



# QoS

## Realtime, Isochronous Traffic, and Heavy Load

(C) Herbert Haas 2005/03/11



*“Well, my terminal's locked up, and I ain't got any Mail,  
And I can't recall the last time that my program didn't fail;  
I've got the stacks in my structs, I've got arrays in my queues,  
I've got the : Segmentation violation – Core dumped blues”*



**From the song "Core dumped blues"**

# Packet Switching Problems



- **Internet is inhomogeneous**
  - ◆ **QoS is given by the weakest link in the chain between sender and receiver**
  - ◆ **Over-provisioning the network cannot be economically justified**
- **QoS mechanisms manage the available bandwidth, queuing, delays**

Synchronous networks (also called "isochronous" networks) have no problem with quality of service: Each piece of data is transmitted in equal times and there are no "**bursts**" of data that would lead to sudden congestion. Real synchronous clocking of all network components means no need of any buffers.

The Internet on the other hand is an extremely inhomogeneous and asynchronous kind of network. It is a network of networks and the prevalent type of traffic is data traffic.

Data traffic is bursty by nature. Quality of service is determined by the weakest link between each two points of communications.

What is meant by "Quality of Service" (QoS) actually? The total network bandwidth is shared by many users and applications. We demand for QoS in that some specific type of traffic should be handled "nicer" than others. When buffers in routers and switches are congested then delays and packet-drops occur. One solution is to over-provision the network, but this is expensive and the behavior is still unpredictable.

Thus QoS mechanisms are thought to **manage** the network resources in terms of bandwidth and queuing behavior, so delays can be limited.

# The Deeper Insights



- QoS is basically a queuing issue
- This queuing issue arises as we do statistical multiplexing of many users/flows over a trunk/network which is under-dimensioned
  - ◆ This is cheaper!
- Observed phenomenons:
  - ◆ Blocking
  - ◆ Overflowed queues

# Blocking



- **First analyzed by A. K. Erlang**
- **Widely known is the "Erlang B" traffic model**

$$E(\rho, C) = \frac{\rho^C}{C!} \left( \sum_{n=0}^C \frac{\rho^n}{n!} \right)^{-1}$$

- **With the advent of data communication, it turned out that this is only valid for Poisson-traffic**
  - ♦ **Where one average session duration is possible**
  - ♦ **But data traffic is fractal !!!**

E is blocking probability,  $\rho$ =access rate/service rate, C is number of channels for this trunk.

Agner Krarup Erlang was born on 1st January, 1878 in Lonborg in Denmark. He studied mathematics and natural science at Copenhagen, where he graduated in 1901 with mathematics as his major subject and physics, astronomy and chemistry as secondary subjects, he taught in schools for several years.

His interests turned towards the theory of probability and he kept up his mathematical interests by joining the Mathematical Association. At meetings of the Mathematical Association he met Jensen who was then the chief engineer at the Copenhagen Telephone Company. He persuaded Erlang to apply his skills to the solution of problems which arose from a study of waiting times for telephone calls.

In 1908 Erlang joined the Copenhagen Telephone Company and began applying probability to various problems arising in the context of telephone calls. He published his first paper on these problems The theory of probability and telephone conversations in 1909. In 1917 he gave a formula for loss and waiting time which was soon used by telephone companies in many countries including the British Post Office.

Erlang died on 3rd, February, 1929

# Self-similar Traffic



- Erlang's formula does not work for data-traffic
- "Bursty" traffic
- Data-traffic is self-similar
  - ◆ Same statistical patterns on all time scales
  - ◆ Analysis through wavelet-spectrum and stochastic processes
- Sophisticated queuing strategies become necessary

# Two Main Reasons for QoS



- **Different traffic classes**
  - ◆ **Realtime Traffic**
    - Jitter sensitive (VoIP)
  - ◆ **High Volume Multicast**
  - ◆ **FTP vs Telnet**
- **Bottlenecks**
  - ◆ **Different physical transmission rates**
  - ◆ **Queue overflow**
  - ◆ **Queues at bottlenecks stay filled**

**Realtime** traffic needs to be **continuous** in order to provide acceptable quality. FTP, HTTP or other **non realtime** traffic do not suffer from delays, i. e. the service is robust.

Note that realtime traffic does not need to be "fast" traffic; but the delay of delivery of data and/or the response must be **bounded**.

# The Idea: Services for Classes



- **Service = pre-defined packet treatment**
  - delay
  - jitter
  - packet loss probability
  - throughput
  - MTU
  - priorities
- **Class = Set of "similar" packets**
  - ♦ **Similar = according traffic filter matching rules**
    - IP header fields
    - Input/Output Interfaces

**Class:** A traffic class can be defined in many ways using specific traffic characteristics or protocol properties.

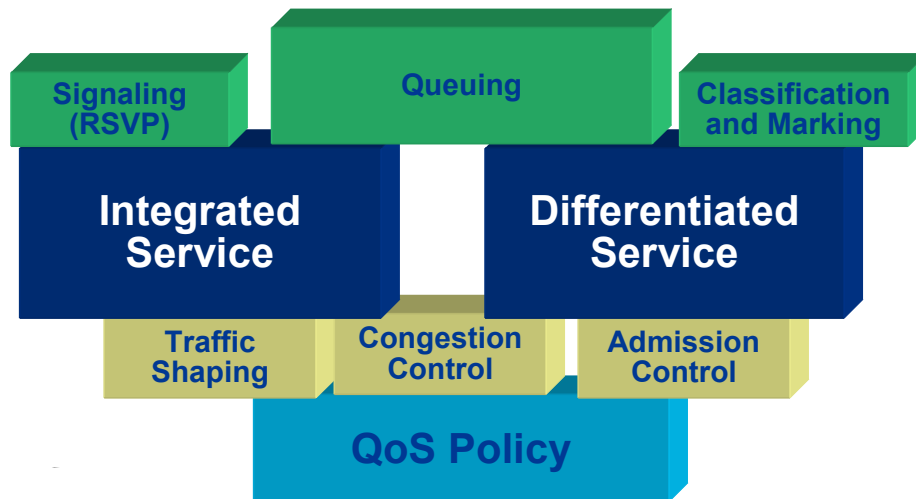
For example:

Class A: RTP  
Class B: Telnet  
Class C: all other traffic

**Service:** Once traffic classes are defined you can assign a packet treatment (service) to each class

Class A: one way delay < 150ms  
jitter < 50ms  
priority High  
Class B: delay < 500ms  
priority Medium  
Class C: priority Low

# Building Blocks



(C) Herbert Haas 2005/03/11

9

The diagram above shows the main building blocks of a QoS design. Let's discuss it from the bottom to the top:

The **QoS Policy** constitutes a fundament for the whole QoS concept. Here all desires and rules are specified.

The next layer represents important tools which are almost always present in a QoS architecture: **Traffic Shaping** (smooth bursts), **Congestion Control** (avoid bottlenecks and denial of service), **Admission Control** (assign QoS to dedicated users).

The next layer represents two totally different QoS architecture principles: **Integrated Service** ("**IntServ**") and **Differentiated Services** ("**DiffServ**"). IntServ is a connection-oriented approach but seems not to be scalable for large networks. DiffServ is connectionless and more scalable but QoS is difficult to control.

Finally, on the top layer, the diagram presents the instances that are used to implement QoS: IntServ uses a signaling protocol called **RSVP**, and DiffServ uses a **Classification** and **Marking** engine in the packet switches (routers).

Both technologies adjust the queuing parameters. **Queuing** is the most fundamental part of QoS and is described later.

## QoS Components (1)



- **Signaling**
  - ♦ E. g. resource **reservation**
- **Queuing**
  - ♦ Prioritize some packets over others and try to remain **fair**
- **Classification**
  - ♦ Detection of specific traffic classes or traffic **flows**
- **Marking**
  - ♦ Assigned a **label** to classified packets to request a special service

**Signaling** in the context of QoS means that particular (if not all) devices that establish a connection are controlled by a special protocol, typically the Resource Reservation Protocol (RSVP).

**Queuing** means, that the buffers of a packet switch (routers) are not only meant to mitigate bursts but also to prioritize some packets over others. The most important goal in queuing is fairness.

**Classification** is needed by routers in order to recognize different types of flows that must be assigned to different classes of service.

**Marking** is typically performed after classification and assigns some kind of label to all packets of a flow in order to request a special treatment (service) from all routers following.

## QoS Components (2)



- **Integrated and Differentiated Service**
  - ♦ Two fundamental **QoS philosophies**
- **Traffic shaping**
  - ♦ Smoothens bursty traffic by **introducing delay**
- **Congestion control**
  - ♦ Reduces packet rate when network congestion occurs
- **Admission control**
  - ♦ Provides QoS features only to dedicated users
- **QoS policy**
  - ♦ Fundamental QoS agreements specifying detail how to handle traffic, traffic classes, signaling, etc.

The **Integrated** and **Differentiated** Services architectures are the two main QoS philosophies known today. They are described later.

**Traffic Shaping** smoothens traffic by buffering bursts within a dedicated time period and assures sending data continuously. Obviously this approach adds additional delay!

**Congestion Control** reduces the emitted traffic as soon as congestion takes place. An example is the usage of FECN and BECN bits in Frame Relay networks.

**Admission Control** is necessary if the network is used by different classes of users. Typically, users must be authenticated and authorized by a dedicated server before QoS is available for them.

A **QoS Policy** is specified in a contract between provider and customer. Here all QoS details for several traffic classes are given in terms of parameters.

## Why Queues ?



- To absorb the burstiness that occurs in **statistically multiplexed** networks
- Necessary because of *short-term mismatches* between packet arrival and departure rates
  - ♦ "Short-term": up to a few transcontinental RTTs, e.g. 100 ms

If the amount of sendable traffic exceeds the available bandwidth, traffic must be queued.

Queuing should only take place for very short periods and to handle short term bursts. If queuing is present for longer time periods, traffic will be discarded.

## ! Practical Statement !



- **Queues should be empty on average**
- **A queue depth  $> 0$  typically leads to congestion very soon !**
- **Queues should just smooth bursts**

It is a common misunderstanding to assume that queues are always filled at a certain percentage. If this would be the case then traffic bursts would occur too frequent and the probability of congestion is very high.

Note that queues should only smooth bursts! On average the traffic rate should be easily handled by the network devices (packet switches in general) without filling the queues considerably.

# Growing Queue Depths



- **Short-term variations in arrival rates**
  - ◆ Irrelevant; smoothed by queues
- **Long-term input rate mismatch**
  - ◆ Excess packets are discarded
  - ◆ Drop rate proportional to excess queue
- **Long-term window-delay mismatch**
  - ◆ Most important case!
  - ◆ Requires completely different solution than the second case!

**Short-term variations** in arrival rates are smoothed by the queues and therefore irrelevant. We discussed this already.

**Long-term mismatch** of the traffic rate and the allowable input rate lead to congestion of the queues and all packets beyond that limit are dropped. This behavior is known as tail dropping, because the first packets entering the queue are forwarded correctly. Of course the service degrades if the traffic rate increases. Service degradation can only be mitigated if some sort of intelligent packet prioritization is applied, e. g. either DiffServ or IntServ.

If a long-term mismatch of the **window size** and the **delay** occurs then the run really into deep trouble and other sophisticated methods are necessary. To be more specific, in this situation the window-size of TCP does not match the product of bandwidth and delay, but practically only the delay can be measured by TCP. Here the delay varies between two extremes and TCP opens and closes its window size alternately. **The state of the network toggles from congestion to silence and back.** One solution is to configure **RED** on the routers (explained later).

## Queuing Methods



- **FIFO**
- **Priority Queuing**
- **Class-based Queuing**
- **Fair Queuing**
- **Weighted Fair Queuing**
- **Class-based Weighted Fair Queuing**

Today, dozens of queuing methods are used in various platforms. This chapter only presents the most important. All other methods can be easily deduced from them (however, some mathematics might be necessary but fortunately the basic principles of the methods listed above can be easily understand without maths).

We start our explanation with the simplest method and we will continuously add some new features that lead us to the next method.

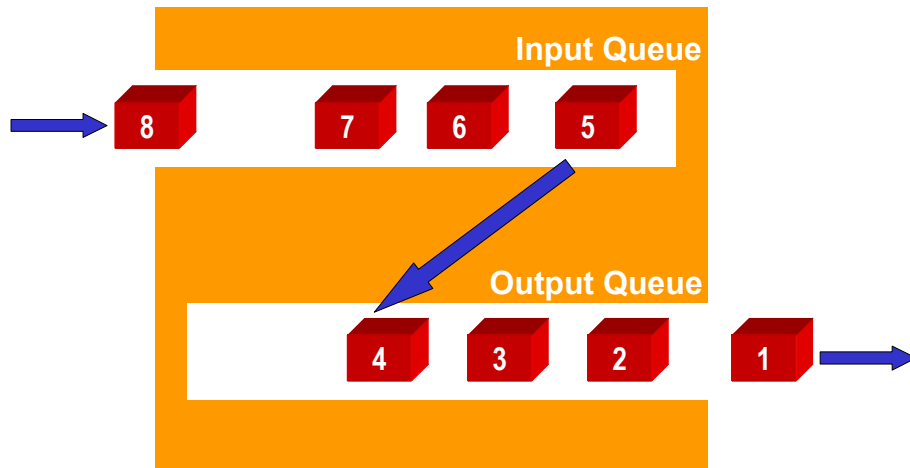
## FIFO Queuing (1)



- **Fastest, simplest, and most common solution**
- **Only **one buffer** for all type of packets available**
- **In case of congestion: No predictable behaviour!**
  - ◆ Usually, complete service degeneration
  - ◆ Delays become longer
  - ◆ Packet drop

First-in first-out (FIFO) queuing is the most trivial method and is realized by a simple buffer.

## FIFO Queuing (2)



The picture above shows the basic principle of FIFO queuing.

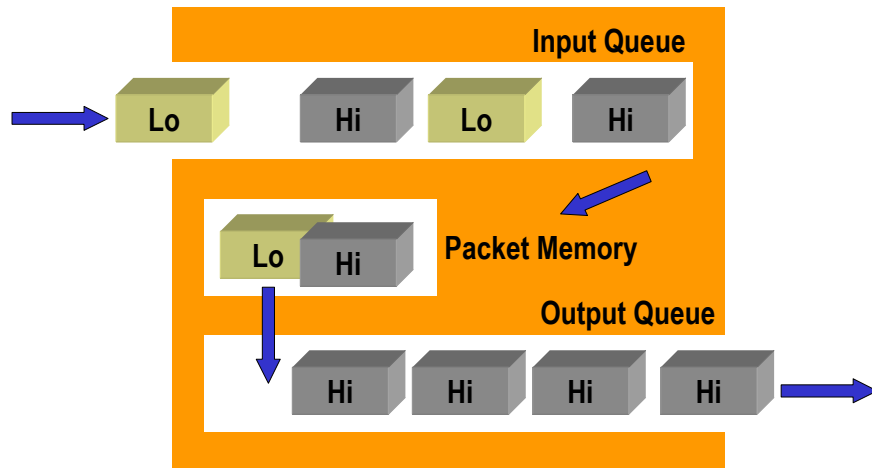
## Priority Queuing (1)



- **Different levels of priority**
  - ♦ For certain protocols and services
  - ♦ E.g. SNA before IP, UDP before TCP, Telnet before FTP
  - ♦ Requires additional (output) queues

By employing multiple FIFO queues all packets could be separated with respect to their priority. The classification only separates packets not flows.

## Priority Queuing (2)



(C) Herbert Haas 2005/03/11

19

As long as traffic exists in a higher priority queue all lower priority queues will not be serviced. This could result in starving queues if high priority traffic is always present.

## Problems with Priority Queuing



- **Packet identification is a time consuming process**
- **Too much priority traffic outperforms low priority traffic → starvation**
  - ◆ **Big delay**
  - ◆ **Low priority packets are dropped**

Assume that there is a lot of SNA traffic which is always considered as high priority compared to IP traffic. In this case SNA packets are continuously forwarded and IP traffic will be dropped.

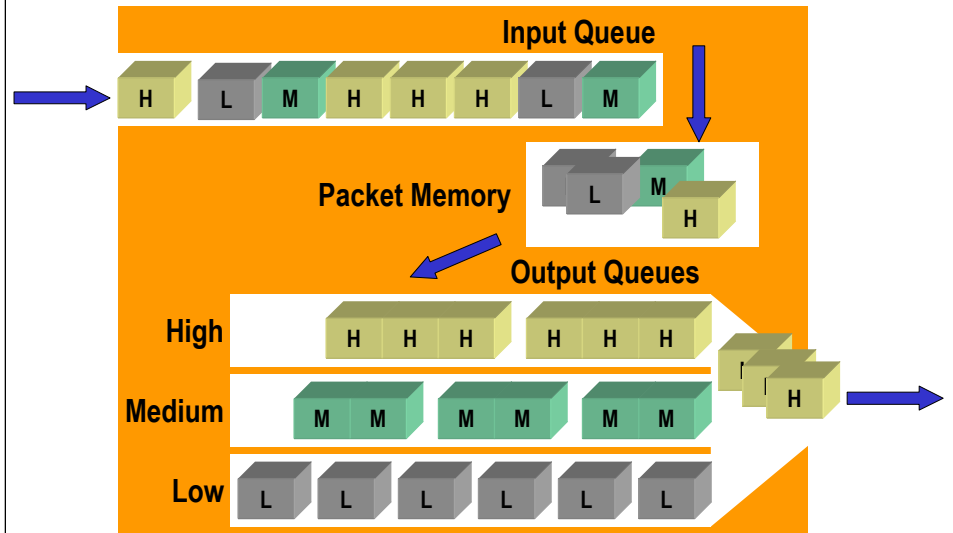
## Class-Based Queuing (1)



- Is simply a **round-robin** processed priority queuing method
- Each turn, depending on the priority of the queue, a predefined amount of data is sent from the queue
  - ♦ High priority packets are sent more often
  - ♦ But also low priority packets are served

In order to avoid starvation of low-priority traffic the class-based queuing method (also known as custom queuing) simply processes all priority queues in a round-robin manner.

## Class-Based Queuing (2)



(C) Herbert Haas 2005/03/11

22

The picture above shows the basic principle of Class-Based Queuing (CBQ).

## CBQ Still Not Fair !



- **Protocols and services are separated into service classes but **session** information is ignored !!!**
  - ◆ **For example, a misbehaving TCP session could push away other sessions within the same priority level if it produces a huge volume of traffic**
- **Round-robin over flows needed !**

But CBQ still processes packet by packet and therefore it can't be fair. Assume that a single misbehaving application or user produces too much traffic compared to other applications or users. That is, on the same priority level not all users (or applications) receive the same amount of service.

Only a round robin over flows would be really fair!

## Fair Queuing (1)



- **Separates incoming traffic into "flows"**
- **Flow = Packets belonging to the same session (socket information)**
- **Guarantees each flow an equal share of the transmission capacity**

Fair queuing is the most sophisticated and resource-demanding approach. Here all traffic flows are separated in single queues which are processed round robin.

A flow is determined by a unique socket information (source and destination address and port number).

## Fair Queuing (2)



- **Ideal goal**
  - ◆ Determine the number of active flows
  - ◆ Store packets of every conversation in separate queues
  - ◆ Serve queues round-robin **bit-by-bit**
- **Reality**
  - ◆ Packet cannot be transmitted one bit-by-bit
  - ◆ But can be approximated

Ideally all flows would be served bit by bit but this is not possible since whole packets must be sent. Hence, the multiplexer must calculate a backlog for each queue: Once a big packet has been sent then other queues may send multiple small packets.

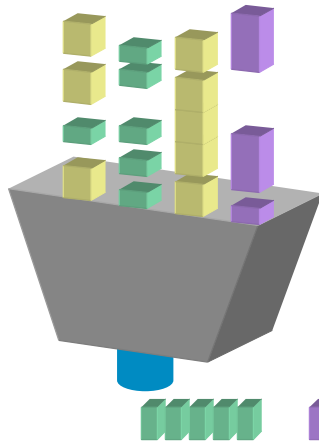
## Weighted Fair Queuing



- **Cisco's Modification**
- **Weighted fair queuing gives certain flows a larger portion of the link capacity**
  - ◆ Certain flows have more “weight”
  - ◆ Time division multiplexing with unequal time slots
- **Weighting based on**
  - ◆ IP precedence bits in the TOS (Type of Service) field in the IP packet header
  - ◆ A signalling procedure (RSVP)

On Cisco routers weighted fair queuing is the default queuing method on interfaces up to 1.5mb. (it requires no configuration effort)

# Leaky Bucket Principle



Every clock-tick the leaky bucket tries to put a fixed number of octets onto the network

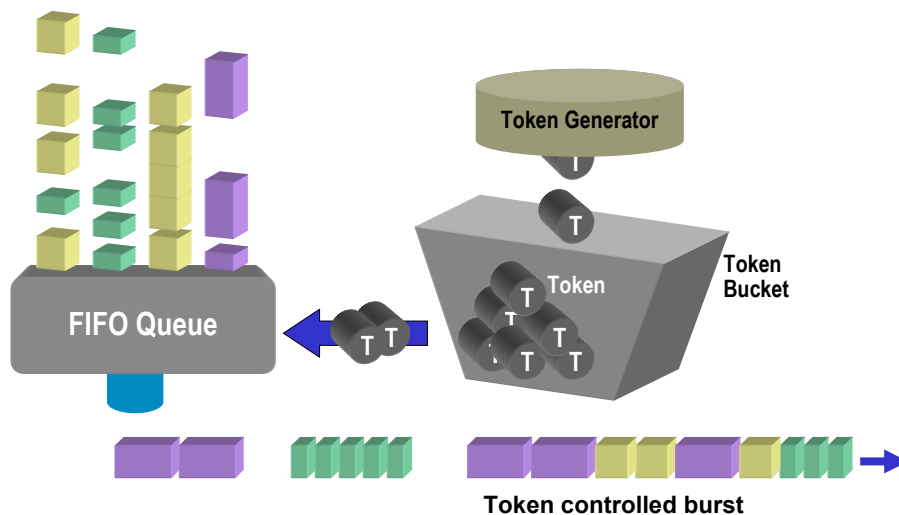
Note: No reordering possible!



This picture illustrates the leaky bucket principle which simply smoothens bursts and assures a continuous traffic forwarding at a fixed sending rate.

This principle corresponds basically to a FIFO buffer.

## Token Bucket Principle



(C) Herbert Haas 2005/03/11

28

The token bucket principle is often used to specify a traffic characteristic in great detail.

A token generator produces so-called "tokens" with a constant frequency. These tokens are collected by a token bucket while traffic packets are collected in a FIFO queue similar to a leaky bucket. But other than the leaky bucket this FIFO queue has a **valve** attached on its output. And this valve is opened by removing tokens from the token bucket. The more tokens are consumed by the valve the more open is it and the more packets can be transmitted.

This way traffic can be specified very precise: The minimum sending rate is given by the token generation frequency. The size of the token bucket determines the maximum burst size that can be transmitted. The size of the FIFO queue determines the time constant between bursts.

## Low Latency Queuing (PQ-CBWFQ)



- **Cisco Implementation preference for Voice over IP**
- **Is actually Priority Queuing – Class Based Weighted Fair Queuing (PQ-CBWFQ)**

Low Latency Queuing paradigm:

The configured classes are policed to Bandwidth to make sure that other traffic is not starved

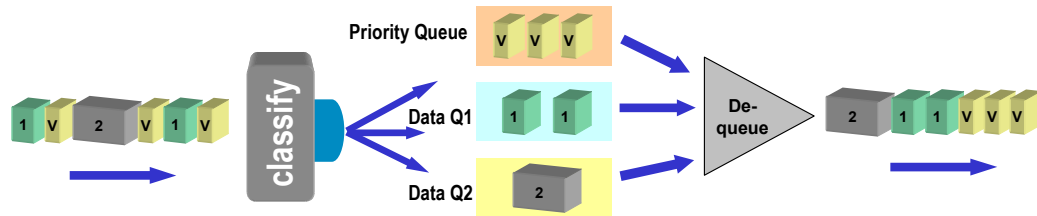
VoIP support over many different Layer 2 technologies

No oversubscription of Bandwidth

If multiple classes send traffic to priority-queue still the rate limit is defined per class

In priority classes RED is not supported

# Low Latency Queuing (PQ-CBWFQ)



The first step is to classify traffic based on:

1. Source / Destination address
2. Session Identifier (TCP Port)
3. Protocol Type
4. Access Lists

The de-queuing process is sending all packets in the priority queue first and then using a weighted fair algorithm for the data queues.

# TCP Performance



- **Not used for voice and other realtime traffic**
  - ♦ Why should we retransmit damaged/lost packets?
- **But good self-regulating mechanism to avoid congestion**
- **TCP is "hungry but fair"**
  - ♦ But only fair to other TCP applications !
  - ♦ Problem together with voice and other realtime traffic

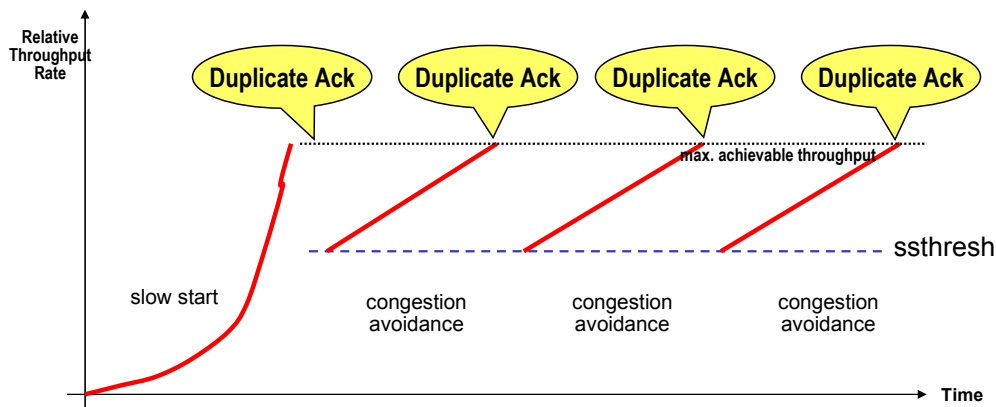
TCP has been designed for data traffic only. Error recovery does not make sense for voice and video streams. TCP checks the current maximum bandwidth and tries to utilize all of it. In case of congestion situations TCP will reduce the sending rate dramatically and explores again the network's capabilities. Because of this behavior TCP is called "hungry but fair".

The problem with this behavior is the consequence for all other types of traffic: TCP might grasp all it can get and nothing is left for the rest.

# TCP is Hungry !



- Tries to fill the "pipe" using **Slow Start** and **Congestion Avoidance**



(C) Herbert Haas 2005/03/11

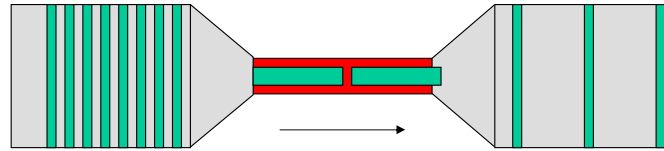
32

The diagram above shows the typical TCP behavior of one flow. There are two important algorithms involved with TCP congestion control: "**Slow Start**" increases the sending rate exponentially beginning with a very low sending rate (typically 1-2 segments per RTT). When the limit of the network is reached, that is, when duplicate acknowledgement occur, then "**Congestion Avoidance**" reduces the sending rate by 50 percent and then it is increased only linearly.

The rule is: On receiving a duplicate ACK, congestion avoidance is performed. On receiving no ACK at all, slow start is performed again, beginning at zero sending rate.

Note that this is only a quick and rough explanation of the two algorithms—the details are a bit more complicated. Furthermore, different TCP implementations utilize these algorithm differently.

# TCP Window Size too Large



- **TCP's Ack-clocking!**
  - ◆ Every node runs exactly at the bottleneck link rate
  - ◆ Queue size has nothing to do with rate mismatch
  - ◆ Queue size is proportional to TCP window size and  $1/(\text{delay} \cdot \text{bandwidth})$

Using TCP the depths of the queues are controlled by the ACK frequency, therefore TCP is called to be **ACK-clocked**. Only when an ACK is received the next segment is sent. Therefore TCP is self-regulating and the queue-depth is determined by the bottleneck: Every node runs exactly at the bottleneck link rate. If a higher rate would be used, then ACKs stay out and TCP would throttle its sending rate.

# RED



- **Known as "difficult to tune"**
- **Self-tuning RED**
- **Running estimate weighted moving average (EWMA) of the average queue size**
- **Inspiration principle: The WC tank!**
  - ◆ **Regulation mechanism guarantees a full tank (Alexander Cummings, 1775)**

Many TCP streams in a network tend to synchronize each other in terms of intensity. That is, all TCP users recognize congestion simultaneously and would restart the slow-start process (sending at a very low rate). At this moment the network is not utilized. After a short time, all users would reach the maximum sending rate and network congestion occurs. At this time all buffers are full. Again all TCP users will stop and nearly stop sending again. This cycle continues infinitely and is called the TCP wave effect. The main disadvantage is the relatively low utilization of the network.

Random Early Discard (RED) is a method to de-synchronize the TCP streams by simply drop packets of a queue randomly. RED starts when a given queue depth is reached and is applied more aggressively when the queue depth increases.

RED causes the TCP receivers to send duplicate ACKs which in turn causes the TCP senders to perform congestion avoidance. The trick is that this happens randomly, so not all TCP applications are affected equally at the same time.

Although the principle of RED is fairly simple it is known to be difficult to tune. A lot of research has been done to find out optimal rules for RED tuning.

## Integrated Services Architecture



- Idea: Client reserves network resources on every router "**upstream**" to the server
- Reservation for each **flow (scalability problem)**
- **RSVP** as signaling protocol

Resource ReSerVation Protocol (RSVP), RFC 2205

Using RSVP a client requests specific QoS parameters at each router along the upstream path. If possible, routers reserve resources for this flow. Each flow uses a single dedicated path.



- **Three Service Types**
  - ♦ **Best Effort**
  - ♦ **Controlled Load**
    - Like Best-Effort without congestion
  - ♦ **Guaranteed**
    - Simulates leased line

IntServ specifies three fundamental service types. Best effort is without QoS.  
Controlled Load simulates an un-congested network.  
Guaranteed QoS simulates a leased line.

# IntServ – Traffic Specification



- **From sender to network and receivers**
  - ◆ **SENDER\_TSPEC**
    - Declares sender's traffic
    - Never modified by intermediate nodes
- **From sender or intermediate nodes**
  - ◆ **ADSPEC**
    - Describes properties of path
    - Availability of QoS service
- **From receiver to intermediate nodes and senders**
  - ◆ **FLOWSPEC**
    - Receiver\_TSpec and Receiver\_RSPEC

# RSVP Basics

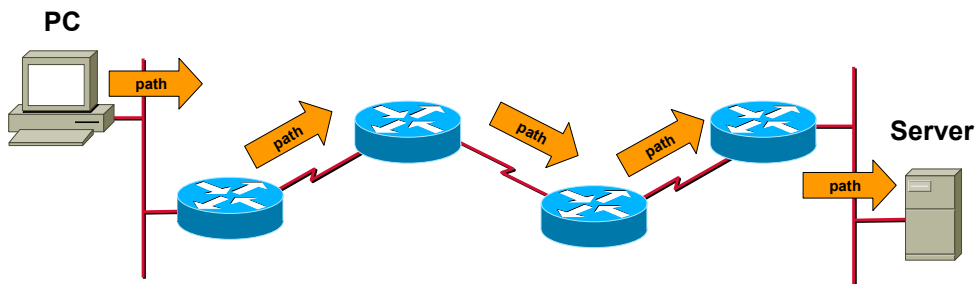


- Used to reserve bandwidth along a "flow-path" between "*source*" and "*destination*"
  - ◆ *Destination* actually reserves bandwidth
- Bandwidth reservation done by establishing a dedicated WFQ with higher a priority than IP precedence could ever reach
- Queue reservation will be cleared after flow is cleared

# RSVP Function I



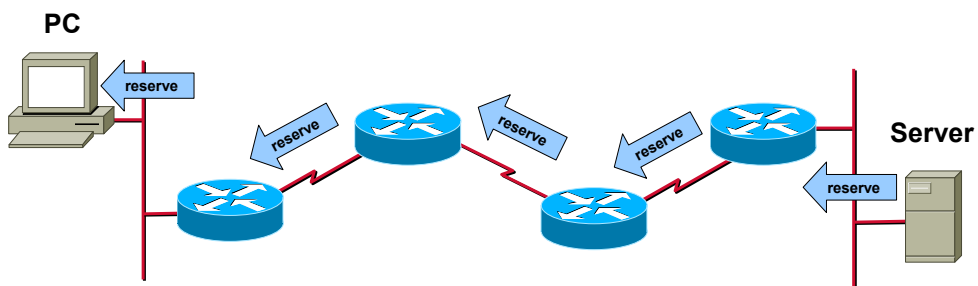
- **Reservation request**
  - ◆ PC requests bandwidth (path message)
  - ◆ Router along path forward "request" to destination



# RSVP Function II



- **Reservation request**
  - ♦ Server accepts reservation-request and sends reservation packet back along path
  - ♦ Each router builds a WFQ for this flow



# RSVP



- **Source**
  - ♦ Traffic Profile Definition
- **Destination**
  - ♦ Reservation Profile Definition
- **Soft states**
  - ♦ Need to be refreshed
- **Per flow reservation**
- **Each intermediate node must support RSVP!**
  - ♦ Does not scale well
  - ♦ Impossible: Signalling through the whole Internet

The problem with RSVP is that each intermediate node must support it—otherwise IntServ does not work well.

Obviously IntServ cannot work throughout the Internet because the total signaling overhead and the number of states would be too high.

## Differentiated Services Framework



- **Idea: Packets are separated into traffic classes – routers treat them accordingly**
- **Requires a traffic identification and marking (labeling) mechanism**
  - ♦ IP Type of Service (ToS) field
  - ♦ Aka "Differentiated Service Code Point (DSCP)"
  - ♦ No reservations necessary → Scalable
  - ♦ Per-hop oriented behavior
- **Static QoS mechanism**

DiffServ is very scalable because it is state-less and does not require any kind of signaling. The basic idea is the "**per-hop behavior**". Each node (router) examines a **label** which is attached to each packet and determines the service it should receive.

Using IPv4 the ToS field is reused as label fields (actually only 6 bits). In IPv6 a similar field (flow discriminator) is used.

Several issues are not completely solved yet, for example the consolidation of labels between different networks and the dynamic configuration of label maps. Furthermore routing protocols should calculate different paths for each label.

# DiffServ – Scalability



- **No signalling!**
- **Goal**
  - ◆ **Flows are aggregated in the network**
  - ◆ **Core routers only need to distinguish a small number of aggregated flows**
  - ◆ **Even if those flows contain thousands or millions of individual flows**