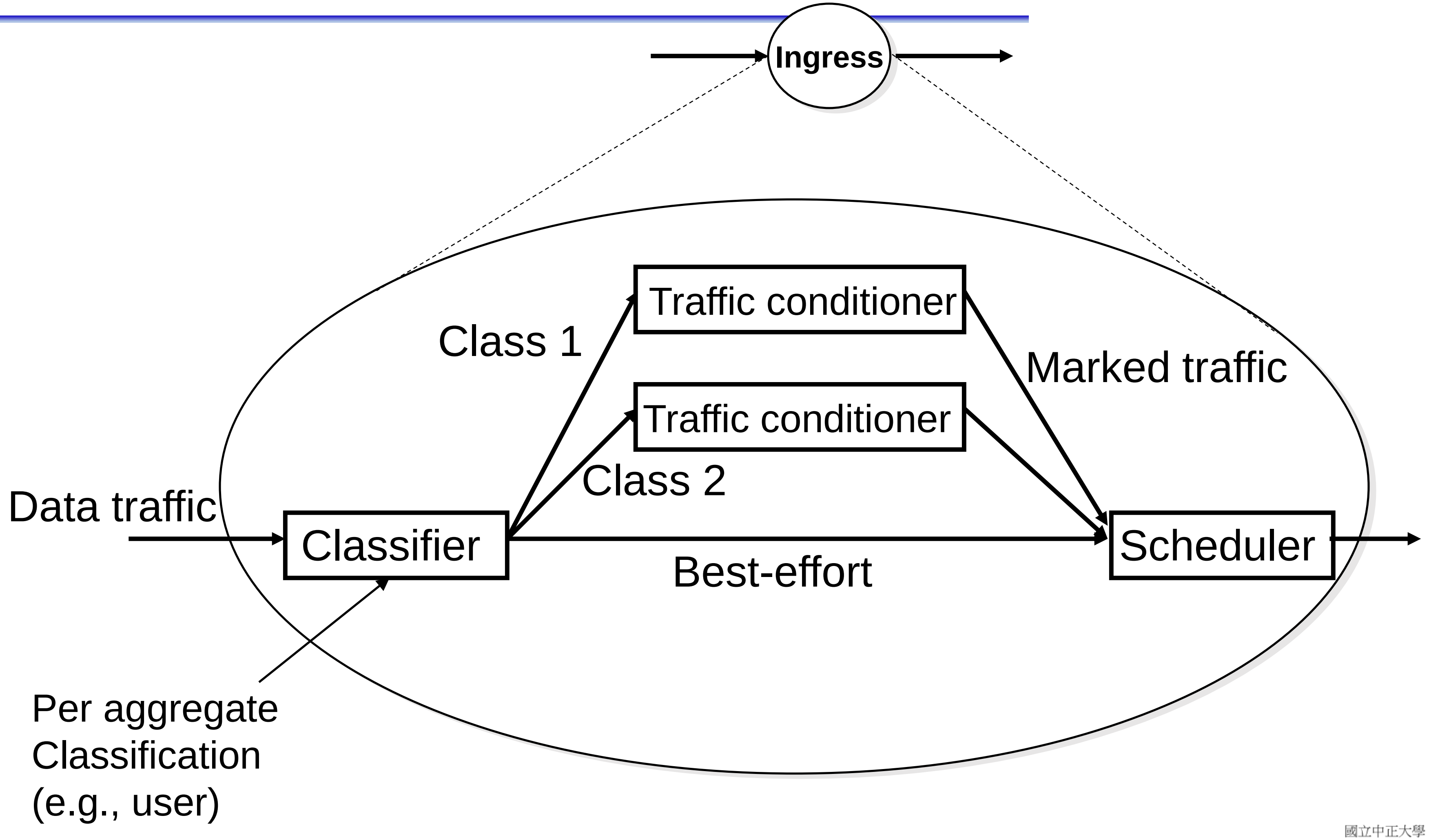


# Differentiated Service (DS) Field




- DS filed reuse the first 6 bits from the former Type of Service (TOS) byte
- The other two bits are proposed to be used by ECN

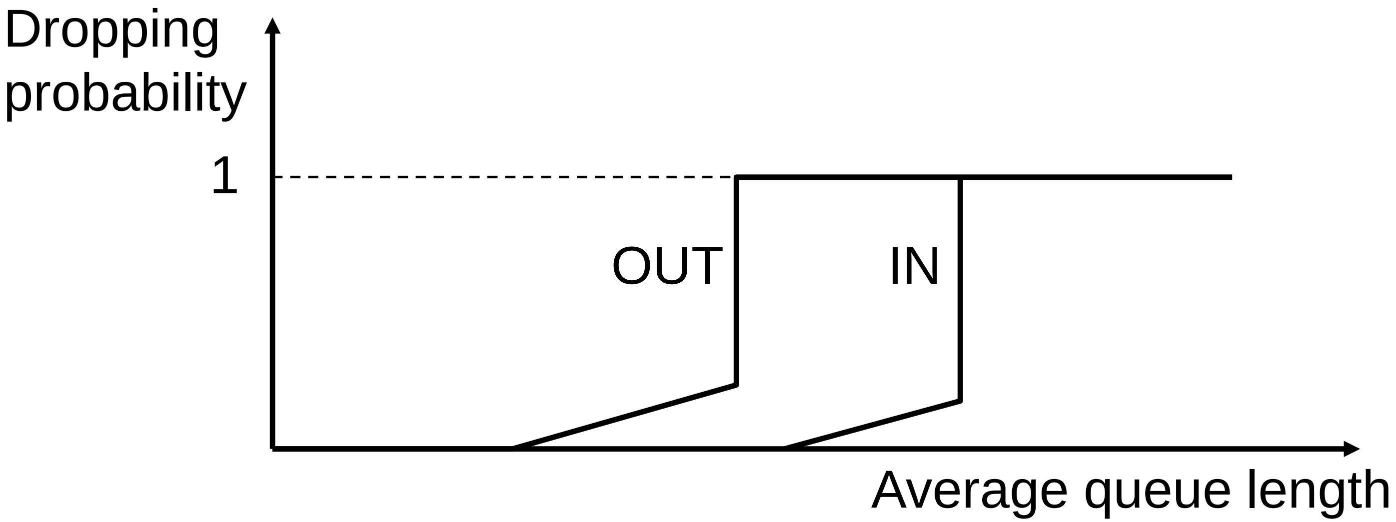
# Edge Router



Ingress
Class 1
Traffic conditioner
Traffic conditioner
Marked traffic
Class 2
Data traffic
Classifier
Scheduler
Best-effort
Per aggregate
Classification
(e.g., user)

# Scheduler

- Employed by both edge and core routers
- For premium service – use strict priority, or weighted fair queuing (WFQ)
- For assured service – use RIO (RED with In and Out)
  - Always drop OUT packets first
    - For OUT measure entire queue
    - For IN measure only in-profile queue



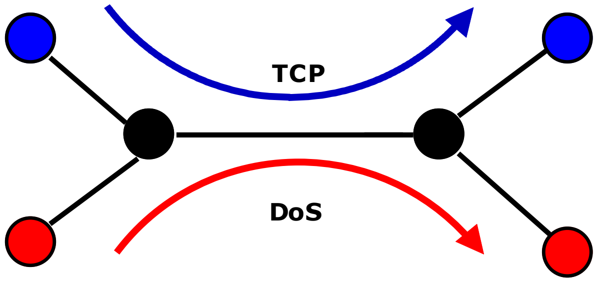

國立中正大學 Cybersecurity Lab

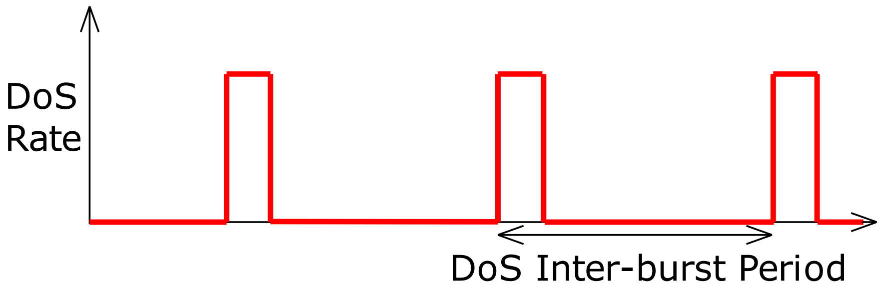
# Comparison: Best-Effort, Diffserv, Intserv

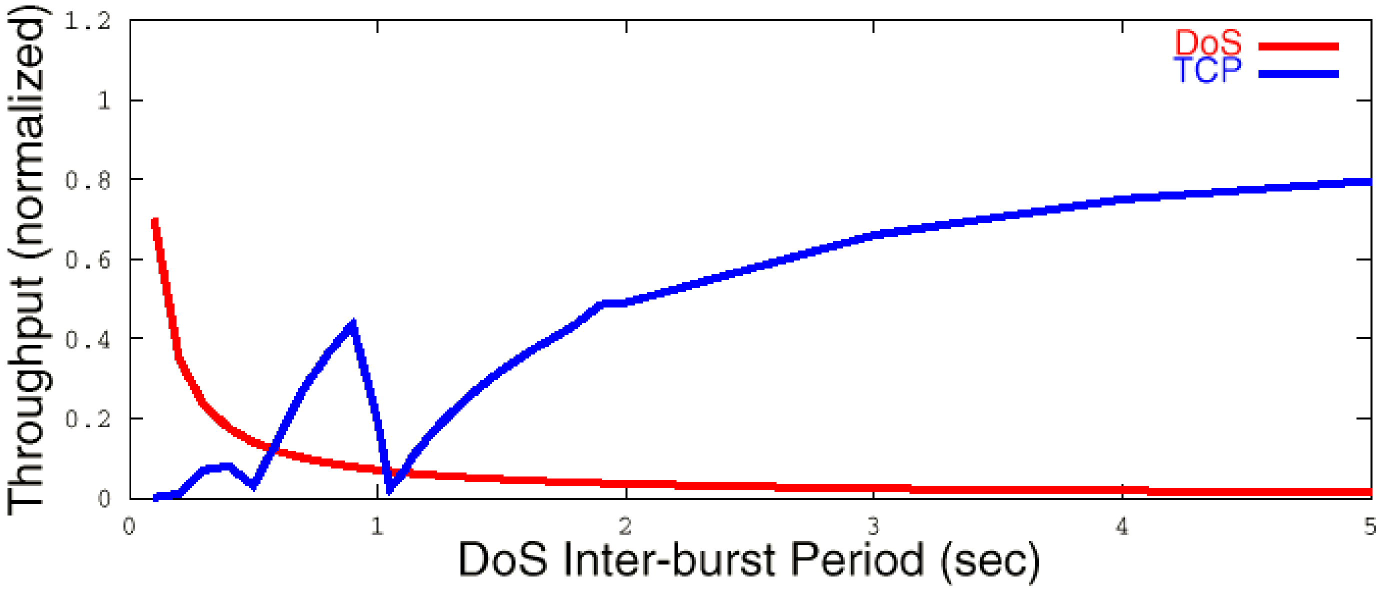
| | Best-Effort | Diffserv | Intserv |
|---|---|---|---|
| Service | Connectivity<br>No isolation<br>No guarantees | Per aggregate isolation<br>Per aggregate guarantee | Per flow isolation<br>Per flow guarantee |
| Service scope | End-to-end | Domain | End-to-end |
| Complexity | No setup | Long term setup | Per flow setup |
| Scalability | Highly scalable (nodes maintain only routing state) | Scalable<br>(edge routers maintains per aggregate state; core routers per class state) | Not scalable (each router maintains per flow state) |

# Low-Rate Attacks

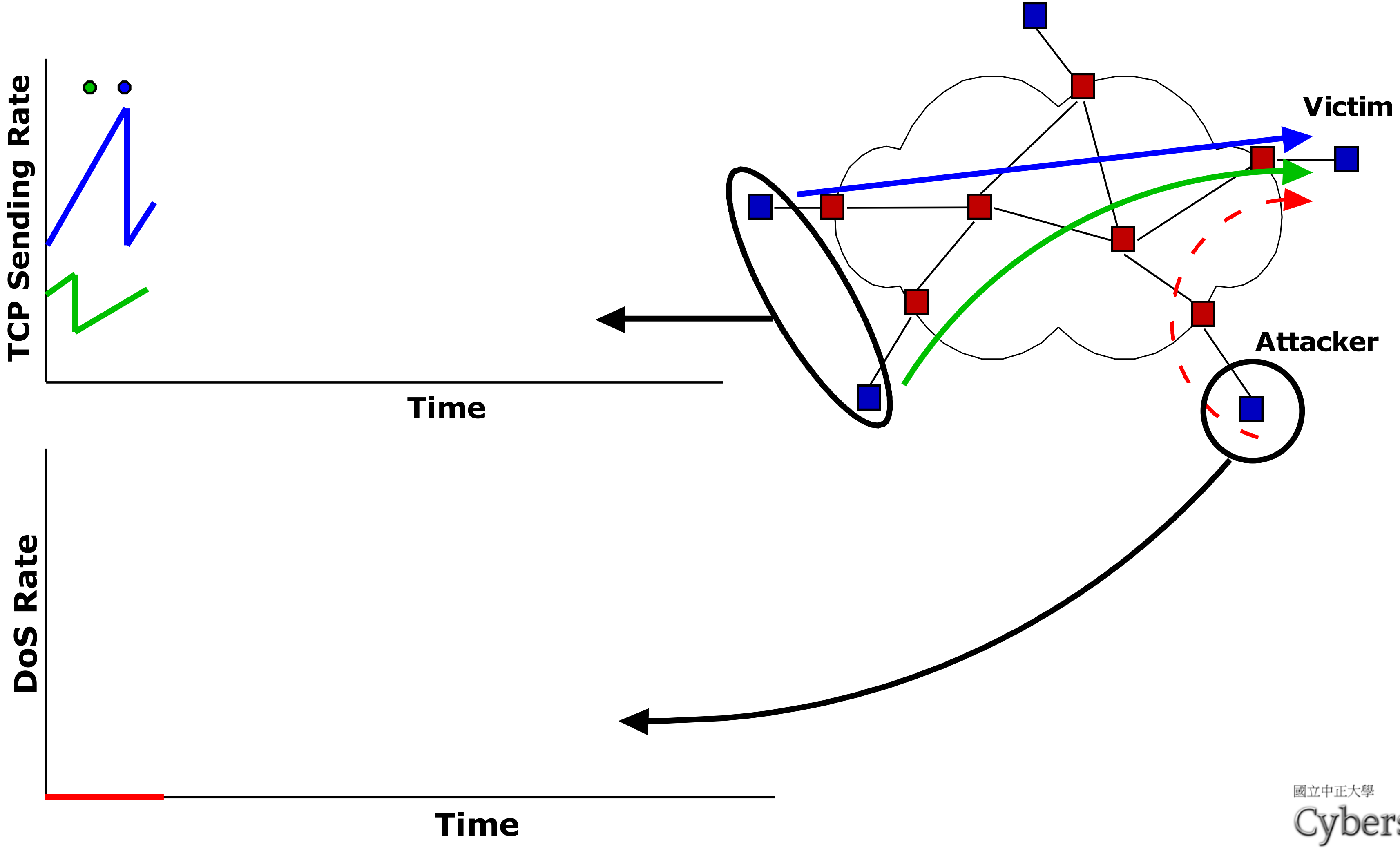
- TCP is vulnerable to low-rate DoS attacks



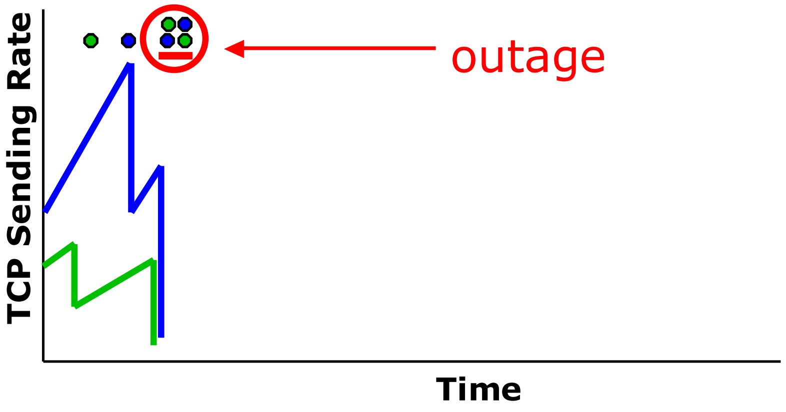



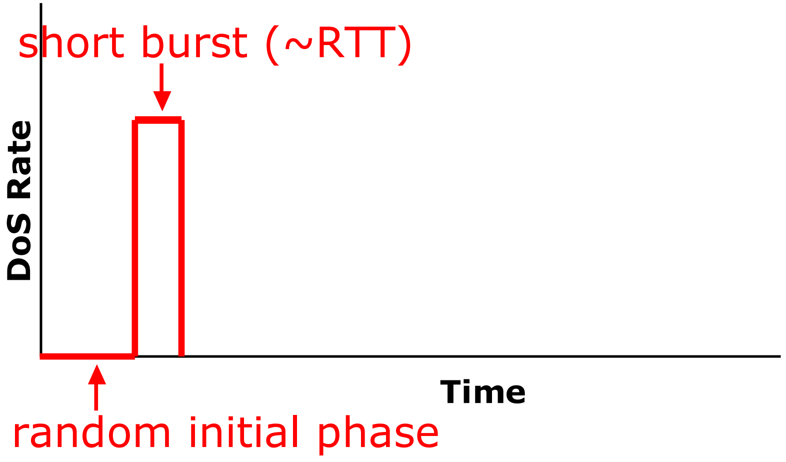



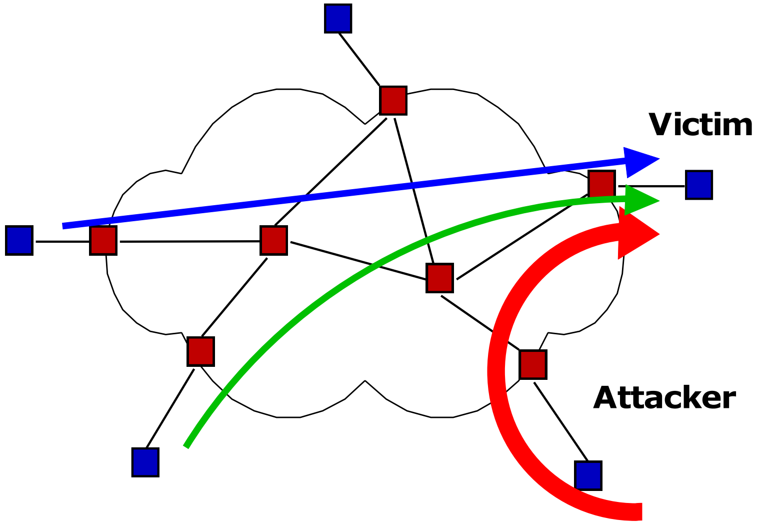
# The Low-Rate Attack



# The Low-Rate Attack

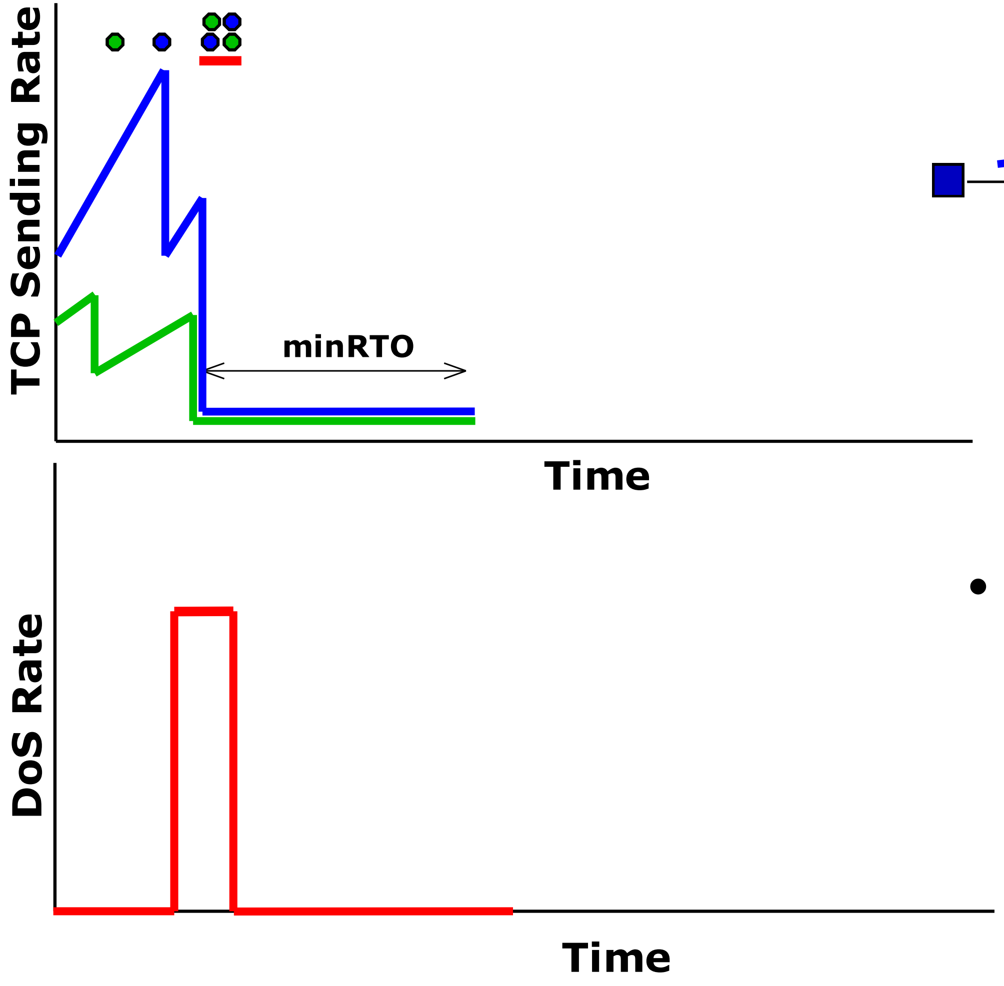


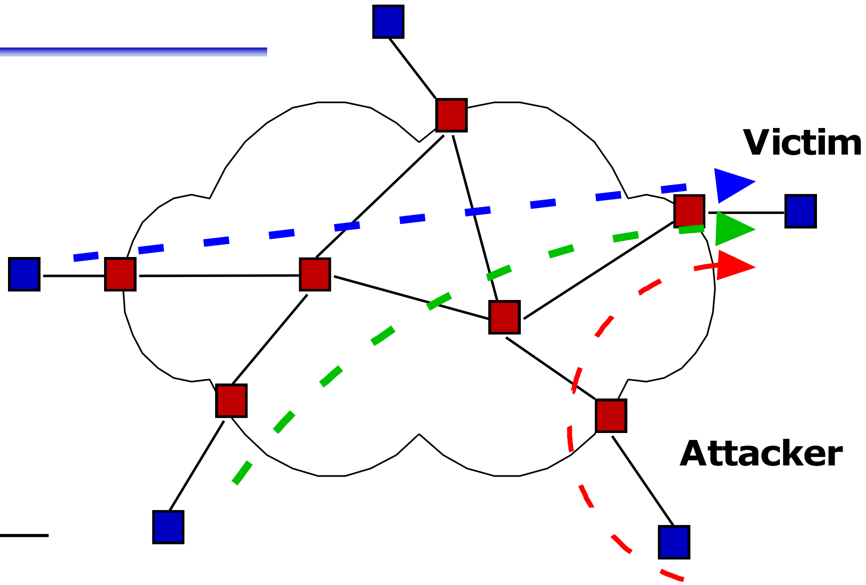




- A short burst (~RTT) sufficient to create outage
- Outage – event of correlated packet losses that forces TCP to enter RTO mechanism

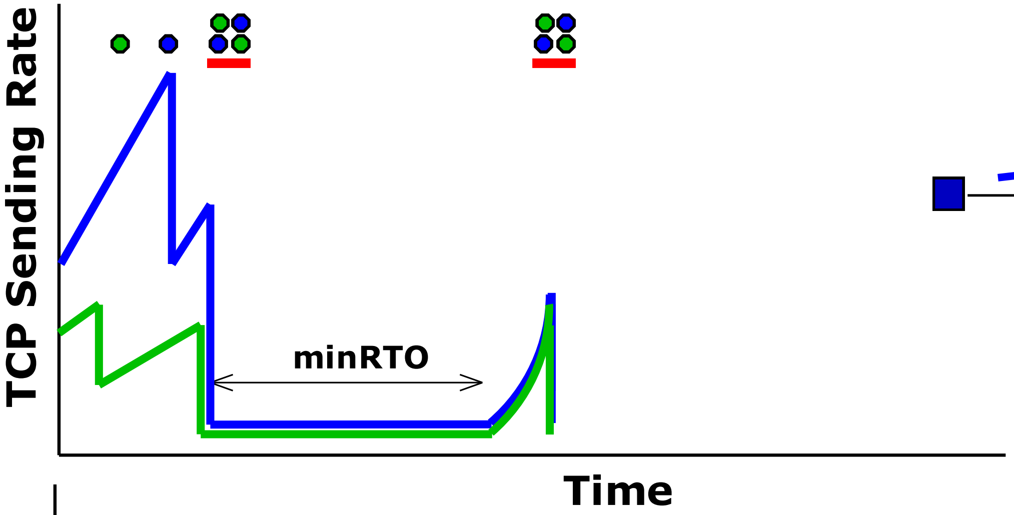
# The Low-Rate Attack

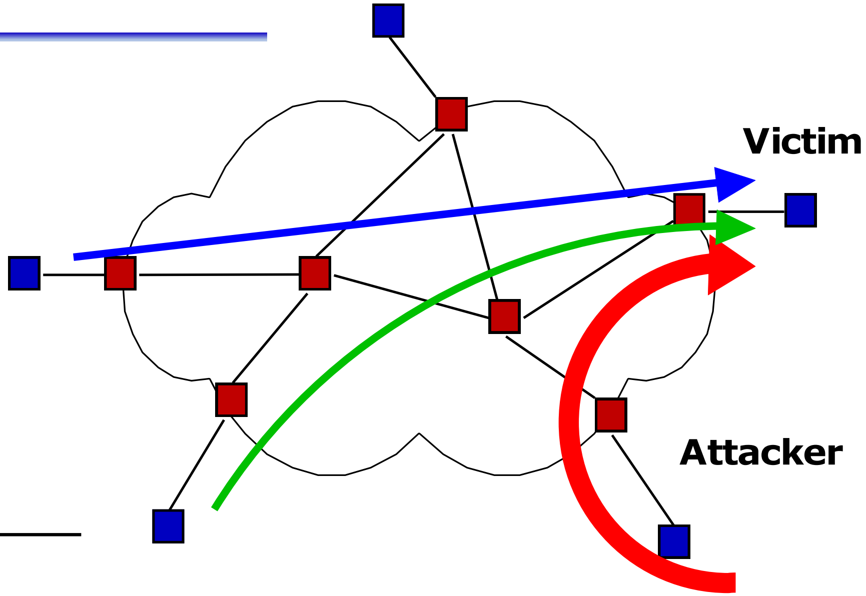


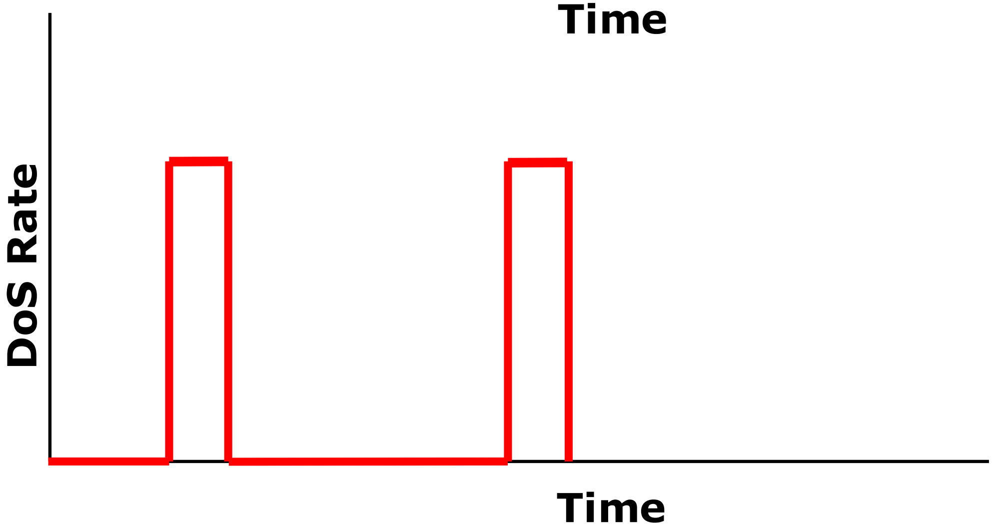




- The outage synchronizes all TCP flows
  - All flows react simultaneously and identically
    - backoff for minRTO

# The Low-Rate Attack

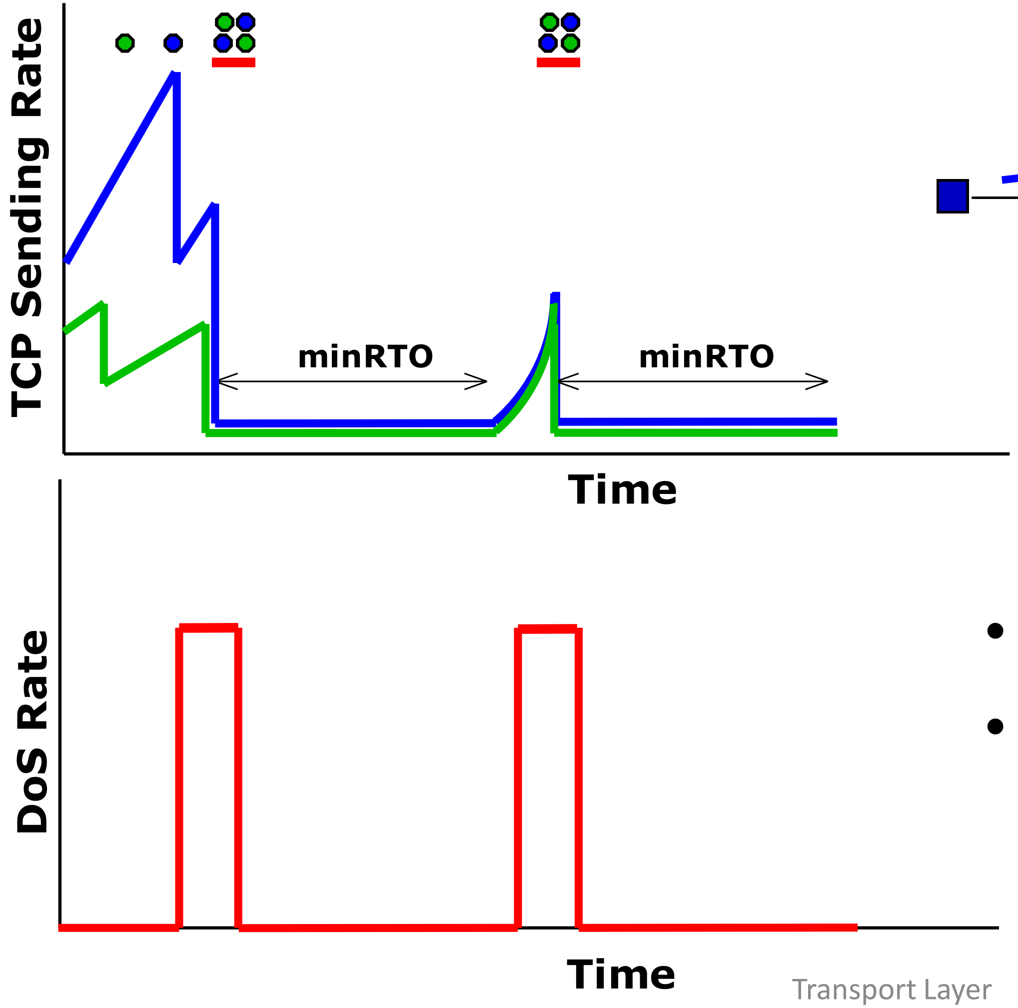


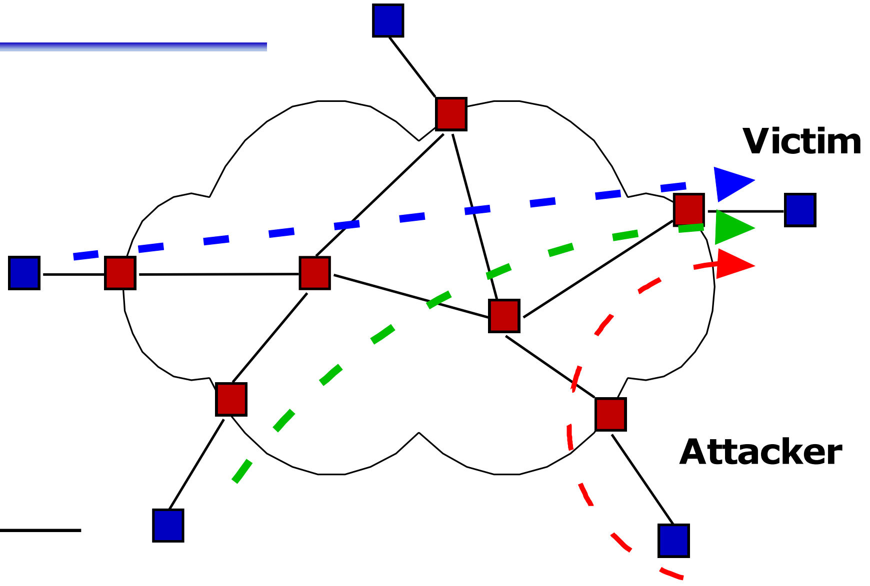




- Once the TCP flows try to recover – hit them again
- Exploit protocol determinism

# The Low-Rate Attack

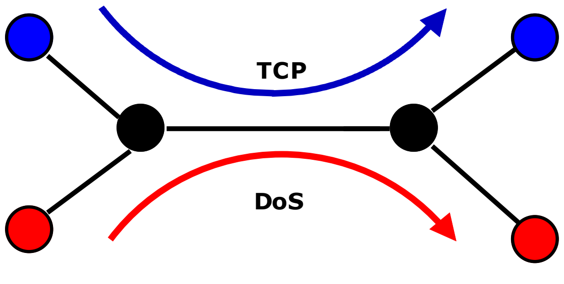


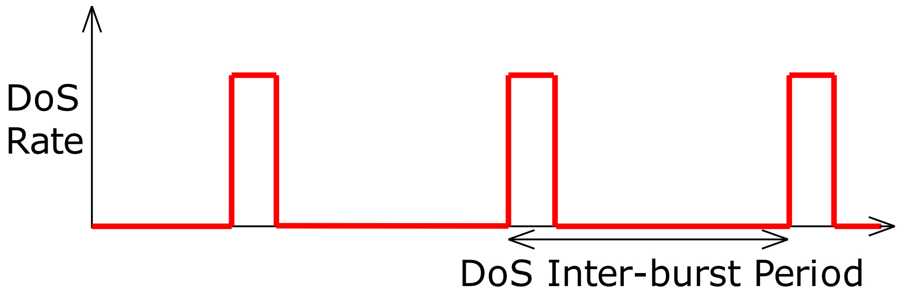



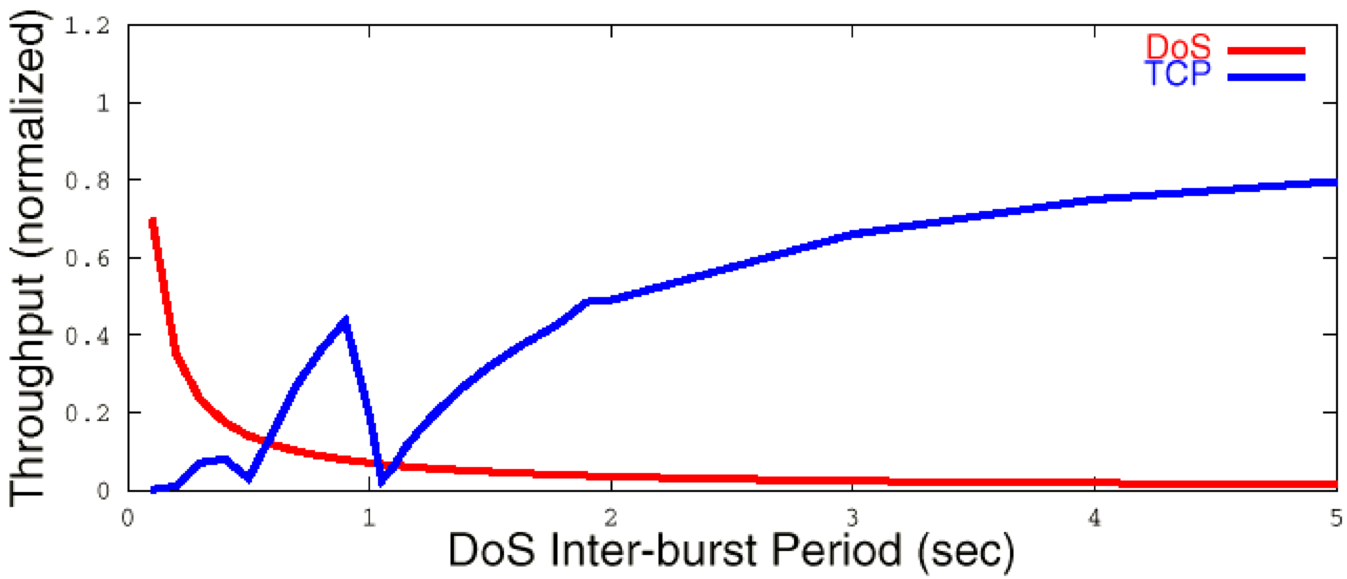

- And keep repeating…
- RTT-time-scale outages inter-spaced on minRTO periods can deny service to TCP traffic

# Low-Rate Attacks

- TCP is vulnerable to low-rate DoS attacks



# Summary

- Congestion control: ABR, Fast Retransmission, AIMD, …
- Differential services
- Low-rate attacks

# Homework 2

- HW2 is out
- You can use ns3 or OMNet++. Either  is ok
- These platforms can be used for your future research

| # | Homework 1 goals | Homework 2 goals |
|---|---|---|
| 1 | Understand network terms/concepts/ basic network protocols | Understand networking techniques & advanced network issues |
| 2 | Configure networking devices (router/switch) | Install a platform for research |
| 3 | Understand how the Internet is operated and design the local networks | Understand how the research is performed in Computer Networks |