RW354
Principles of Computer Networking

A.E. Krzesinski and B.A. Bagula
Department of Computer Science
University of Stellenbosch

Last updated: 15 September 2004

The material presented in these slides is used with permission from

- Larry L. Peterson and Bruce S. Davie. Computer Networks: A Systems Approach (Second Edition). Morgan Kaufmann Publishers. ISBN 1-55860-577-0.
- William Stallings. Data and Computer Communications (Sixth Edition). Prentice-Hall Inc. ISBN 0-13-571274-2.
- Andrew S. Tannenbaum. Computer Networks (Fourth Edition). Prentice Hall Inc. ISBN 0-13-349945-6.

Congestion Control: Introduction

Network resources – link bandwidth & buffers in routers – should be allocated fairly among the competing users.

Packets are queued at a router waiting to be transmitted over a link. Congestion occurs when too many packets are waiting for the link: packets are dropped.

Remedies

- *allocate network resources dynamically so that congestion never occurs – difficult*
- *let congestion occur & recover from congestion – easier, but potentially disruptive*
- *compromise – allocate resources as best you can & recover from congestion if it occurs.*

Congestion Control: Issues

Resource allocation is implemented in the routers & in the transport protocol running on the end systems.

Resource allocation

- *routers try to meet the competing demands of network applications for network resources.*

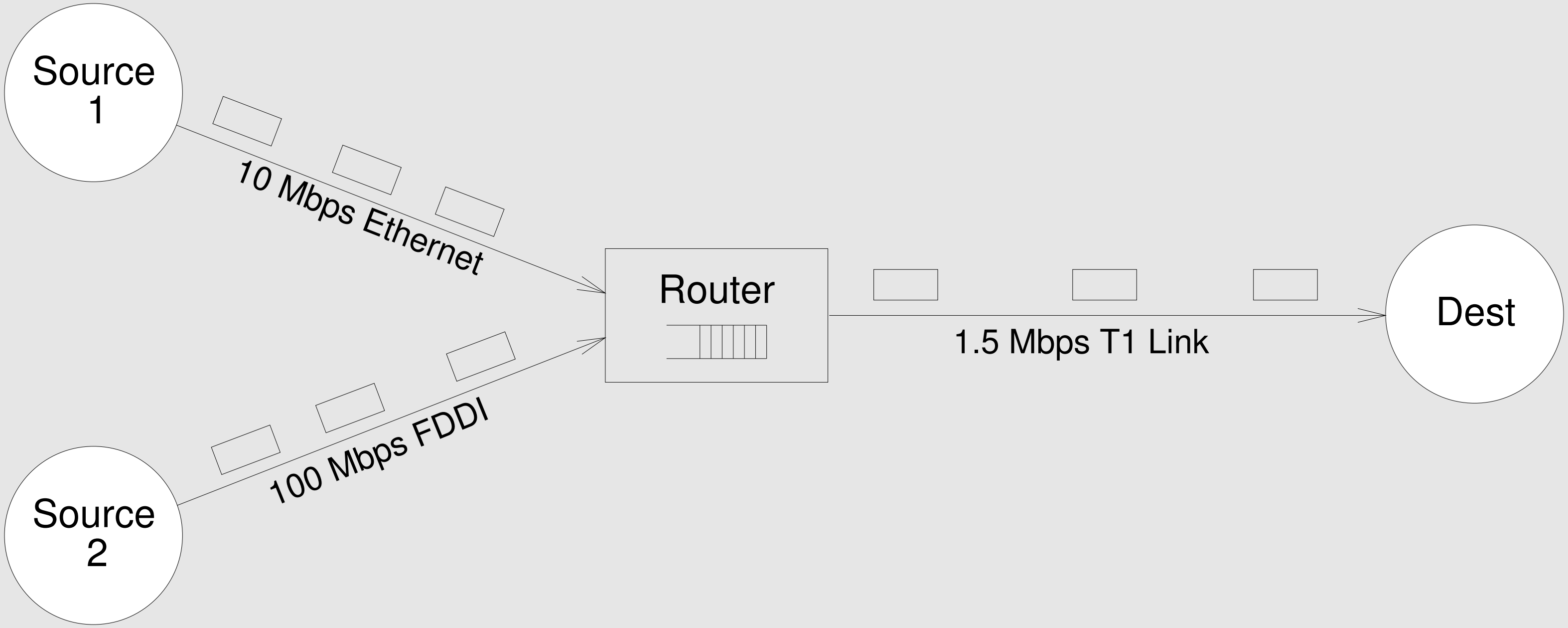
Congestion control

- *efforts made by the network to prevent or respond to overload*
- *fairness*
  - *congestion control mechanisms have resource allocation built into them.*

Congestion Control: Overview

Congestion control & resource allocation are closely related

- *pre-allocate resources to avoid congestion*
- *send data: control congestion if it occurs.*



Congestion Control: Overview

- *Two points of implementation*
  - *hosts at the edges of the network: transport protocol*
  - *routers inside the network: the queuing discipline of the packet scheduler.*
- *The underlying service model*
  - *best-effort: assume for now*
  - *multiple qualities of service: later.*

Congestion Control: Flows & Soft State

Although packets are switched independently, a stream of packets between an O-D pair often flows through a particular set of routers.

A flow is a sequence of packets sent between an O-D pair that follow the same route.

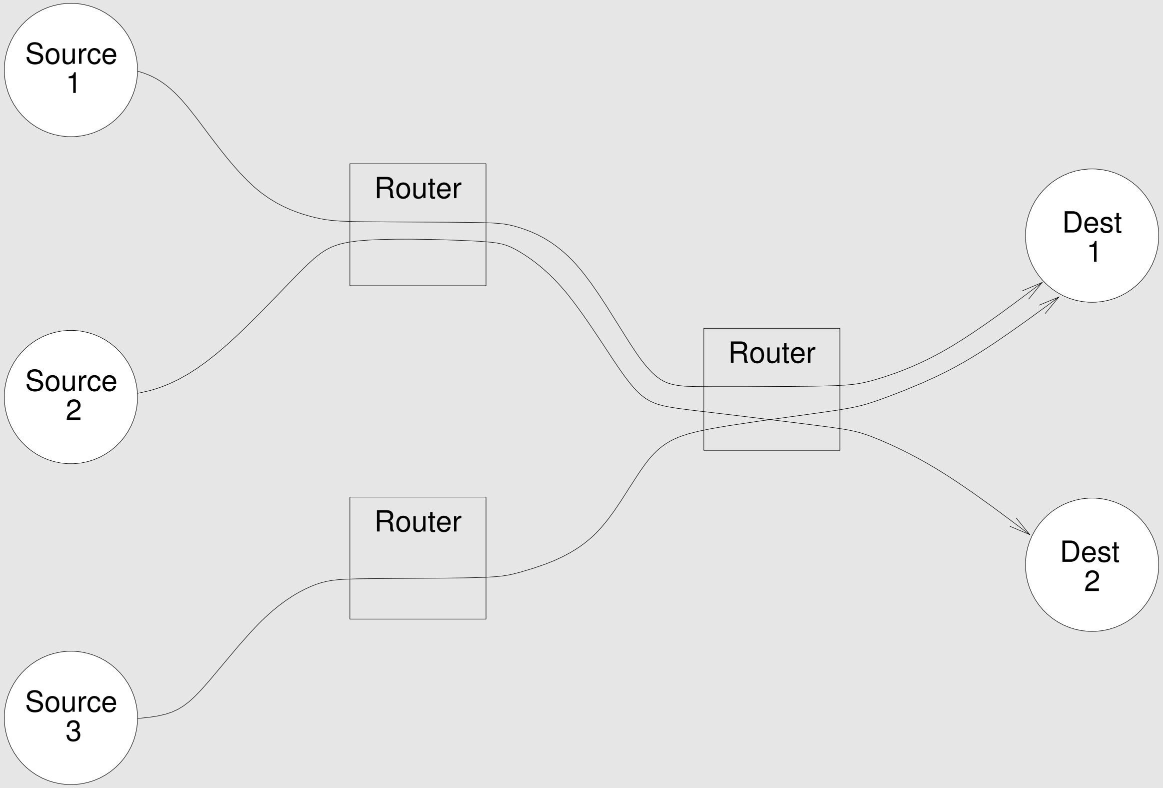
Each router maintains soft state information to make resource allocation decisions for packets belonging to different flows.

In contrast, hard state information is created by signalling.

Soft state does not have to be present for the network to function, but its presence assists the router to handle packets belonging to flows.

Congestion Control: Connectionless Flows

- *sequence of packets sent between an O-D pair*
- *maintain soft state at the routers*



Congestion Control: Taxonomy

Router-centric versus host-centric

- *Router-centric: each router decides*
  - *which packets to forward*
  - *which packets to drop*
  - *the rate at which hosts offer traffic.*
- *Host-centric: the end hosts*
  - *observe network conditions*
  - *adjust their behaviour accordingly.*

Router- and host-centric strategies are often combined.

Congestion Control: Taxonomy

Reservation-based versus feedback-based

- *Reservation-based: QoS service model*
  - *the end host asks the network for a certain amount of capacity when the flow is established*
  - *each router allocates resources: buffers & bandwidth*
  - *if capacity is not available the flow is rejected.*
- *Feedback-based: best-effort service model*
  - *the end hosts start to send data without reserving resources*
    - *implicit CN: the host slows down if it observes congestion through for example lost packets*
    - *explicit CN: a router sends a throttle message to a host.*

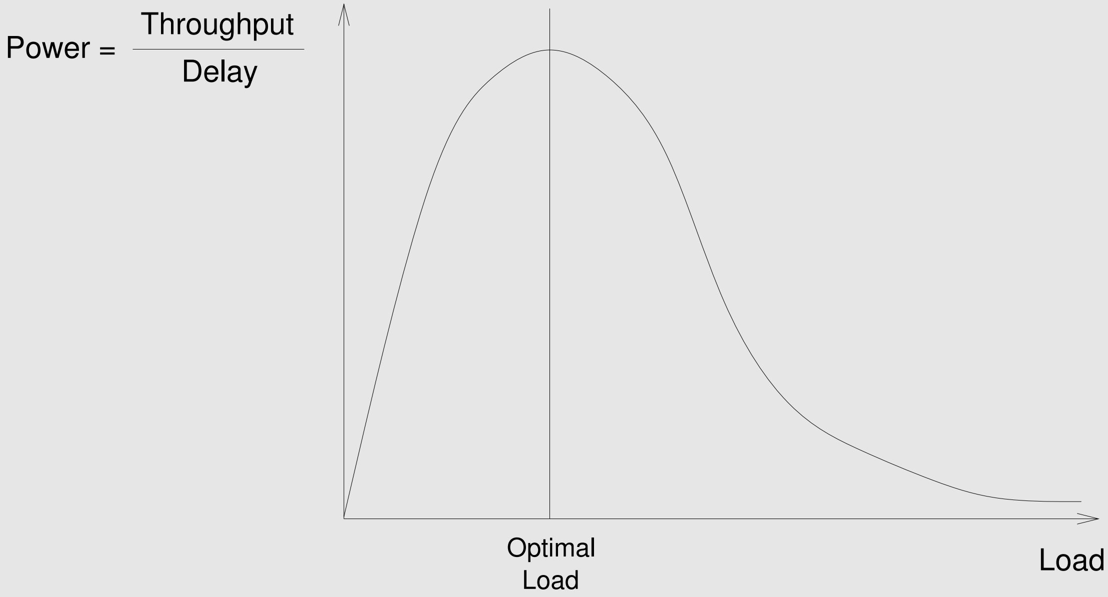
Congestion Control: Taxonomy

Window-based versus rate-based

- *Window-based*
  - *the receiver advertises a window which states how much buffer space the receiver has & which limits how much data the sender can send.*
- *Rate-based: QoS service model*
  - *the sender makes a reservation for x bps*
  - *each router along the path determines if it can support the rate, given the other flows in progress on the router*
  - *supports different qualities of service.*

Congestion Control: Effectiveness

The power metric measures the effectiveness of a resource allocation scheme.



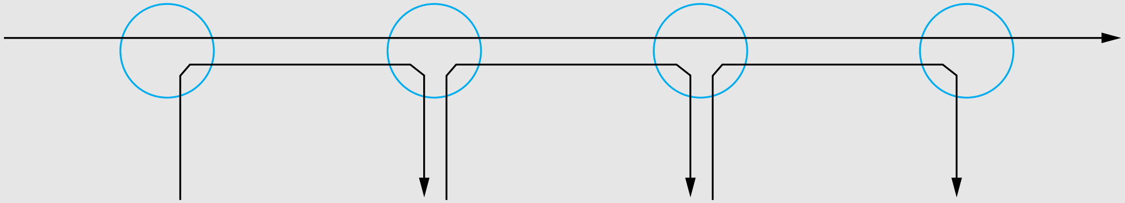
Model a router as an M/M/1 queue

$$W = Q/\lambda = \rho/(1-\rho)/\lambda$$

where $\rho = \lambda/\mu$. The power $Z = \rho(1-\rho)\mu^2$ has a maximum at $\rho = 0.5$.

Congestion Control: Fairness

Consider one 3-hop flow competing with three 1-hop flows



Assume that all paths have equal lengths. The fairness index is

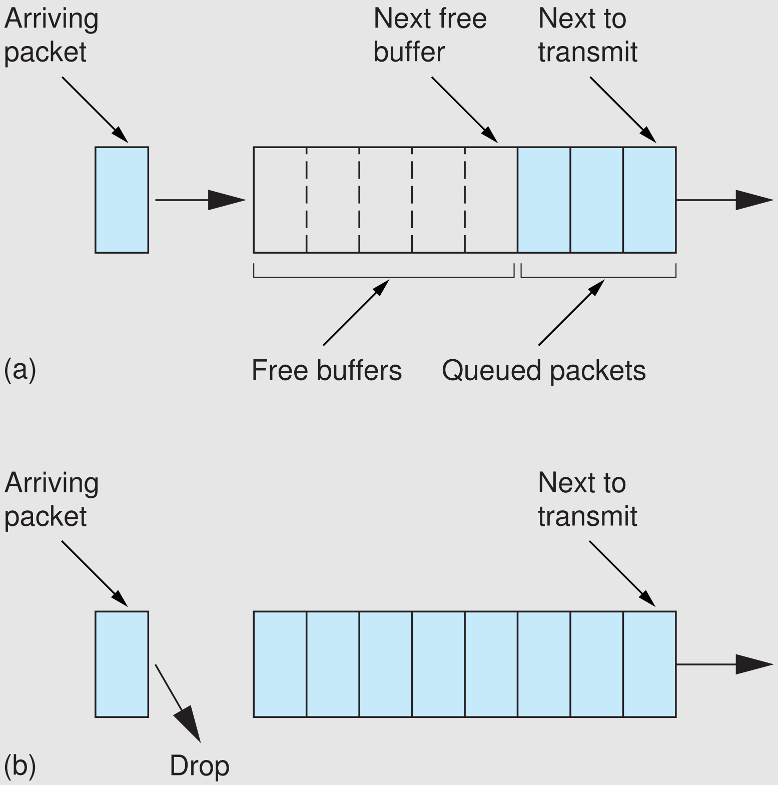
$$f(x_1,\ldots,x_n) = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n\sum_{i=1}^{n} x_i^2}$$

where x_i is the throughput on path i & $0 \le f(\cdot) \le 1$ where 1 corresponds to the greatest fairness.

Exercise: let $\sum_{i=1}^{n} x_i = F$. Then $f(\cdot)$ is maximal when $x_i = F/n$ and $f(F/n,\ldots,F/n) = 1$.

Queueing Disciplines: First-In-First-Out

FIFO with tail drop is widely used in routers



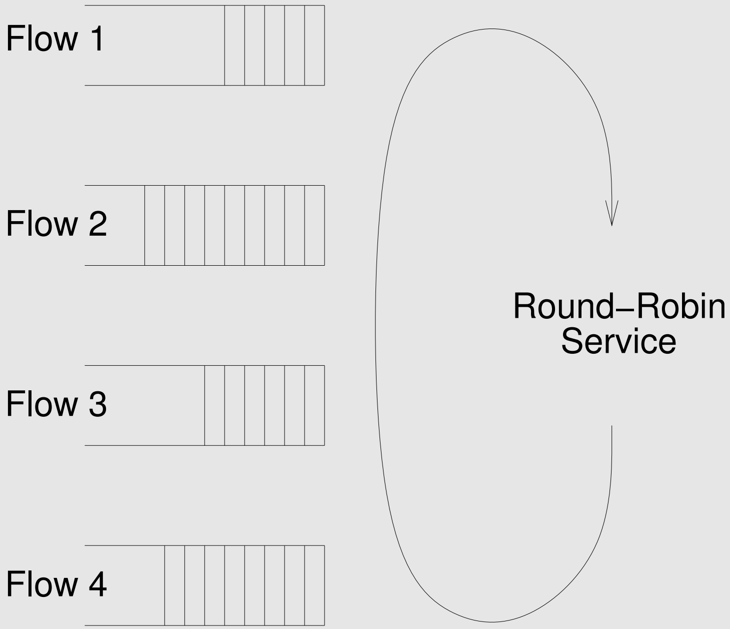
- *does not discriminate between traffic sources*

Queueing Disciplines: Priority Queuing

No service guarantees are made to any priority class: high-priority packets get better service

- *mark each packet with a priority in the TOS field of the IP header*
- *the routers implement multiple queues, one for each priority class*
- *the router transmits packets out of the highest-priority non-empty queue.*

Queueing Disciplines: Fair Queuing



- *packets from different flows are sent to different queues*
- *no flow captures more than its share of the capacity*
- *packets are dropped from full queues.*

Queueing Disciplines: Fair Queueing

Problem: packets are not all of the same length

- *really want bit-by-bit round robin*
- *it is not feasible to interleave bit transmissions: the link is scheduled on a packet basis*
- *simulate bit-by-bit RR by determining when a packet would finish if it were sent bit-by-bit.*

Queueing Disciplines: Fair Queueing

For a single flow

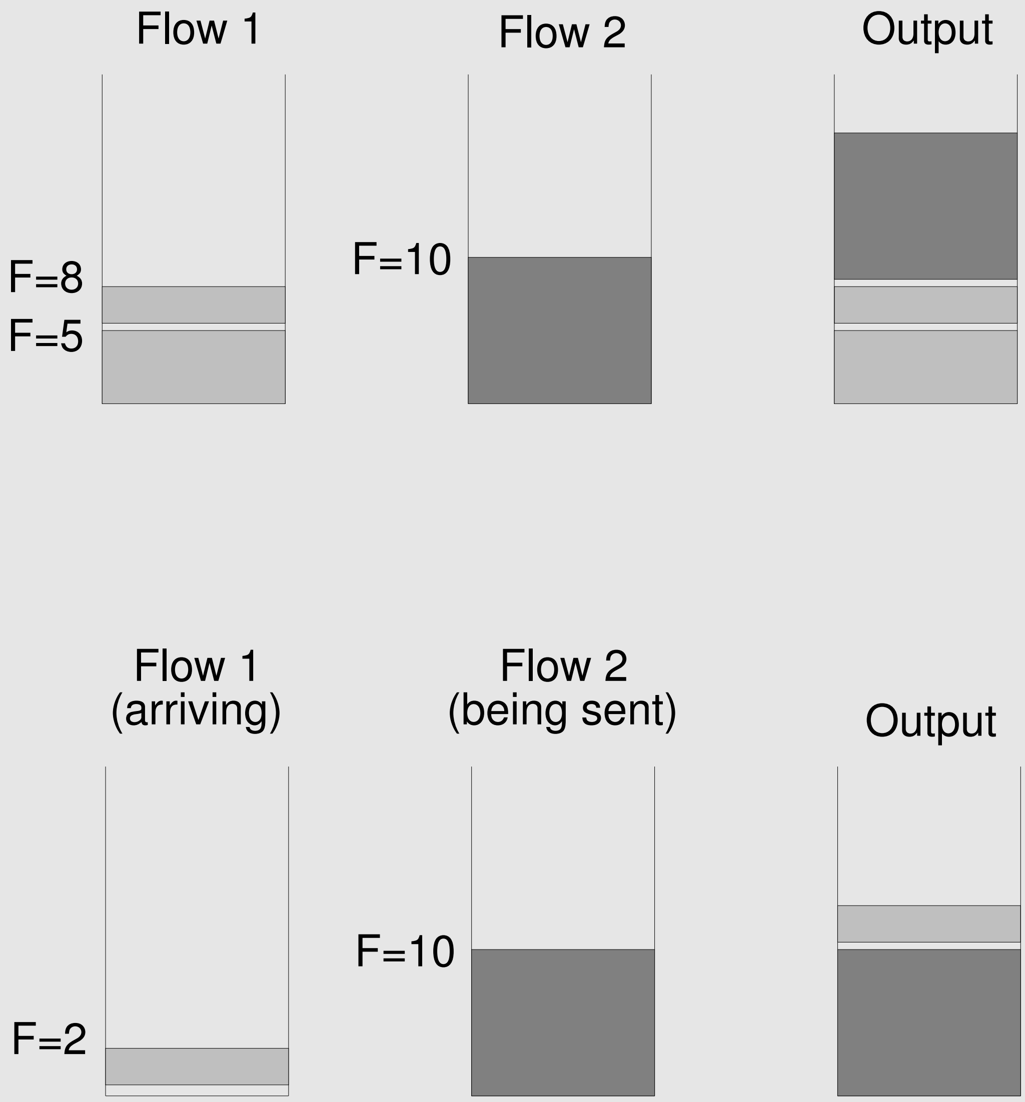
- *suppose the clock ticks each time a bit is transmitted*
- *let P_i denote the length of packet i*
- *let S_i denote the time when packet i starts transmission*
- *let F_i denote the time when packet i ends transmission*
- $F_i = S_i + P_i$
- *S_i: when does the router start transmitting packet i?*
  - *if packet i arrived before the router completed packet $i - 1$ from this flow, then packet i starts at time F_{i-1} immediately after last bit of packet $i - 1$*
  - *if packet i arrived at time A_i after the router completed packet $i - 1$, then packet i starts at time A_i*
- $F_i = \max(F_{i-1}, A_i) + P_i$.

Queueing Disciplines: Fair Queueing

- *For multiple flows*
  - *calculate F_i for each packet that arrives on each flow*
  - *F_i is the time that packet i would complete if it was transmitted using bit-by-bit RR*
  - *treat all F_i's as timestamps*
  - *the next packet to transmit is the one with the lowest timestamp.*
- *This scheduling method is not perfect: it does not preempt the packet currently being transmitted.*

Queueing Disciplines: Fair Queueing

- *Example*



TCP Congestion Control: Overview

TCP sends packets into the network without any prior resource reservation & reacts to any observed results which indicate congestion

- *assumes a best-effort network: FIFO or FQ routers*
- *each source independently determines the network capacity*
- *uses implicit feedback: ICN*
- *the ACKs pace transmission: self-clocking.*

How does TCP

- *determine the available capacity in the first place?*
- *adjust to changes in the available capacity?*

Additive Increase/Multiplicative Decrease

TCP adjusts the rate at which traffic is offered to the network to adapt to changes in the available capacity.

Introduce a new state variable `CongestionWindow` *for each TCP connection which limits how much data the source has in transit*

- *used to be*

  $(\mathtt{LastByteSent} - \mathtt{LastByteAcked}) \leq \mathtt{AdvertisedWindow}$

  $\mathtt{EffectiveWindow} = \mathtt{AdvertisedWindow} - (\mathtt{LastByteSent} - \mathtt{LastByteAcked})$

- *now*

  $\mathtt{MaxWin} = \mathtt{MIN}(\mathtt{CongestionWindow}, \mathtt{AdvertisedWindow})$

  $\mathtt{EffectiveWindow} = \mathtt{MaxWin} - (\mathtt{LastByteSent} - \mathtt{LastByteAcked})$

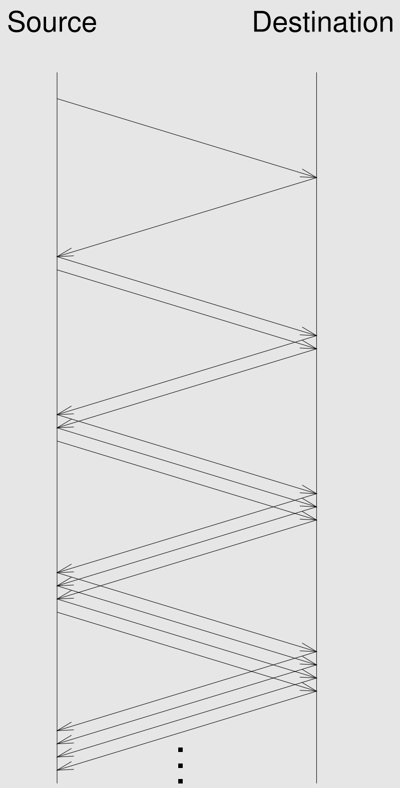
- *increase/decrease the* `CongestionWindow` *when congestion decreases/increases.*

Additive Increase/Multiplicative Decrease

- *Question: how does the source determine if the network is congested?*
- *Answer: a timeout occurs*
  - *a timeout signals that a packet was lost*
  - *packets are seldom lost due to transmission errors*
  - *lost packets imply congestion.*

Additive Increase/Multiplicative Decrease

- *increment the* `CongestionWindow` *by one segment per RTT: additive increase*
- *divide the* `CongestionWindow` *by two whenever a timeout occurs: multiplicative decrease.*



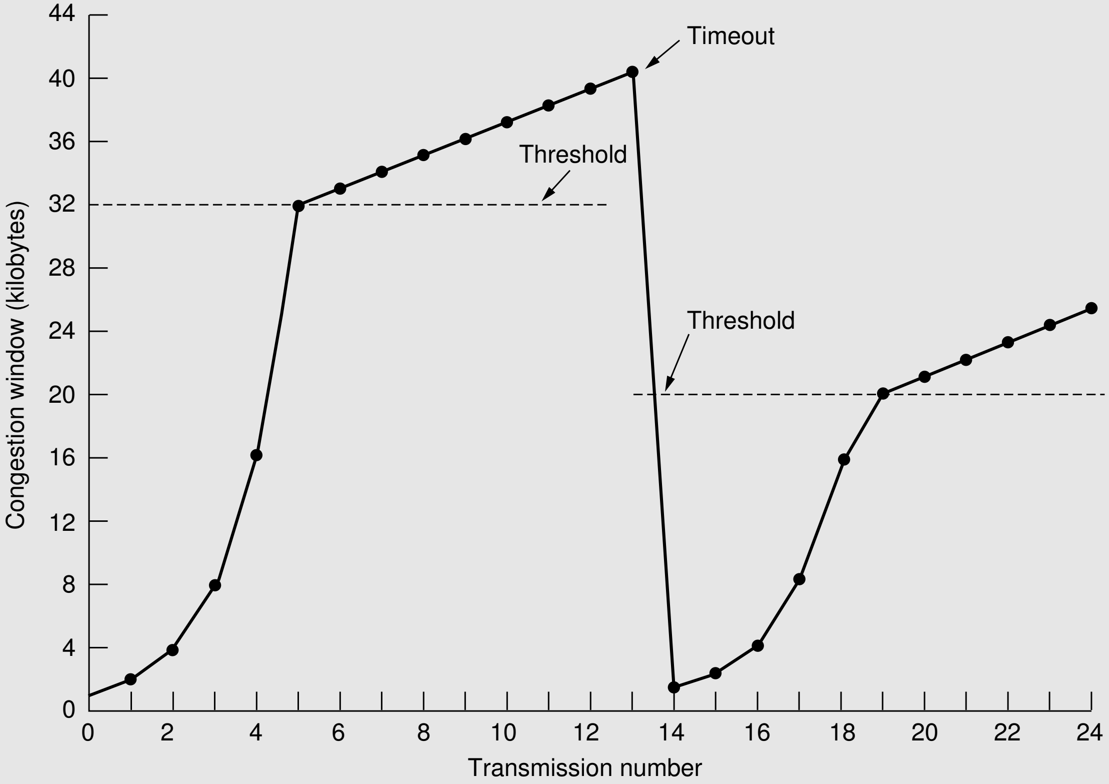
Additive Increase/Multiplicative Decrease

In practice: increment `CongestionWindow` *a little for each ACK received*

$$\mathtt{Increment} = \mathtt{MSS} * (\mathtt{MSS}/\mathtt{CongestionWindow})$$
$$\mathtt{CongestionWindow} + = \mathtt{Increment}$$

TCP is said to be in the collision avoidance phase when the `CongestionWindow` *increases by 1 segment per RTT.*

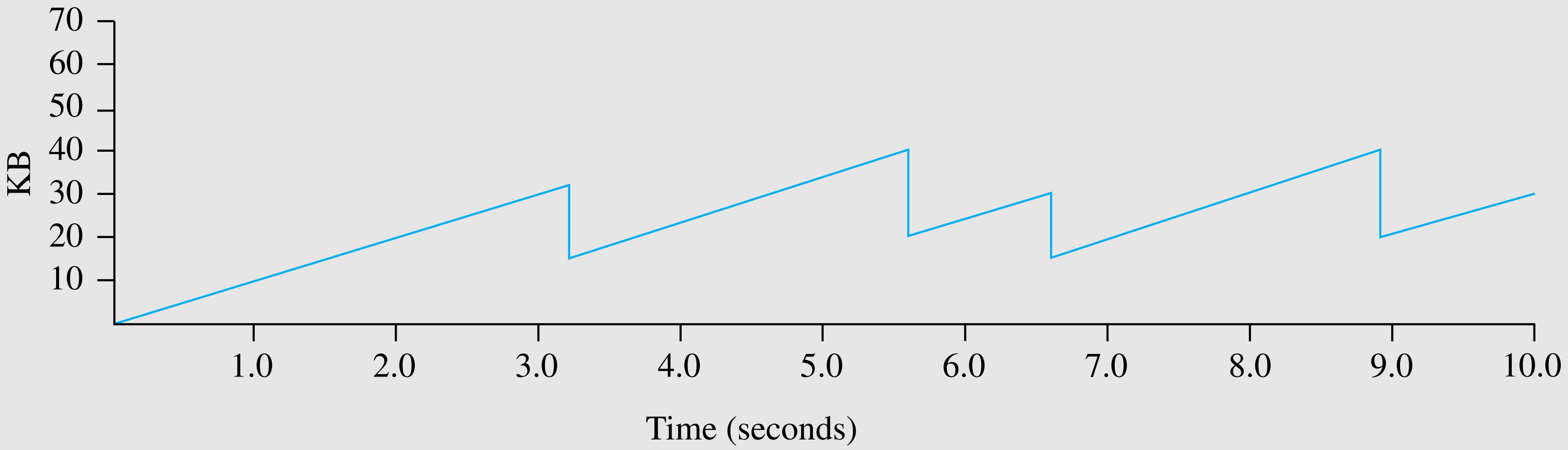
Collision Avoidance



TCP is said to be in the collision avoidance phase when the `CongestionWindow` *increases by 1 segment per RTT.*

Additive Increase/Multiplicative Decrease

Example trace of TCP in the CA phase: the `CongestionWindow` *exhibits a sawtooth behaviour showing additive increase, multiplicative decrease.*

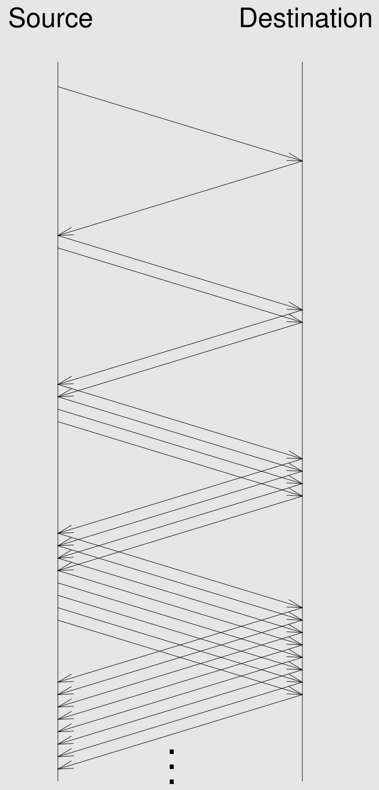


Additive increase/multiplicative decrease is necessary for a congestion control mechanism to be stable.

TCP Congestion Control: Slow Start

How does TCP determine the available capacity in the first place?

- *begin with* `CongestionWindow` *= 1 packet*
- *increment the* `CongestionWindow` *by 1 packet for each ACK thus doubling the* `CongestionWindow` *each RTT.*



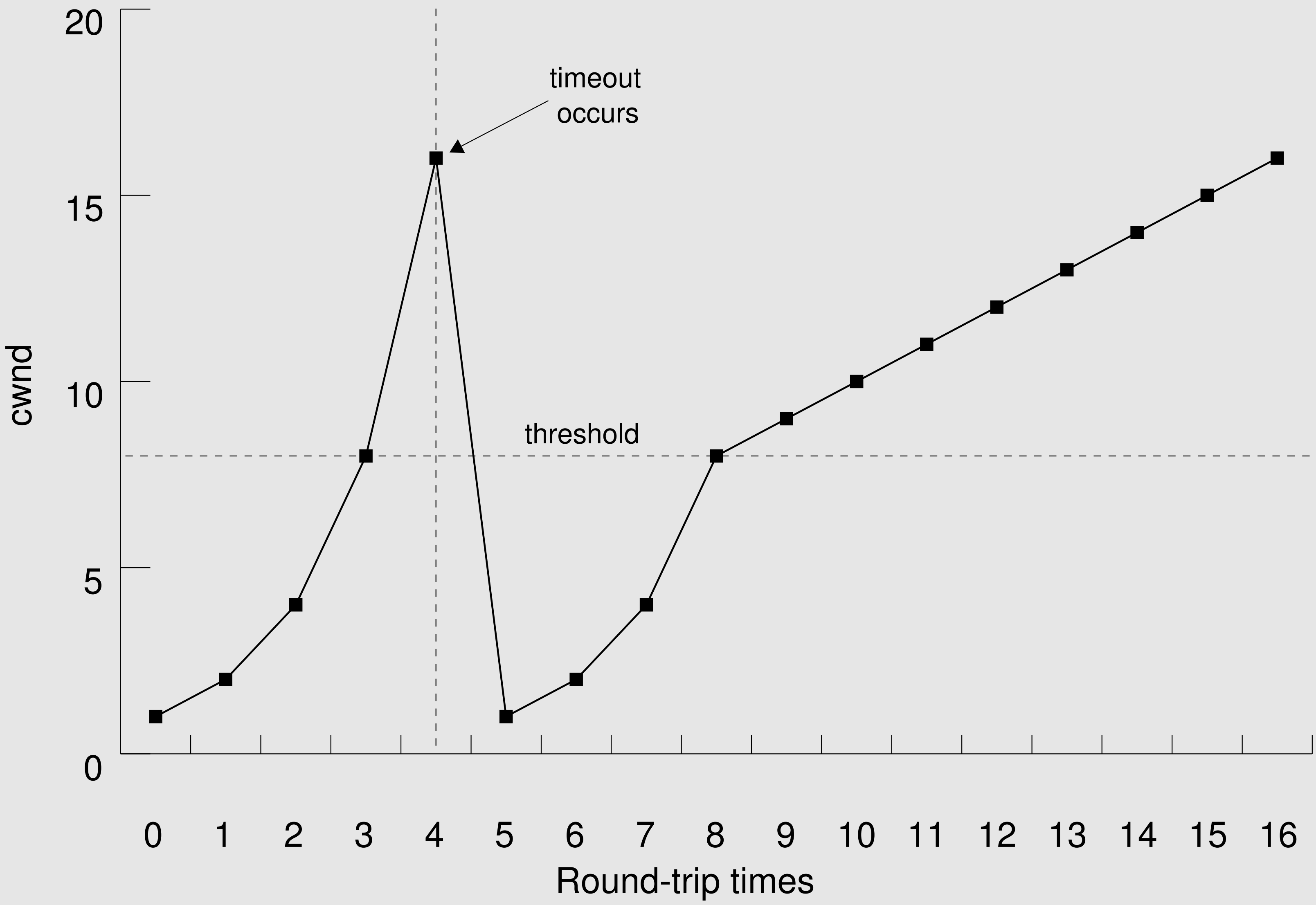
TCP Congestion Control: Slow Start

In the slow start phase the `CongestionWindow` *grows exponentially but it does grow more slowly than it would if all the packets in the* `AdvertisedWindow` *were sent in one go: hence the name slow start.*

Slow start is used

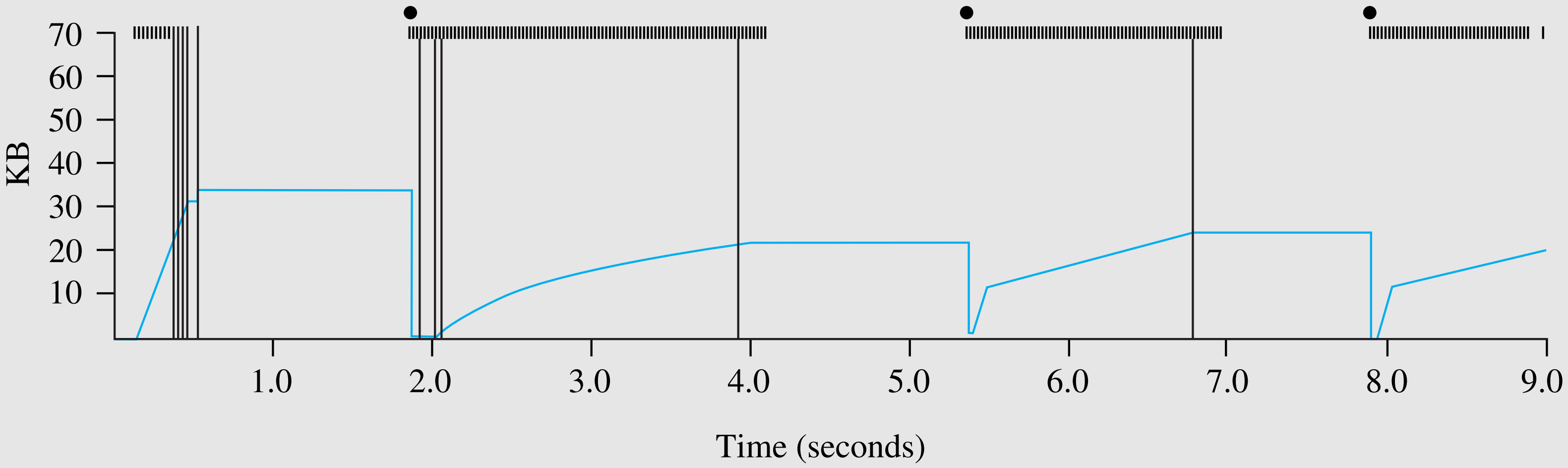
- *when a connection first starts*
- *when a re-transmitted packet is acknowledged after a timeout.*

TCP Congestion Control: Slow Start



TCP Congestion Control: Slow Start

Example trace



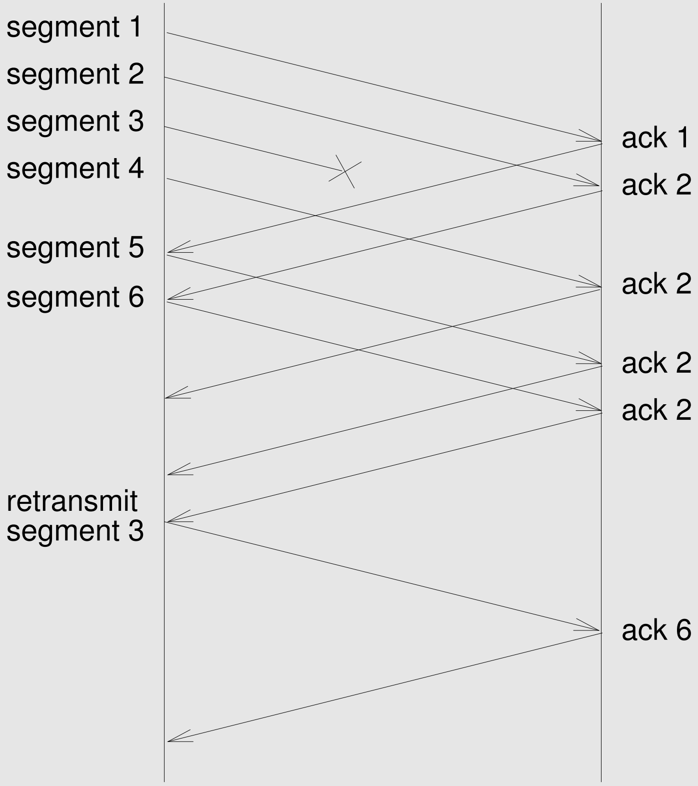
Problem: packets are lost & the corresponding ACKs are not sent – these give rise to long time periods during which the link is idle waiting for a timeout to occur.

Solution: the fast re-transmit algorithm.

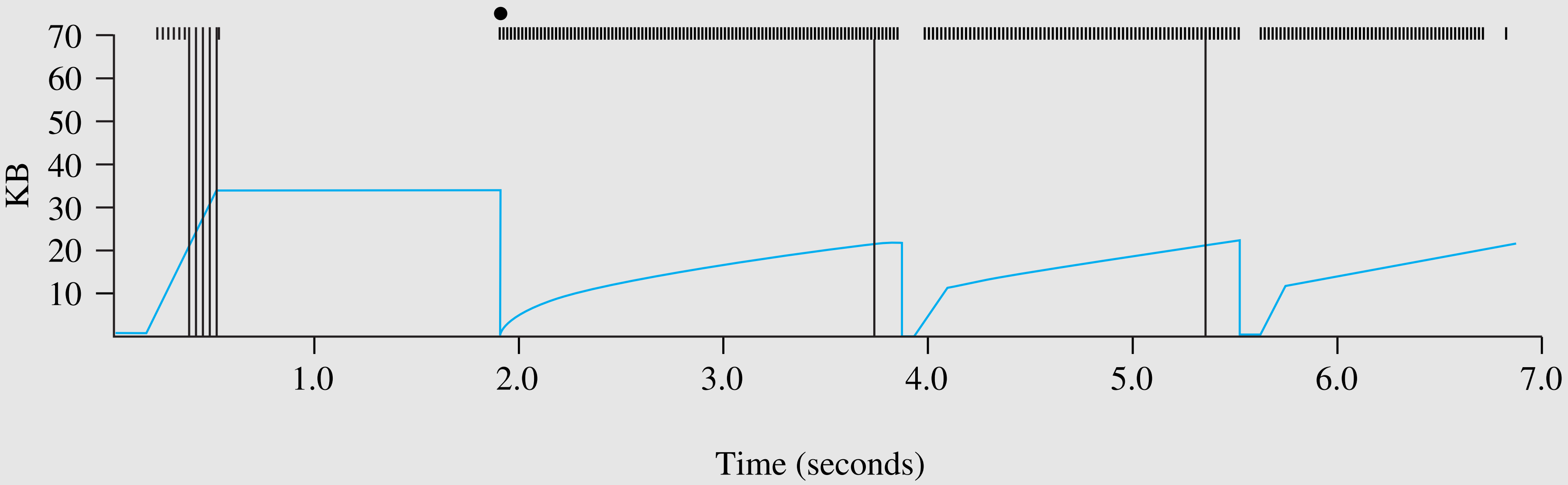
TCP Congestion Control: Fast Re-transmit

Problem: coarse-grain TCP timeouts lead to idle periods

- *fast re-transmit uses triple duplicate ACKs to trigger re-transmission of the "lost" packet.*



TCP Congestion Control: Fast Recovery



Another improvement: if the fast re-transmit is successfull

- *do not re-begin the slow start phase*
- *go to the CA phase and set the* `CongestionWindow` *to half the last successful* `CongestionWindow`*.*

Additive Increase/Multiplicative Decrease

Let $w(n)$ denote the size of the congestion window during the n^{th} round trip.

For a TCP Reno connection in the CA phase for which all packet losses are detected by triple ACKs and where fast re-transmit applies

$$w(n+1) = (w(n)+1)(1-B)^{w(n)} + \frac{w(n)}{2}\left(1-(1-B)^{w(n)}\right)$$

where B is the packet loss probability

- *the first term models the increase of the congestion window by 1 every RTT*
- *the second term models the halving of the congestion window when a packet loss occurs.*

Additive Increase/Multiplicative Decrease

$$w(n+1) = (w(n)+1)(1-B)^{w(n)} + \frac{w(n)}{2}\left(1-(1-B)^{w(n)}\right)$$

The average window size w is obtained by letting $n \to \infty$

$$w = (w+1)(1-B)^{w} + \frac{w}{2}\left(1-(1-B)^{w}\right).$$

A lower limit of $w \geq 1$ and an upper limit $w \leq w_{\max}$ are applied.

Additive Increase/Multiplicative Decrease

$$w = (w+1)(1-B)^w + \frac{w}{2}(1-(1-B)^w)$$

For small values of B *the above equation becomes*

$$w \sim (w+1)(1-Bw) + \frac{w}{2}Bw$$

which yields a quadratic equation

$$Bw^2 + 2Bw - 2 = 0$$

which has the positive solution

$$w = \sqrt{1+2/B} - 1 \sim 1.414/\sqrt{B}.$$

This derivation of the $w \propto 1/\sqrt{B}$ *law is simple compared to other derivations.*

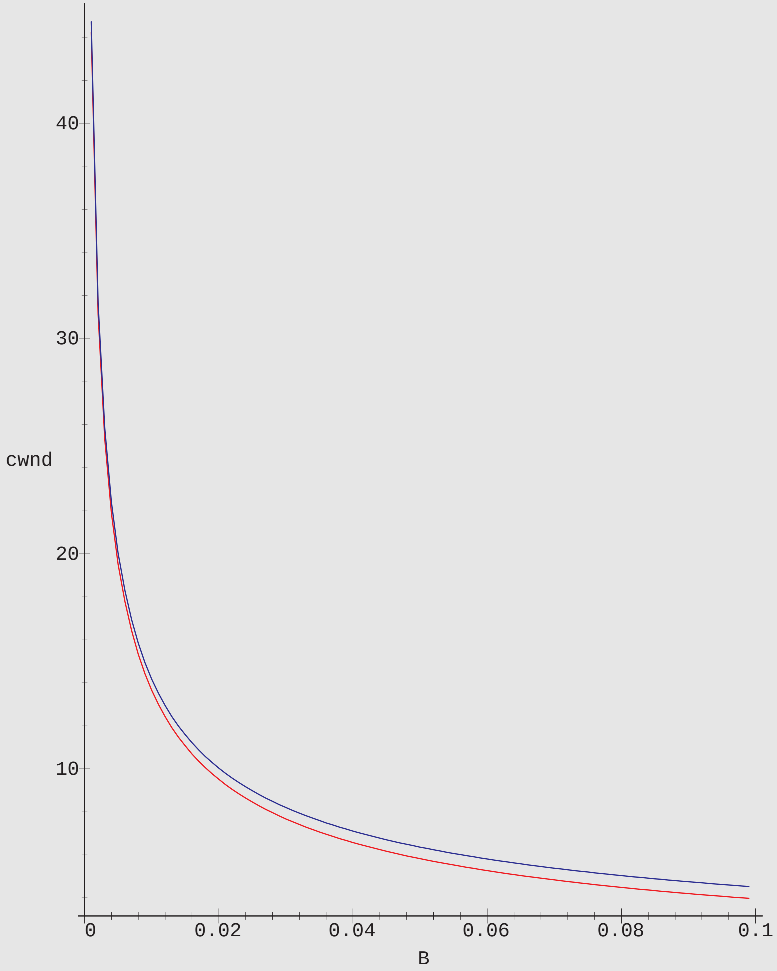
Additive Increase/Multiplicative Decrease

```
> B:=’B’:w:=’w’:  # clear the variables
> X:=array(1..99,1..2):
> Y:=array(1..99,1..2):
> eqn:=w-(w+1)*(1-B)^w-w/2*(1-(1-B)^w);
```

$$eqn := w - (w+1)\,(1-B)^{w} - \frac{1}{2}\,w\,(1-(1-B)^{w})$$

```
> for i from 1 to 99 do
>     B:=i/1000; # the packet loss probability
>     x:=fsolve(eqn=0,w); # numerical equation solver
>     X[i,1]:=B;
>     X[i,2]:=x;
>     Y[i,1]:=B;
>     Y[i,2]:=sqrt(2/B) # the approximate solution
> od:
> data_list1:=convert(X,’listlist’):
> exact_solution:=plot(data_list1,color=red):
> data_list2:=convert(Y,’listlist’):
> approx_solution:=plot(data_list2,color=blue):
> with(plots):
> plotsetup(ps,...):
> display([exact_solution,approx_solution],...);
```

Additive Increase/Multiplicative Decrease



TCP

TCP provides reliable transport of user data. Packet losses are detected in two ways: by the reception of three duplicate acknowledgments (DA) or by a re-transmission timeout (TO).

The transmissions proceed in five "states"; the first visit to slow start (state SS′), subsequent visits to slow start (state SS), congestion avoidance (state CA), DA re-transmission (state DR) and TO re-transmission (state TR).

- *SS′* *A transmission starts in SS′ with a congestion window size $w = 1$. For each acknowledgment received, w is increased by one. This means that over one window round trip time (RTT), during which w packets are sent, w will double in size up to w_{rx}, the window size advertised by the receiver, which imposes a maximum limit. The increase in w over time is thus exponential until a packet loss is detected (see loss below).*
- *SS* *Subsequent visits to slow start SS are similar to the first visit SS′ except the window size starts at size $w = 2$ and stops growing (exponentially) when it has reached the threshold window size w_{th}. At this point the state changes from SS to CA.*
- *CA* *For each acknowledgment received, w is increased by $1/w$. This means that over one window RTT, during which w packets are sent, w will increase by one. The increase in w over time is thus linear until w reaches w_{rx}.*

TCP

If a packet loss is detected in SS$'$, SS or CA, the threshold window size w_{th} is set equal to half the value of w, but at least two.

- DR *If a loss is detected by DA, the state changes to DR and the packet is re-transmitted immediately (the fast re-transmit algorithm). If the re-transmission is acknowledged, the state is set to CA and w is set to w_{th} (the fast recovery algorithm), otherwise a TO will occur and the state will change to TR.*
- TR *If a loss is detected by TO, the state changes to TR. In TR a timeout scaling factor is doubled (up to its maximum value of 64) and the packet is re-transmitted. If a re-transmission is acknowledged, the state is set to SS and w is reset to one, otherwise the procedure is repeated until an acknowledgment is received or, in some TCP implementations, until a maximum number of re-transmission attempts has been made.*

The re-transmission timeout is adjusted by on line measurements and set to the mean plus four times the mean deviation of the RTT (Jacobson's algorithm). Only packets which are successful in their first transmission attempt are used in the on line measurements and only these packets cause the timeout scaling factor to be reset to one (Karn's algorithm).

ACK packets in TCP Reno contain an indication of the next user packet expected. The acknowledgment of packet n may therefore acknowledge all packets up to and including packet n by indicating that it expects packet $n+1$. Because of this, TCP can cope with some loss of ACKs; the larger the window size the smaller the impact of the losses.

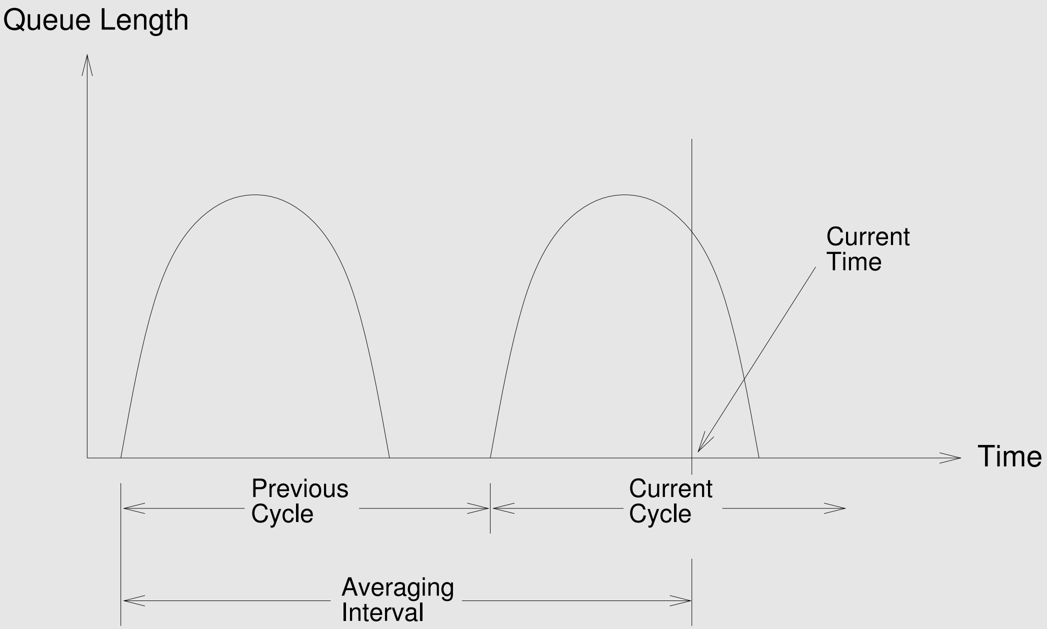
Congestion Avoidance: Overview

- TCP’s strategy
  - control congestion once it happens
  - repeatedly increase the load in an effort to find the point at which congestion occurs, & then back off.
- Alternative strategy
  - predict when congestion is about to happen, & reduce the rate at which hosts send data just before packets start being discarded
  - we call this congestion avoidance, to distinguish it from congestion control.
- Two possibilities
  - router-centric: DECbit & RED gateways
  - host-centric: TCP Vegas.

Congestion Avoidance: DECbit

Add a congestion bit to each packet header. Each router

- *monitors its average queue length (a measure of its load) over last busy+idle cycle plus the current busy cycle*



- *sets the congestion bit if the average queue length is greater than 1 when the packet arrives.*

Congestion Avoidance: DECbit

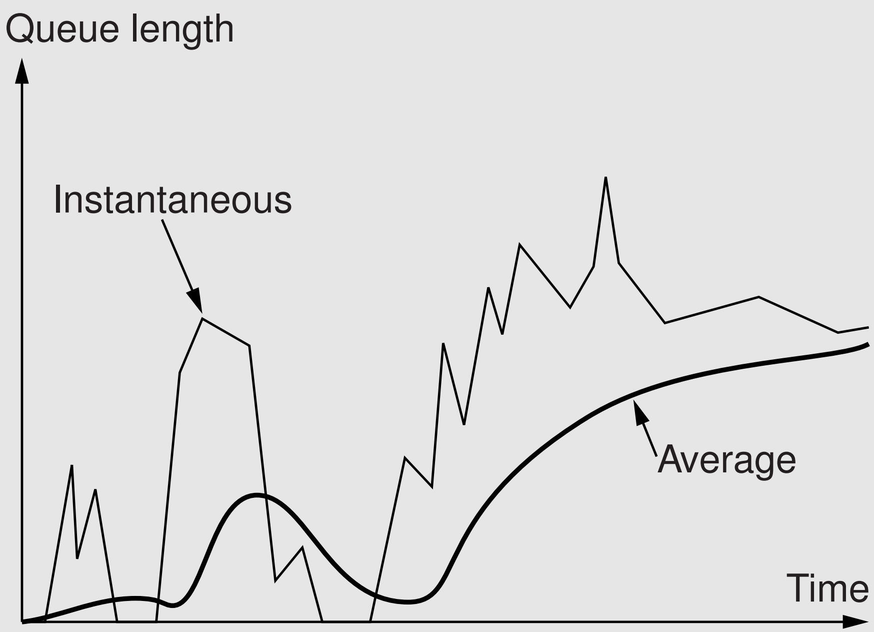
The source and the destination hosts cooperate to implement additive increase/multiplicative decrease

- *the destination host copies the congestion bit into an ACK which it sends back to the source host*
- *the source host records how many packets had their DECbits set*
  - *if less than 50% of the last window's worth had their bit set, then increase the* `CongestionWindow` *by 1 packet*
  - *if 50% or more of the last window's worth had their bit set, then decrease the* `CongestionWindow` *by a factor 0.875.*

Random Early Detection

- *RED is a form of implicit congestion notification*
  - *drop a packet: TCP will timeout*
  - *RED could implement explicit congestion notification by marking the packet.*
- *Random early detection*
  - *rather than wait for the queue to become full, drop each arriving packet with some drop probability whenever the queue length exceeds some drop level.*
- *The RED algorithm*
  - *compute the average queue length*
    $\mathtt{AvgLen} = (1 - \mathtt{Weight}) * \mathtt{AvgLen} + \mathtt{Weight} * \mathtt{SampleLen}$
    - $0 < \mathtt{Weight} < 1$ *(usually 0.002)*
    - $\mathtt{SampleLen}$ *is the queue length each time a packet arrives.*

Random Early Detection

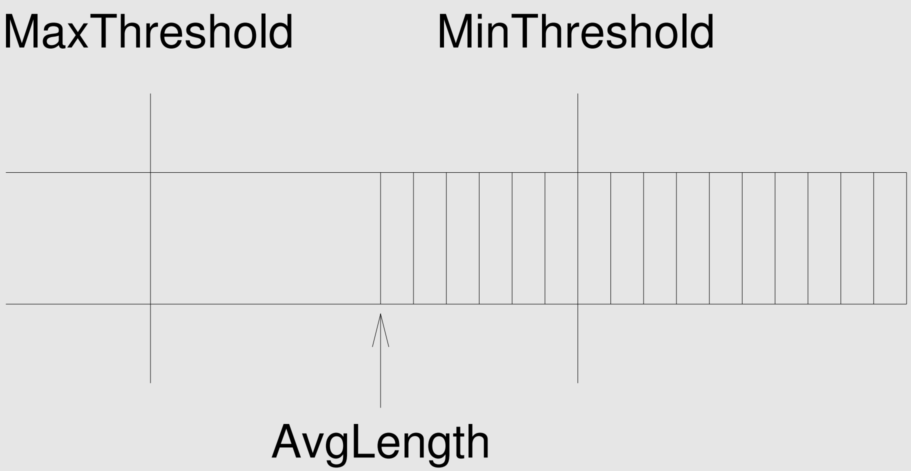


The weighted running average measure `AvgLen` *of the queue length detects long lived congestion & filters out short term congestion caused by the bursty nature of the Internet traffic.*

Random Early Detection

FIFO uses tail drop, but RED has two queue length thresholds

```
if AvgLen $<=$ MinThreshold then enqueue the packet
if MinThreshold $<$ AvgLen $<$ MaxThreshold {
  calculate probability P
  drop arriving packet with probability P
}
if AvgLen $>=$ MaxThreshold drop the arriving packet
```



Random Early Detection

The drop probability P is a function of two variables

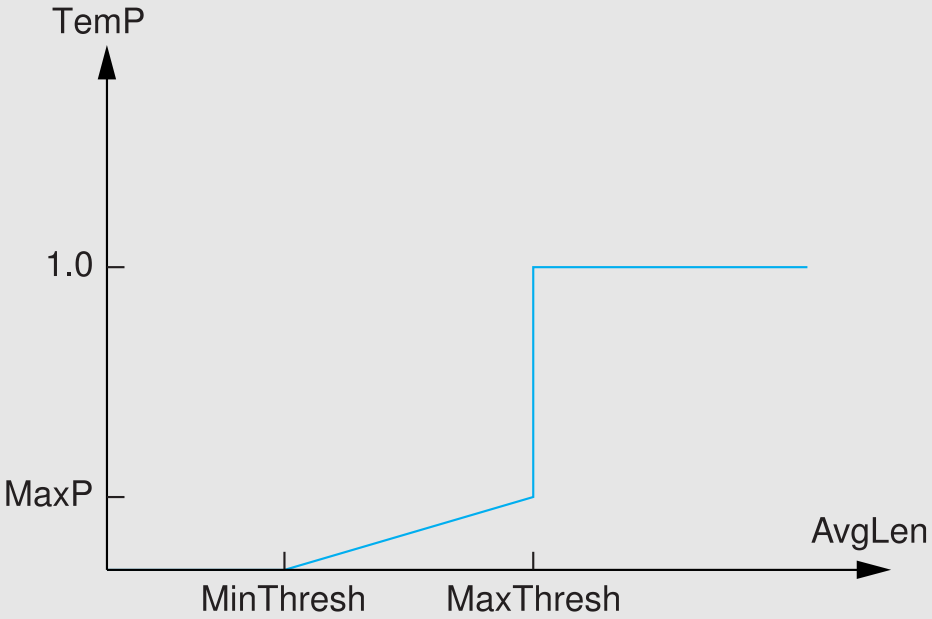
- `AvgLen`, *and*
- `count` *which counts the packets which were queued since the last drop: it keeps track of new packets that have been queued while* `AvgLen` *was between the two thresholds*

$$\begin{aligned}\mathtt{TempP} &= \mathtt{MaxP} * (\mathtt{AvgLen} - \mathtt{MinThreshold})\\ &\quad /(\mathtt{MaxThreshold} - \mathtt{MinThreshold})\\ \mathtt{P} &= \mathtt{TempP}/(1 - \mathtt{count} * \mathtt{TempP})\end{aligned}$$

- `count` *ensures that closely spaced drops are unlikely.*

Random Early Detection

The drop probability P



Random Early Detection

- *The probability of dropping a particular flow's packet(s) is roughly proportional to the share of the bandwidth that flow is currently getting.*
- `MaxP` *is typically set to 0.02, meaning that when the average queue size is halfway between the two thresholds, RED drops roughly one out of 50 packets.*
- *If the traffic is bursty then* `MinThreshold` *should be sufficiently large to allow the link utilization to be maintained at an acceptably high level.*
- *The difference between two thresholds should be larger than the typical increase in the calculated average queue length in one RTT; setting* `MaxThreshold` *to twice* `MinThreshold` *is reasonable for traffic on today's Internet.*

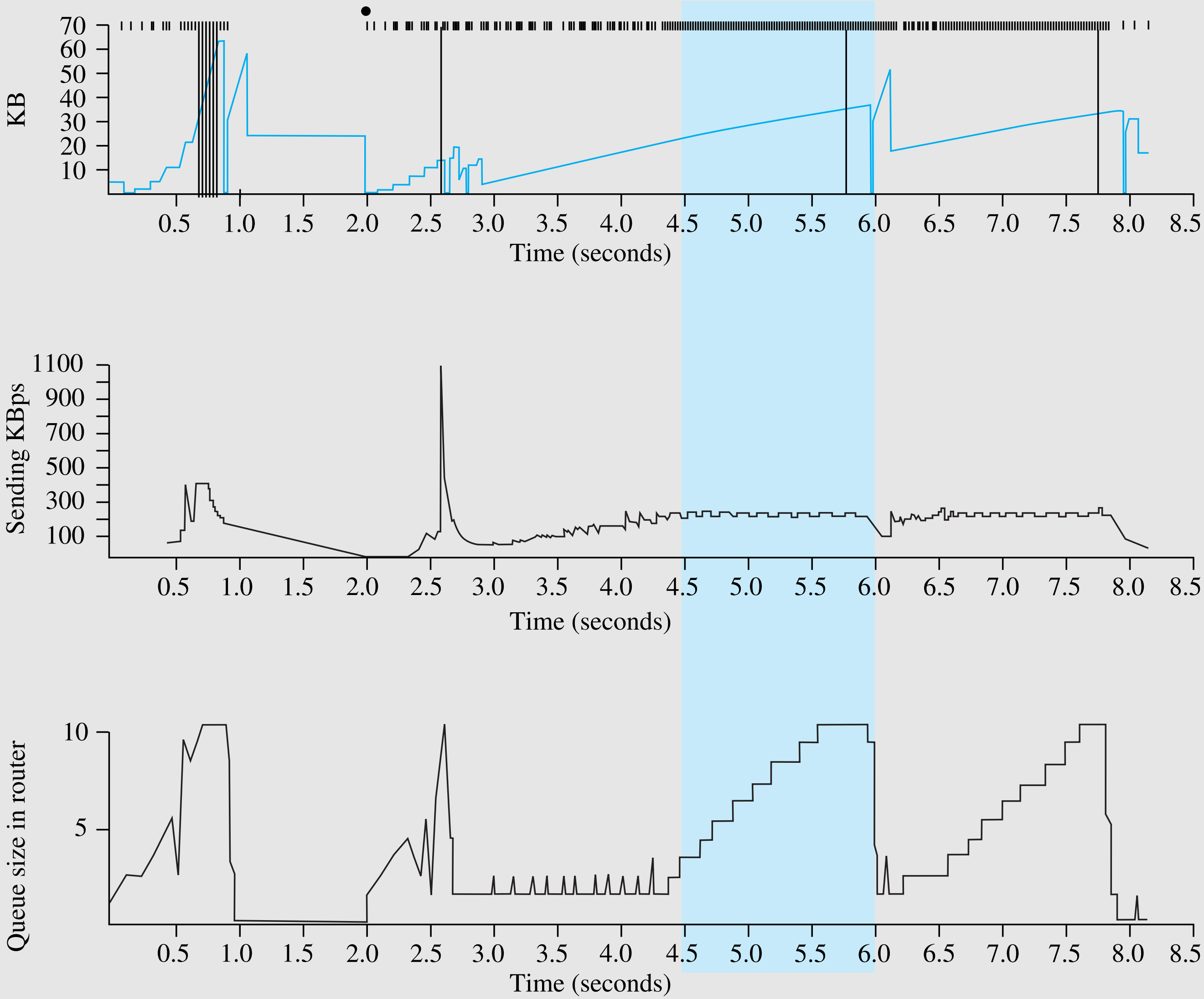
Congestion Avoidance: TCP Vegas

Idea: the source watches for some sign that some router's queue is building up & congestion will happen soon; e.g.

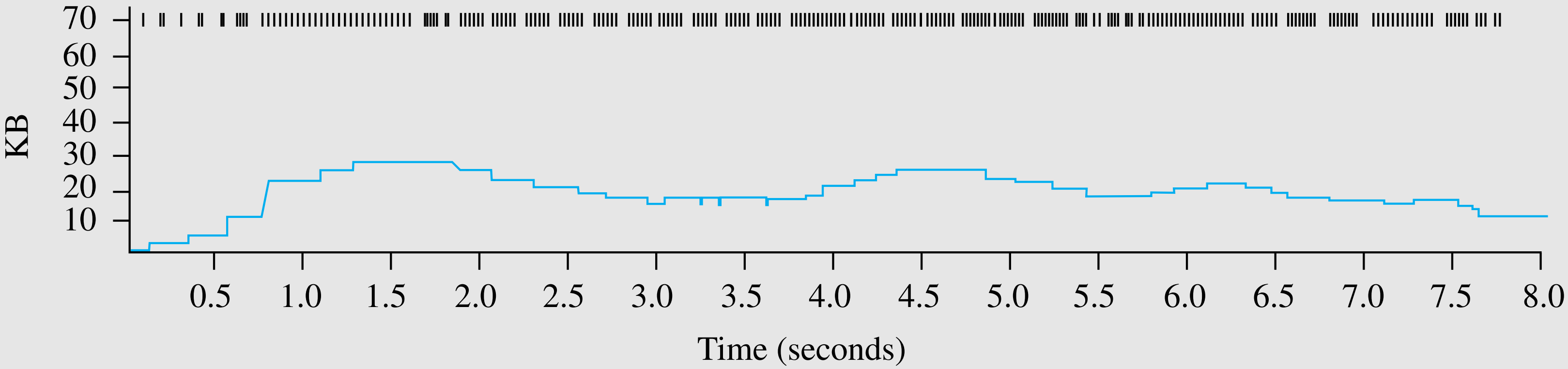
- *the RTT is growing*
- *the sending rate flattens.*

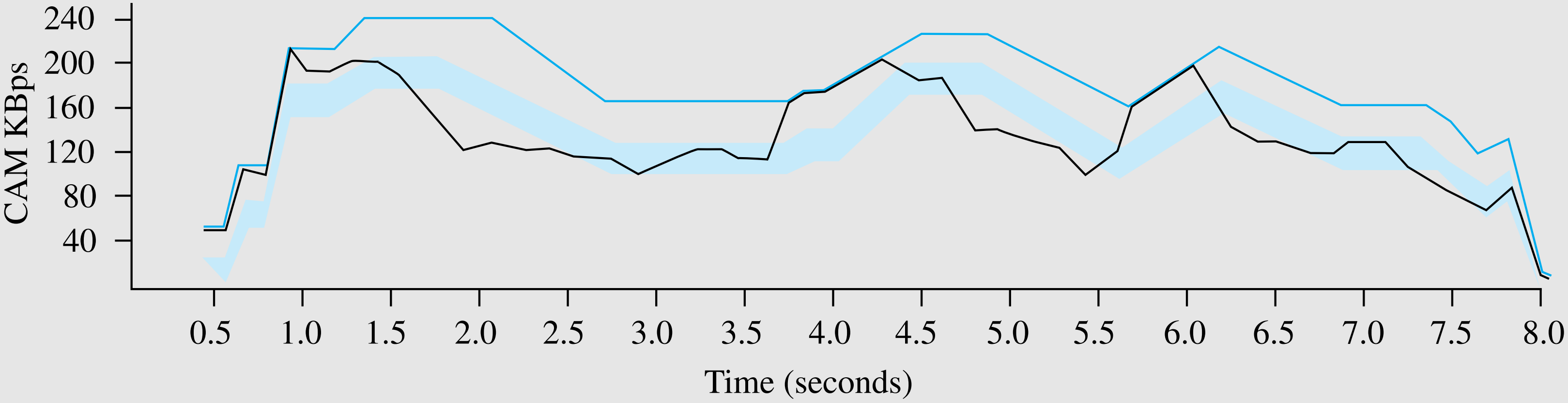
Congestion Avoidance: TCP Vegas

Intuition



Congestion Avoidance: TCP Vegas





Congestion Avoidance: TCP Vegas

The Algorithm

- `BaseRTT` *is a measure of the RTT when the flow is not congested:* `BaseRTT` *is the minimum of all measured RTTs (commonly the RTT of the first packet)*
- *if the connection is not congested then*
  $\texttt{ExpectedRate} = \texttt{CongestionWindow}/\texttt{BaseRTT}$
- *the source measures the current sending rate* `ActualRate` *once per RTT*
- *the source compares* `ActualRate` *with* `ExpectedRate`
  $\texttt{Diff} = \texttt{ExpectedRate} - \texttt{ActualRate}$
  $\texttt{if Diff} < \alpha$ `increase CongestionWindow linearly`
  $\texttt{else if Diff} > \beta$ `decrease CongestionWindow linearly`
  `else leave CongestionWindow unchanged`

Congestion Avoidance: TCP Vegas

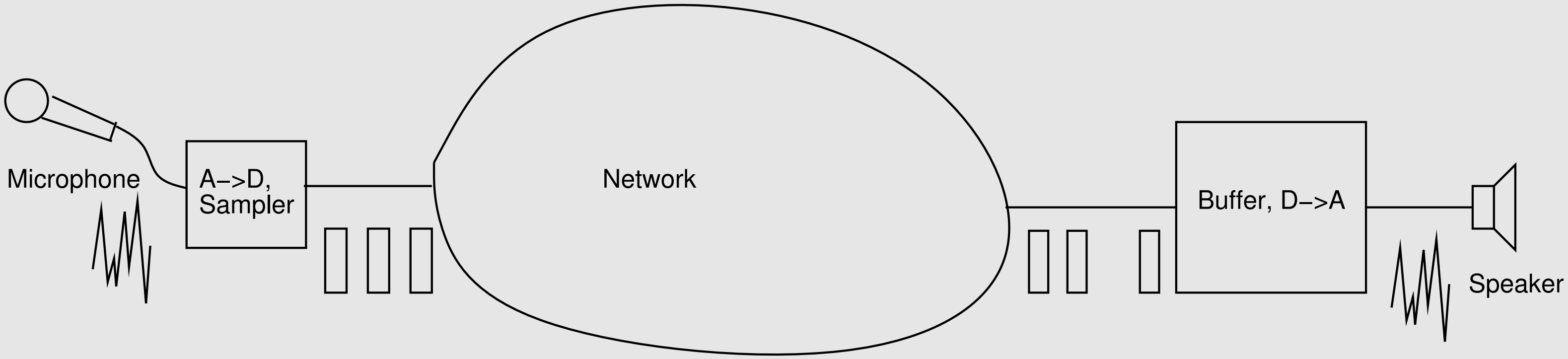
- *Parameters*
  - $\alpha = 1$ *packet*
  - $\beta = 3$ *packets*
- *Even faster retransmit*
  - *keep fine-grained timestamps for each packet*
  - *check for timeout on first duplicate ACK*

QoS: Overview

- *High-speed networks have enabled new applications*
  - *they also need "deliver on time" assurances from the network.*
- *Applications that are sensitive to the timeliness of data delivery are called real-time applications*
  - *voice*
  - *video*
  - *industrial control.*
- *Timeliness guarantees must be implemented by the routers in the network*
  - *end-hosts cannot correct late packets like they can correct for lost packets.*
- *Need more than best-effort*
  - *IETF is standardising extensions to the best-effort service model.*

QoS: Service Model

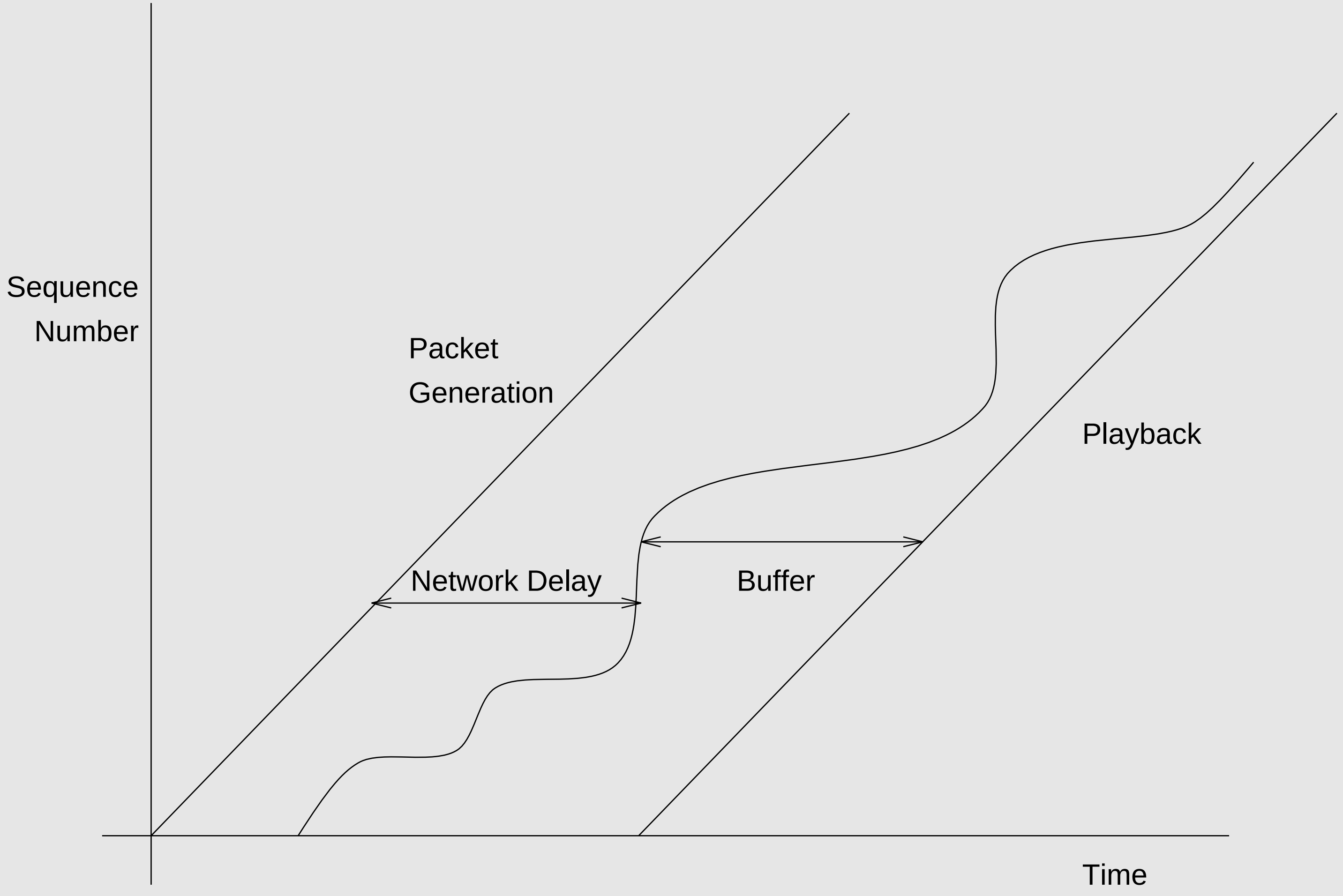
- *Two types of applications*
  - *real-time*
  - *non real-time: elastic.*
- *An example of a real-time application: audio*



  - *sample the analog voice signal once every 125μs*
  - *each sample has a playback time: one every 125μs*
  - *packets experience variable delay in network*
  - *add a constant factor to the playback time: playback point.*

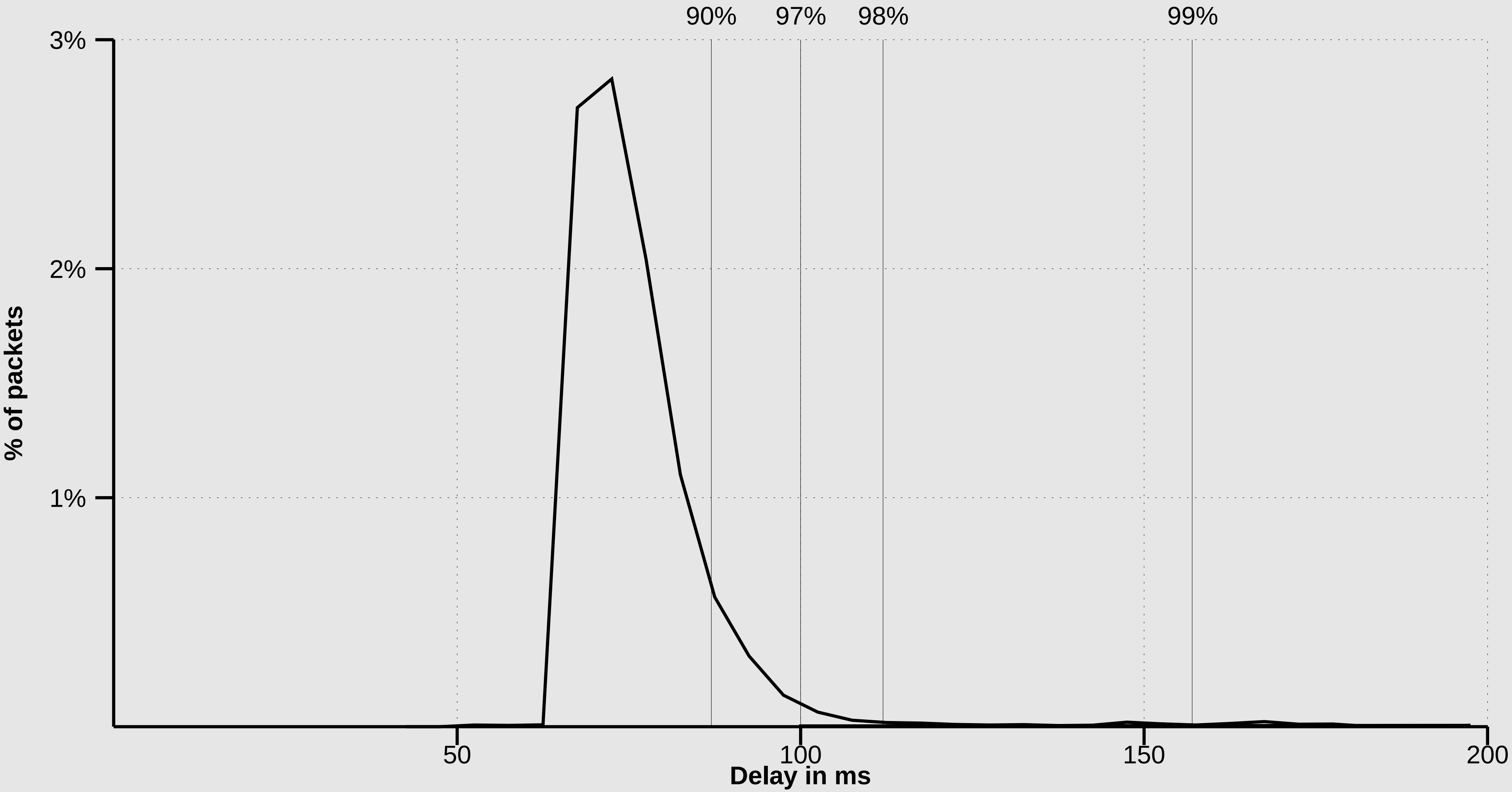
QoS: Service Model

Playback buffer



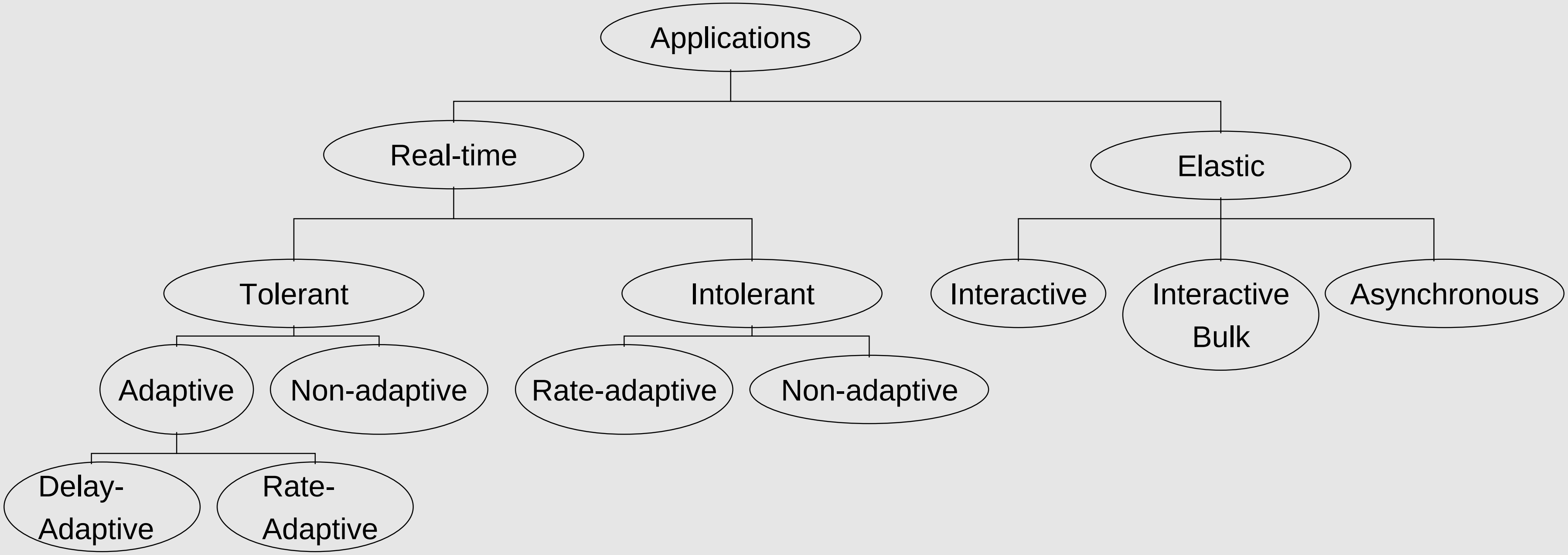
Few packets arrive after the playback time.

Example distribution of delays



QoS: Service Model

A taxonomy of real-time applications



Integrated services specifies how RSVP can reserve resources for two service classes

- *guaranteed: no packets arrive after the playack time*
- *controlled-load: provides a lightly loaded path even if the network is heavily loaded.*

Intserv: Flowspecs

- *Rspec: describes the service requested from the network*
  - *guaranteed: specify the delay target*
  - *controlled-load: specify no parameters.*
- *Tspec: describes the flow's traffic characteristics*
  - *average bandwidth + burstiness: token bucket filter*
    - *token generation rate r per second*
    - *bucket depth B tokens*
    - *must have a token to send a byte*
    - *must have n tokens to send n bytes*
    - *start with no tokens*
    - *accumulate tokens at a rate of r per second*
    - *can accumulate no more than B tokens.*

Intserv: Admission Control

A flow is a set of packets from a single application that have common resource requirements.

The function of admission control

- *decide if a new flow can be supported*
  - *the answer depends on the service class of the new flow.*

Admission control is not the same as policing which ensures that a flow conforms to its Tspec by

- *dropping offending packets, or*
- *marking offending packets to be dropped later if necessary.*

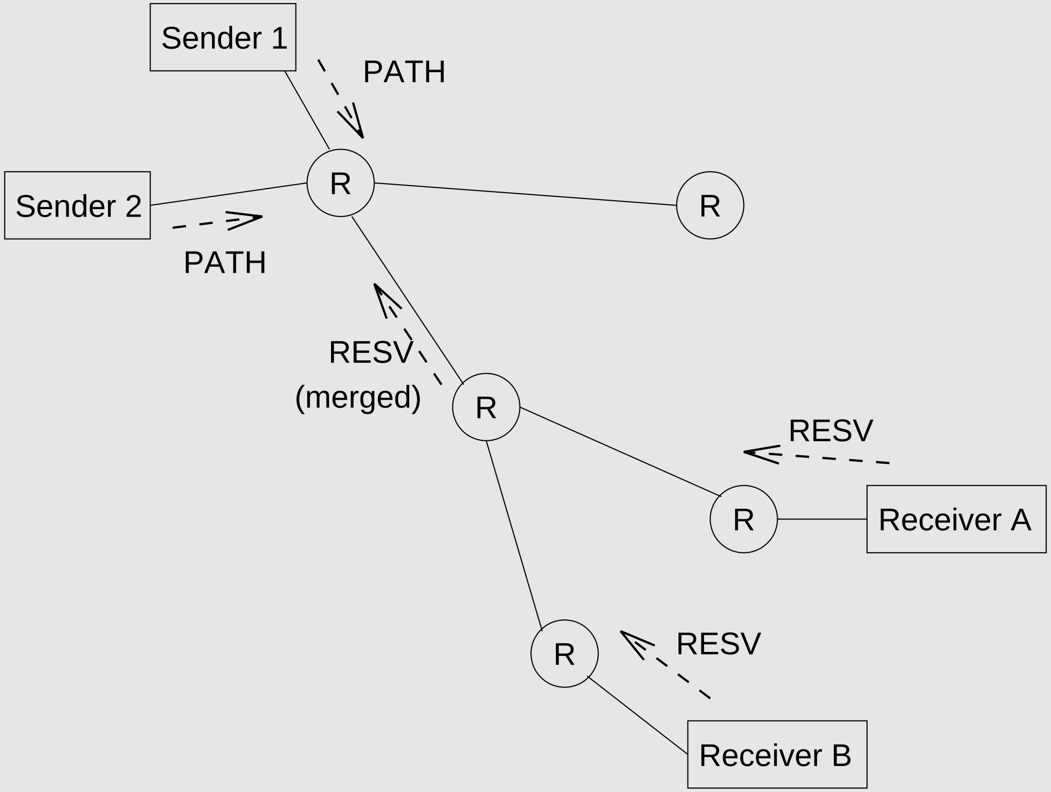
Intserv: Reservation Protocol

- *a proposed Internet standard: RSVP*
- *consistent with the robustness of the connectionless model*
- *uses soft state which is refreshed periodically*
- *designed to support multicast*
- *is receiver oriented*
- *. . .*

Intserv: Reservation Protocol

- *RSVP uses two messages: PATH and RESV*
- *the source transmits PATH messages every 30 seconds with Tspec & route information*
- *the destination responds with RESV message with Rspec information*
- *an error is returned to the receiver if any router on the path cannot make the reservation*
- *the requirements are merged in the case of multicast.*

Intserv: Mechanisms



The Rspecs of the receivers A & B are merged at the upstream routers.

The Tspecs of the senders 1 & 2 are merged at the downstream routers.

Intserv: Packet classifying & scheduling

- *packet classifying: associate each packet with the appropriate reservation so that it can be handled correctly*
- *packet scheduling: manage the packets in the queues so that they receive the service that has been requested.*
- *scalability issues*
  - *IPv4 examines 5 items in the IP header: srd/dest IP address, src/dest port, protocol number*
  - *IPV6 examines the flow label.*
  - *soft state must be kept at each router for each flow*
  - *each router must queue, classify & police packets from every flow.*

Diffserv: Introduction

Intserv allocates resources to individual flows.

Diffserv introduces traffic aggregation: resources are allocated to a small number of traffic classes.

The best effort service model is enhanced with a premium class

- *an edge router at the boundary of an administrative domain sets the premium bit in the packet header*
- *the interior routers apply Per-Hop-Behaviour (PHB) rules to give preferential service to premium packets.*

The Diffserv Code Point (DSCP) is defined by 6 bits in the TOS field of the IP header.

Core routers need only examine the DSCP to decide on the packet service.

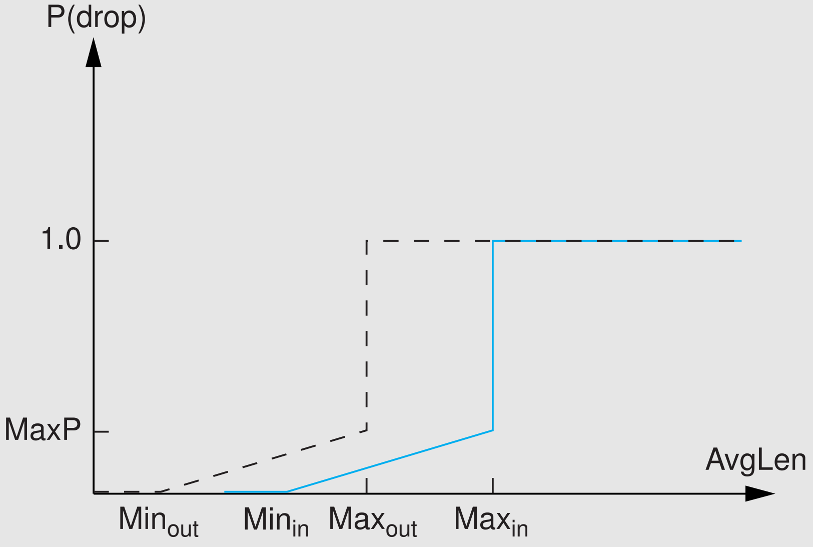
Diffserv: PHB

Expedited forwarding (EF) provides a virtual leased line service. The packet is forwarded with minimal delay & loss

- *per flow policing is applied at the edge of the network: EF packets that exceed their negotiated rate are dropped at the ingress router*
- *routers implement EF processing with a PQ or a WFQ.*

Diffserv: PHB

Assured forwarding (AF) uses Random Early Detection (RED) with in & out of profile (RIO)



- *a customer can send up to x Mbps of AF traffic*
- *the edge router marks packets: if $y \leq x$ then the packet is in profile else the packet is out of profile*
- *core routers provide in profile packets with a high assurance (not a guarantee) that they will be delivered.*

Diffserv: Weighted RED

How are the profile & the RIO parameters set correctly?

WRED has more than two profiles. The value in the DSCP field of the IP header is used to select a queue in a WFQ packet scheduler. For example, the weights W_{P}, W_{BE} compute a bandwidth

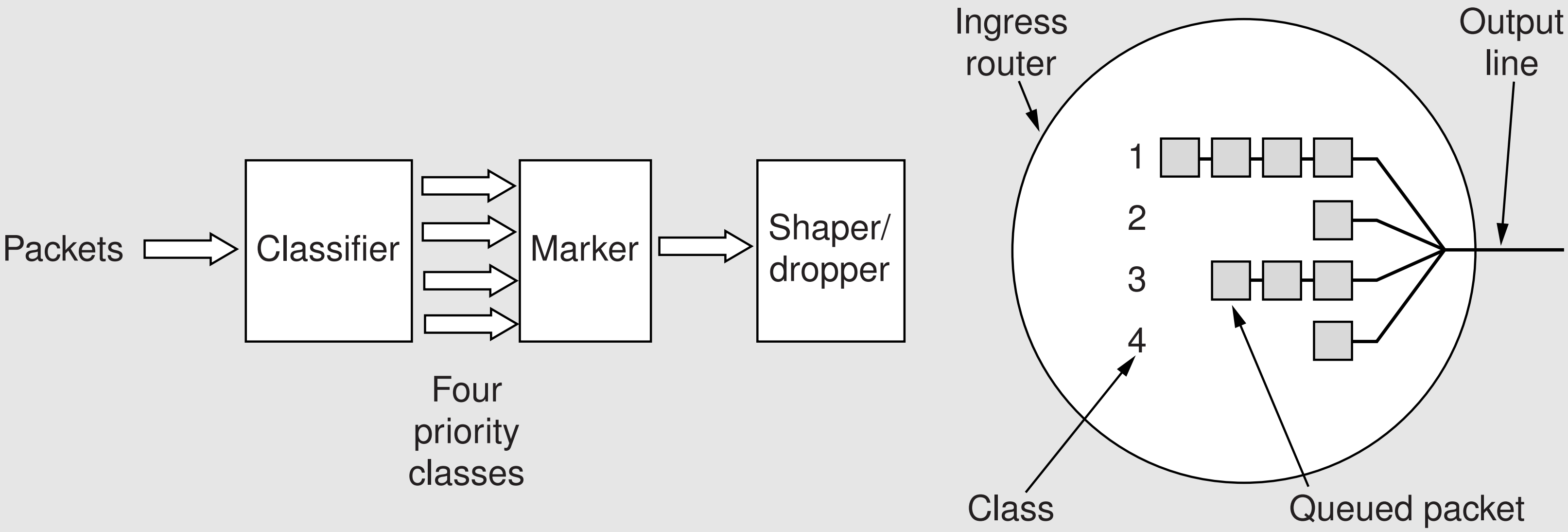
$$B_{\mathrm{P}} = W_{\mathrm{P}}/(W_{\mathrm{P}} + W_{\mathrm{BE}})$$

for premium-class packets.

Assured forwarding defines 12 DSCP values to select from 4 queues in a WFQ packet scheduler with 3 drop preferences.

Intserv & Difserv should not be seen as competing, but as complementary technolgies. Proposals have been made to use Intserv in the access network & Difserv in the core network.

Diffserv: Weighted RED



QoS: ATM

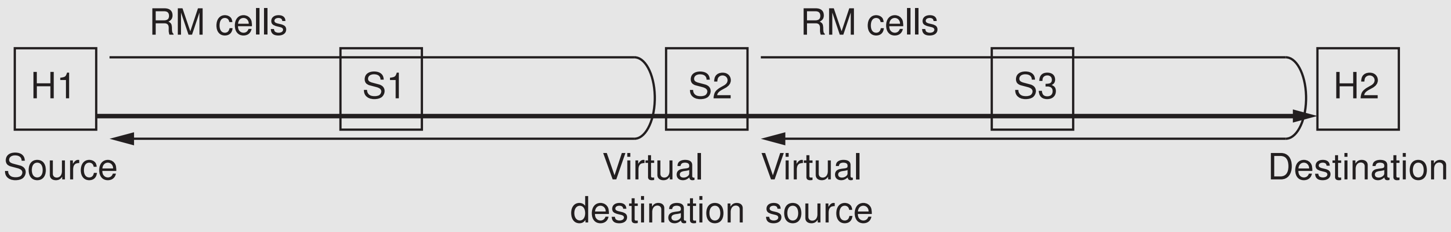
ATM provides 5 service classes

- *constant bit rate CBR*
- *variable bit rate-real-time VBR-rt* $\sim$ *Intserv guaranteed service class*
- *variable bit rate-non real-time VBR-nrt* $\sim$ *Intserv controlled load service*
- *available bit rate ABR*
- *unspecified bit rate UBR* $\sim$ *best effort.*

ABR implements congestion control mechanisms.

QoS: ATM ABR

ABR congestion control is implemented by the source & destination of a VPC which exchange resource management (RM) cells to implement explicit congestion notification.



QoS: RSVP versus ATM (Q.2931)

- *RSVP*
  - *receiver generates reservation*
  - *soft state (refresh/timeout)*
  - *separate from route establishment*
  - *QoS can change dynamically*
  - *receiver heterogeneity*
- *ATM*
  - *sender generates connection request*
  - *hard state (explicit delete)*
  - *concurrent with route establishment*
  - *QoS is static for life of connection*
  - *uniform QoS to all receivers*