Quality-of-Service in IP Networks

Roch Guérin
University of Pennsylvania

IEEE RTAS'2000
Washington, D.C.
May 30, 2000

Outline

- Introduction
  - Definition of Quality-of-Service
    - Goals and requirements
- Freshening up on basic building blocks
  - Traffic contracts
  - Scheduling and buffer management
  - Call admission
- The Internet approaches
  - A first step
    - Differentiated Services
  - Going all the way (maybe)
    - Integrated Services and RSVP
- Summary and references

Delivering Different Levels of Service

- Focus so far has been on mechanisms related to providing *connectivity*, i.e., delivering data from source to destination
  - Routing protocols establish forwarding state
  - Forwarding mechanism determines where to send packets next
- But what if I want more than just bare connectivity
  - Guarantees regarding
    - Available bandwidth (minimum or sustained), bounds on delay or delay variations, losses, etc.
    - Quantitative (worst case or statistical) or qualitative (better than) performance measures

  $\Rightarrow$ Service classes provide different combinations of service guarantees above and beyond connectivity

Service Differentiation

- Two major components to service differentiation
  - Data path identifies packets eligible for service guarantees and enforces them
  - Control path determines if and how guarantees can be provided
- Data path
  - Packet classifiers
    - which packet is entitled to what
  - Scheduling
    - controls access to transmission opportunities
  - Buffer management
    - controls access to storage opportunities
- Control path (call admission)
  - Based on
    - traffic characteristics
    - type of service guarantees
    - current network state (available resources)
  - Multiple time scales possible
    - from provisioning to on-demand (signalling)

Freshening Up

Building Blocks

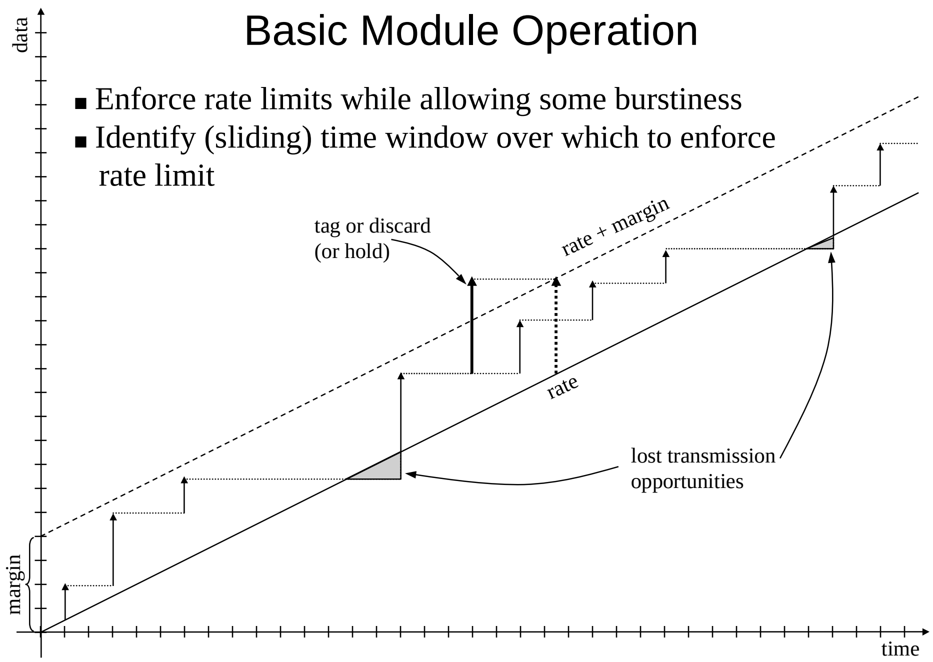
- Traffic characteristics
  - How to describe the set of packets that are to receive a certain level of service?
- Scheduling
  - Which packet goes out next?
  - What guarantees?
  - Efficiency vs complexity
- Buffer management
  - Which packets to store?
  - What guarantees?
  - Efficiency vs complexity
  - Coupling to scheduling

Overview of Traffic Characteristics

- Purpose
  - Specify the traffic (set of packets) to which the service guarantees applies
- Requirements
  - Simplicity of expression
  - Ease of verification
  - Implementation complexity and scalability
- Generic method
  - Token bucket a.k.a. *leaky bucket*
  - Deterministic algorithm that *bounds* traffic
  - Controls rate and burst size

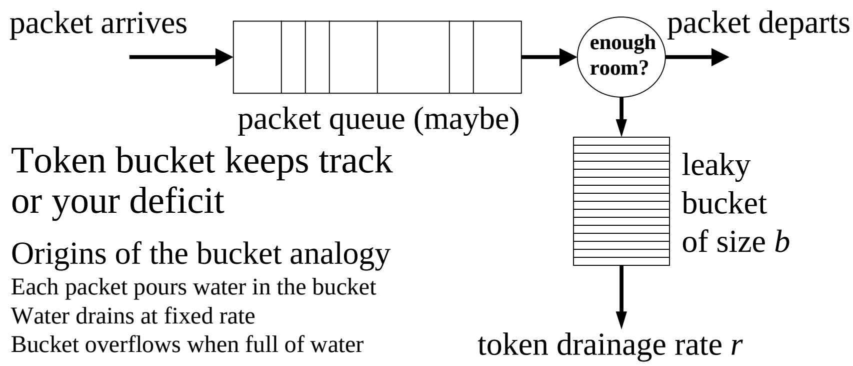
Basic Module Operation

- Enforce rate limits while allowing some burstiness
- Identify (sliding) time window over which to enforce rate limit



The *Pessimist* Token Bucket Definition

- Token bucket limits flow of packets *into* the network
  - Packets generate tokens (in proportion to their size) upon entering the network
  - Tokens drain at a contracted token rate (*r*)
  - Limit *(b)* on the deficit a user can run
  - In the presence of too high a deficit, packets are
    - Dropped, marked, or shaped



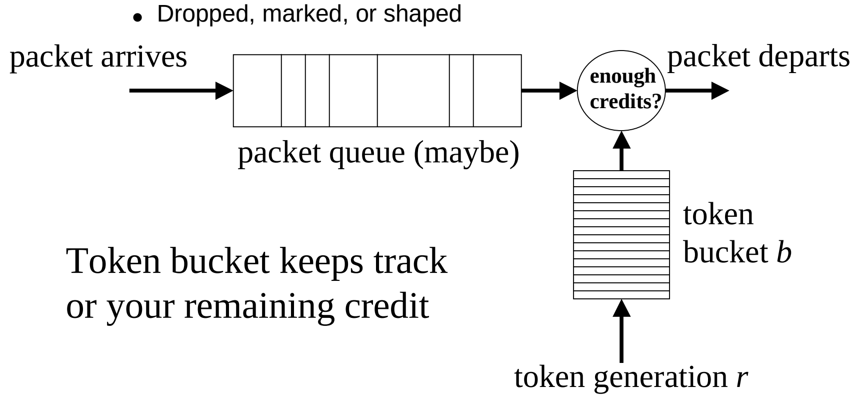
Token bucket keeps track or your deficit

Origins of the bucket analogy

Each packet pours water in the bucket
Water drains at fixed rate
Bucket overflows when full of water

The *Optimist* Token Bucket Definition

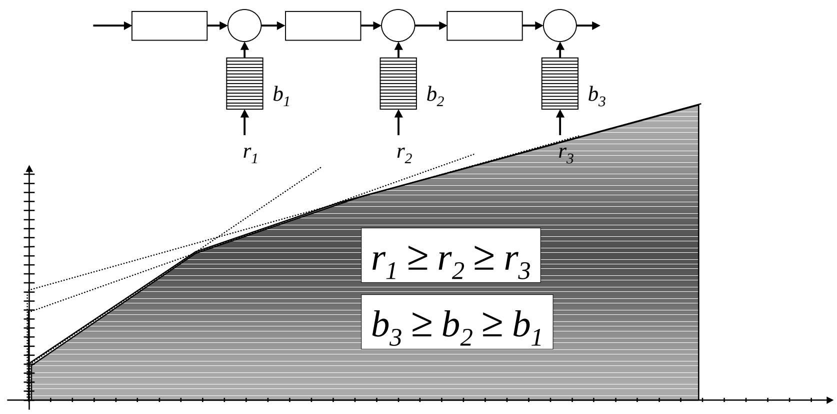
- Token bucket limits flow of packets *into* the network
  - Packets require credits to enter (in proportion to their size)
  - Credits/tokens accumulate at a contracted token rate (*r*)
  - Limit *(b)* on the number of credits that can be accumulated
  - In the presence of insufficient credits packets are
    - Dropped, marked, or shaped



Token bucket keeps track or your remaining credit

Concatenation of Buckets

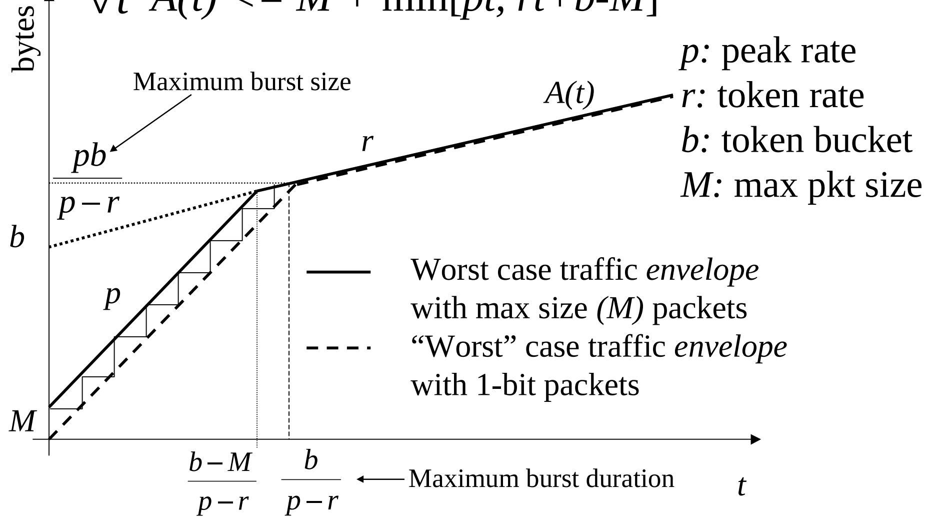
- Multiple buckets can be concatenated to control multiple rates
  - Rate is minimum across all buckets



The *Dual* Token Bucket

Token bucket limits *short* and *long* term rates

$$\forall t \;\; A(t) <= M + \min[pt, rt+b-M]$$



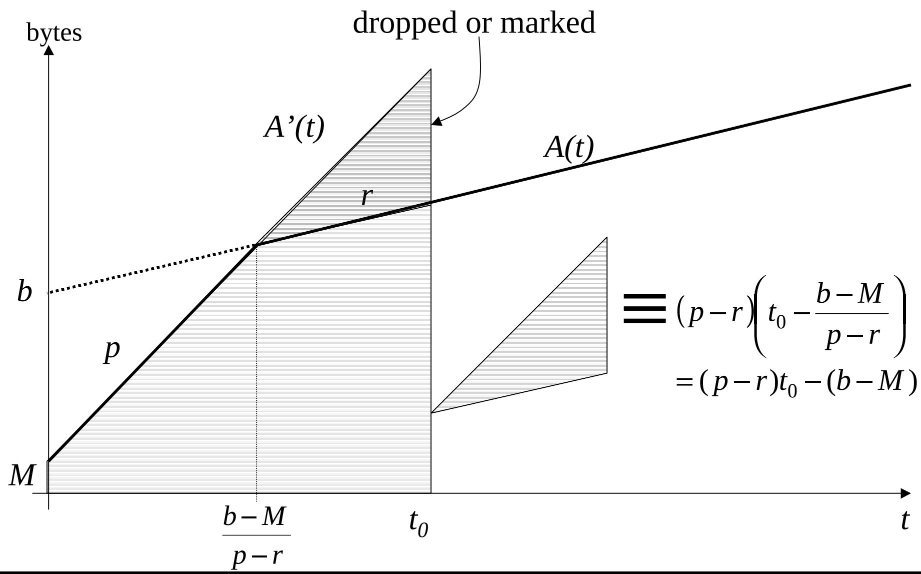
Maximum Burst Size

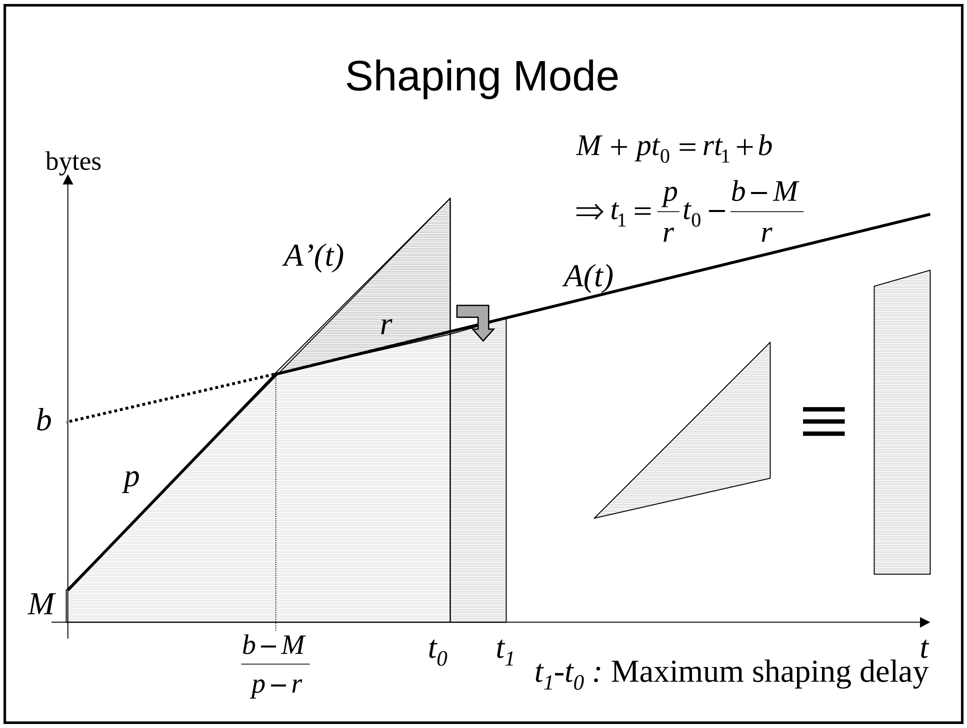
- Why is maximum burst size $\frac{b}{p-r}$?
  - Time to use up initial token pool: $t_0=b/p$
  - By time t_0, N_0 tokens have been accumulated: $N_0=r\,t_0=rb/p$
  - Those N_0 tokens are used up in time $t_1= N_0/p= rb/p^2$
  - After time t_1, N_1 tokens have been accumulated: $N_1=r\,t_1=r^2b/p^2$
  - And so on
  - Total time to run out of tokens is $T_b = \frac{b}{p}\sum_{i=0}^{\infty}\left(\frac{r}{p}\right)^i$

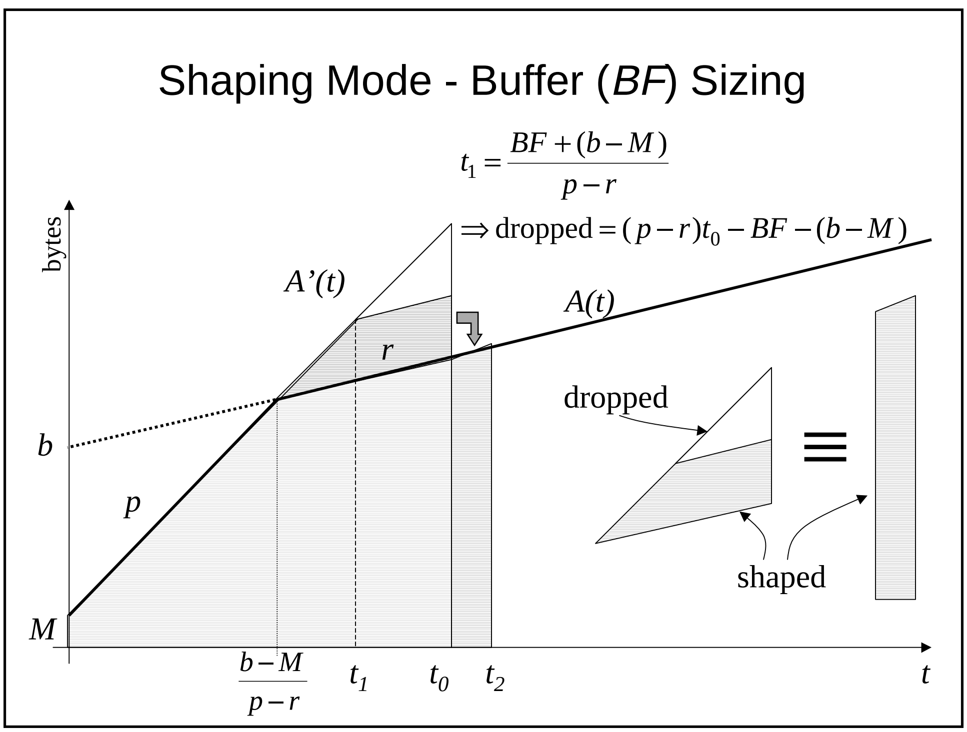
$$T_b = \frac{b}{p}\frac{1}{1-\frac{r}{p}} = \frac{b}{p-r}$$

  - Simpler derivation based on fluid model
    - Time to drain b at rate p-r

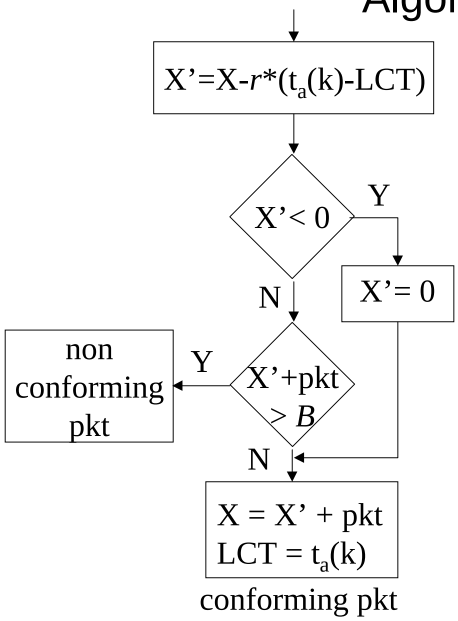
Dropping/Marking Mode




Shaping Mode
$M + pt_0 = rt_1 + b$
$\Rightarrow t_1 = \frac{p}{r} t_0 - \frac{b-M}{r}$
bytes
A'(t)
A(t)
r
b
p
M
$\frac{b-M}{p-r}$
t_0
t_1
t
t_1-t_0 : Maximum shaping delay


Shaping Mode - Buffer (BF) Sizing
$t_1 = \frac{BF + (b-M)}{p-r}$
$\Rightarrow \text{dropped} = (p-r)t_0 - BF - (b-M)$
bytes
A'(t)
A(t)
r
dropped
b
p
shaped
M
$\frac{b-M}{p-r}$
t_1
t_0
t_2
t

"Continuous State" Leaky Bucket Algorithm



Approach: Maintain running count X of token deficit

LCT: last conformance time

$t_a(k)$: arrival time of pkt k

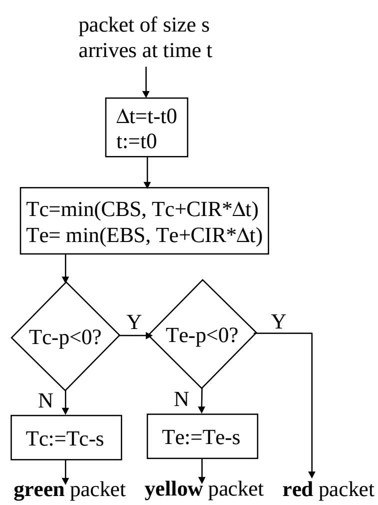
X: token deficit (in bytes)

pkt: packet size

$r*(t_a(k)\text{-LCT})$: potential credit accumulated since arrival of last conformant pkt

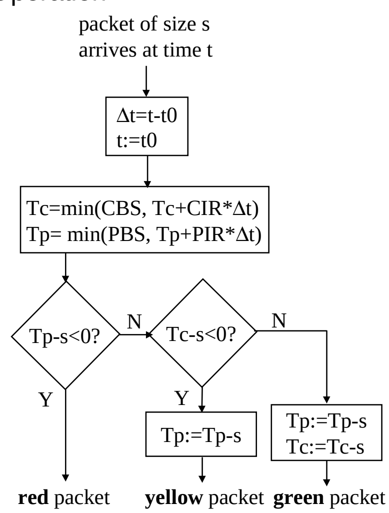
Other Token Bucket Variations (1)

- Single rate three color marker
  - Extends burst size limit to allow two priority levels
  - Three colors identify outcome of conformance test
    - green=conformant
    - yellow=non-conformant but within limit of excess burst
    - red= non-conformant and beyond excess burst
  - Essentially same algorithm as used in Frame Relay
- Parameters
    - Committed Information Rate (CIR)
    - Committed Burst Size (CBS)
    - Committed token count (Tc)
    - Excess Burst Size (EBS)
    - Excess token count (Te)
- Operation



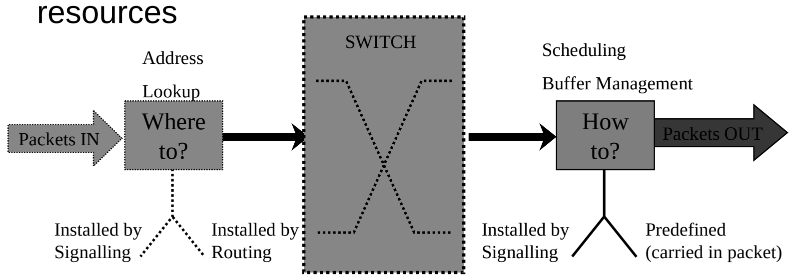
Other Token Bucket Variations (2)

- Two rates three color marker
  - Separate control for peak rate and committed rate with individual burst sizes
  - Similar to ATM dual leaky bucket but does not mandate discarding of packets that do not conform to peak rate contract
  - Marking
    - Red if fails peak rate check
    - Yellow if pass peak rate check but fails committed rate check
    - Green if pass both checks
  - Parameters
    - Committed Information Rate (CIR)
    - Committed Burst Size (CBS)
    - Committed token count (Tc)
    - Peak Information Rate (PIR)
    - Peak Burst Size (PBS)
    - Peak token count (Tp)
- Operation



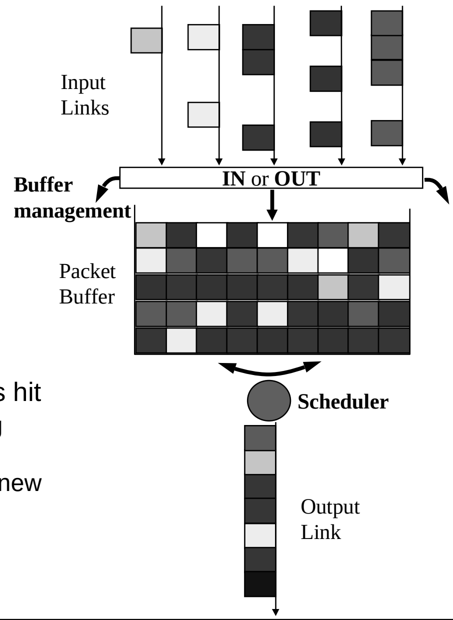
Scheduling & Buffer Management

- Where are they used?
  - Any place where congestion can occur
- What are they?
  - Data path mechanisms that makes storage and transmission decisions on packets
- What do they do?
  - enforce service guarantees and/or fair access to resources



Buffer Management and Scheduling Goals

- Buffer management
  - When to drop or accept a packet
- Scheduling
  - Which packet to send next
- Main goals
  - Service *guarantees*
    - bandwidth, delay, loss, etc.
  - *Sharing* of excess resources
    - Fair sharing
  - Minimize data and control paths hit
    - Minimize per-packet processing requirements
    - Simple decisions on whether a new flow can be admitted



Scheduling

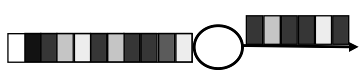
- Types of guarantees
  - Bandwidth: Make sure that a given flow gets *enough* transmission opportunities when it has packets waiting to be transmitted (is backlogged)
    - Need to define *interval* over which this is measured
  - Delay: Ensure upper *bound* on the maximum (average) amount of time a packet can wait in the buffer
  - Jitter: Provide bound on the delay *difference* of consecutive packet transmissions (for the same flow)
- Fairness
  - Distribution of excess bandwidth (E) across active connections
    - $E = C - \Sigma$(reservations)
    - Fair allocation gives each one of N active connections E/N in addition to their reserved rate

Scheduling Mechanisms and Characteristics

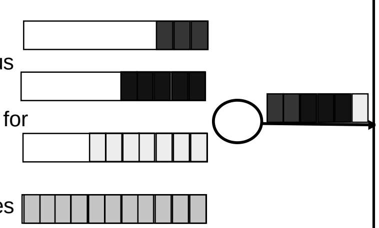
- Basic properties
  - Flow isolation
    Ability to guarantee service to one flow *independent* of the behavior of other flows
    - Important if incoming traffic is not constrained (leaky bucket)
  - Support of excess traffic and fairness
    If you send more than you are entitled to but resources are available, can you take advantage of it and if yes, how much?
    - How much deviation from the "fairest" scheduler
  - Implementation complexity
    - Computation of packet transmission times
    - Selection and update of next packet to transmit
  - Efficiency
    For a given set of guarantees and level of available resources, how many flows can I accept
    - Local vs end-to-end efficiency (network setting)
- Basic scheduling building block
  - Compute *desired transmission time* of packets
    - Based on service guarantees for each flow
    - Transmit packet with the *smallest* one
  - Schedulers differ in how they compute desired transmission times

Schedulers Examples (1)

- First-Come-First-Served (FCFS)
  - Packets are served in the order they arrive
    - Desired transmission time is time of arrival
  - Properties
    - Very simple to implement
    - Delay guarantees proportional to buffer size
    - No flow isolation or bandwidth guarantees
      - One flow can hog the entire link if unconstrained



- Priority queue
  - Multiple FCFS queues, where high priority queues always transmit before lower priority ones
    - Desired transmission time is time of arrival plus very large constant ($C_1 < C_2 < ... < C_N$)
    - Class *i* is guaranteed better delay than class *j* for *i*<*j*
    - Lower priority classes can be starved
      - Isolation is only from lower priority classes
    - Remains simple to implement (for few classes)



Scheduler Analysis (1)

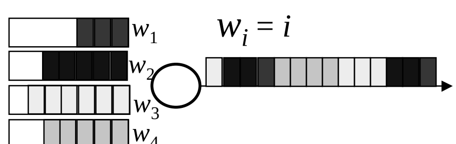
- FCFS scheduled can be analyzed using M/M/1 or M/G/1 queueing models
- Analysis of priority queue can be done using M/G/1 with priority
  - Average waiting time W_k for packets of class k is

$$W_k = \frac{\sum_{i=1}^{N} \lambda_i X_i^2}{2(1-\rho_1-\cdots-\rho_{k-1})(1-\rho_1-\cdots-\rho_k)}$$

  where λ_i is packet arrival rate for class i, ρ_i is load induced by class i ($\rho_i = \lambda_i/\mu_i$), and X_i^2 is the second moment of the transmission time of packets of class i
  - Can go to infinity when the total load of higher priority classes exceeds 1

More Scheduler Examples
Weighted Round-Robin & Virtual Clock

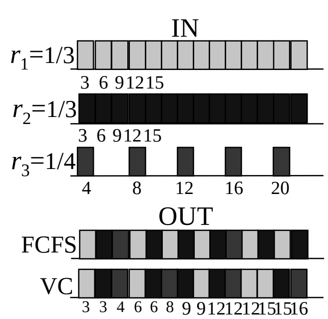
- Problems with previous schedulers
  - Hard to precisely allocate *individual* bandwidth guarantees



- Weighted Round-Robin
  - Each flow has its own queue and weight w_i
  - Server visits each queue in turn and transmits w_i packets (bits)
  - Simple but limited flexibility in allocating bandwidth & handling variable size packets
- Virtual Clock
  - Basic idea is to make packet priority a function of the rate allocated to its flow
    - Flow j is allocated r_j
    - On arrival of packet k of length L_j^k, "priority" of packet k is set to

$$W_j^k = W_j^{k-1} + \frac{L_j^k}{r_j}$$

    - Packets are transmitted in order of priority



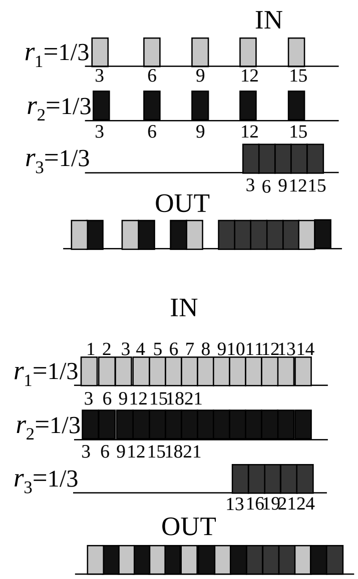
Problems With Virtual Clock

- Priority accumulation when idle
  - Idle flow can shut-off other flows for extended periods of time
  - Solution: Disallow accumulation

$$W_j^k = \max(W_j^{k-1}, a_j^k) + \frac{L_j^k}{r_j}$$

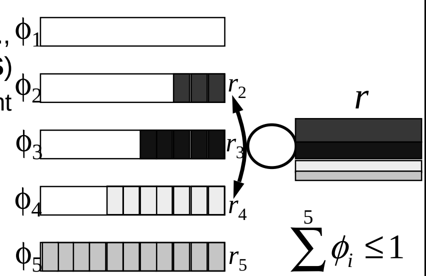
a_j^k : arrival time of packet k of flow j

- Problem with previous solution
  - Penalize excess usage even when no one else needed the bandwidth
    - Flow can be shutoff for extended periods if it previously used idle bandwidth
  - Unacceptable in the context of packet networks and adaptive applications



Improving on Virtual Clock
Weighted Fair Queueing (WFQ)

- Based on idealized fluid flow model, i.e., ***Generalized Processor Sharing*** (**GPS**)
  - Server can serve an infinitesimal amount (one bit) of data at each visit
  - Frequency of visit based on weights, amount served so far, and *active* flows
    - Server keeps rate allocated to each flow proportional to its weight ϕ_i
- Requirements
  - Each flow is assigned its own queue
  - Track and continuously compute amount transmitted for each active flow
- Characteristics
  - Provides *rate* and *delay* guarantees
  - Enforces flow isolation
  - *Fair* sharing of excess bandwidth



$$\sum_{i=1}^{5} \phi_i \leq 1$$

$$\phi_1 \geq \phi_2 \geq \phi_3 \geq \phi_4 \geq \phi_5$$

$$r_j = \frac{\phi_j}{\sum_{i=2}^{5} \phi_i} r, \; j = 2,3,4,5$$

Rate guarantee and fair allocation of excess bandwidth

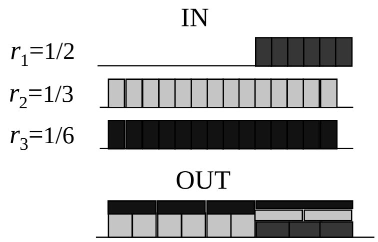
GPS Example

- Three flows with weights/rates $\phi_1=1/2$, $\phi_2=1/3$, $\phi_3=1/6$
- Initially, only flows 2 and 3 are active

$$r_2 = \frac{1/3}{1/3+1/6} = 2/3; r_3 = \frac{1/6}{1/3+1/6} = 1/3$$

- Flows 1, 2, and 3 are ultimately active

$$r_1 = \frac{1/2}{1/2+1/3+1/6} = 1/2, r_2 = \frac{1/3}{1/2+1/3+1/6} = 1/3; r_3 = \frac{1/6}{1/2+1/3+1/6} = 1/6$$



From Fluid to Packets

- How much deviation from the fluid model (GPS) do packets (PGPS=WFQ) introduce and how to minimize it?
  - Cannot interrupt packet transmission once started
    - Granularity in how transmission opportunities are allocated
    - Inability to change decision even if higher priority (allocated rate) packets arrive
- Approach
  - Emulate the fluid system (GPS) as closely as possible
    - Desired transmission time is *finish* transmission time in fluid system
    - Select packet with *smallest* finish transmission time in the fluid system (*assuming there would be no more arrivals after this time*)
- Issues
  - Can we bound discrepancies with fluid model (PGPS vs GPS)?
  - Complexity of "simulating" the fluid system to keep track of its transmission times

Bounding The Difference With Fluid Model

- Basic results
  - Packet finish transmission times in PGPS are *at most one maximum size packet later* than in GPS

    $$\forall p, \hat{F}_p - F_p < \frac{L_{\max}}{r}$$

    Note: This is only an upper bound, i.e., WFQ could be significantly ahead of GPS

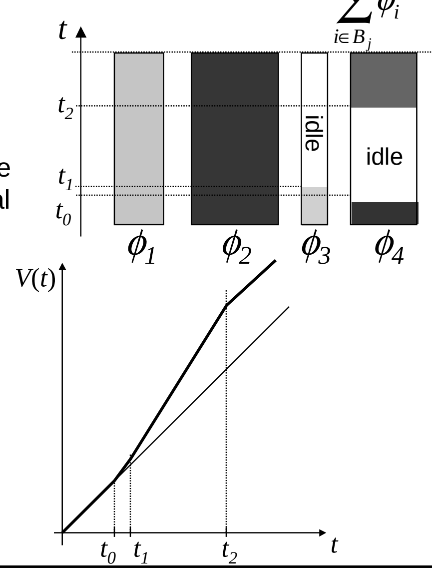
    $L_{\max}$ is maximum packet size, r is link rate, F_p is finish time of pth packet in GPS, and $\hat{F}_p$ is finish time of pth packet in PGPS
  - Queue sizes in PGPS and GPS is *at most one maximum size packet larger* than in GPS

    $$\forall t, \hat{Q}_i(t) - Q_i(t) \le L_{\max}$$
- Basic implementation issue is computation of finish times in fluid system
  - Hard to track and continuously update
  - Solution based on virtual time approach,
    - Keep track of the marginal rate at which backlogged sessions receive service, and update at each event (packet arrival and departures)

Virtual Time in Fluid Model

- Virtual time measures the marginal rate at which fair service should be given
  - Enables tracking of how much service each flow should have received
- Previously idle flow becoming active changes the *rate* at which the virtual time evolves
- Newly active flow becomes immediately eligible for service, but only from the corresponding virtual time level
  - No credit for period when idle
- Main issue is to determine simple method for computing virtual time evolution

Virtual time $V(t)$ evolves as $\frac{t}{\sum_{i \in B_j} \phi_i}$



A Virtual Time Implementation of WFQ

- The virtual time $V(t)$ captures the evolution of the rate of service for backlogged connection

$$V(t_{j-1}+\tau)=V(t_{j-1})+\frac{t}{\sum_{i\in B_j}\phi_i}$$

where B_j is the set of backlogged connections between times t_j and t_{j-1}

- Using virtual time to compute packet transmission times (*service tags*) in GPS
  - Define virtual start and *finish service times* for packet *k*

$$S_i^k=\max\{F_i^{k-1},V(a_i^k)\} \quad \text{(definition)}$$

$$F_i^k=S_i^k+\frac{L_i^k}{r\phi_i} \quad \text{(Why?)}$$

- Advantages of virtual start and finish service times
  - Updated only at packet arrivals and departures
  - Virtual finish time can be determined at packet arrival time'
  - Packet are served in order of virtual finish time
  - Worst case complexity can be $O(N)$ for N connections

End-to-End vs Single Node Performance (1)

- Latency Rate Server (LRS) framework
  - Definition: An LRS server is characterized by two parameters, r_i and Θ_i, for session i, such that for all time intervals $(\tau, t]$ during which session i is continuously backlogged, the amount of service $W_{i,j}(\tau,t)$ offered to session i is lower bounded by

$$W_{i,j}(\tau,t)\geq \max(0,r_i(t-\tau-\Theta_i))$$

- Basic model characteristics
  - Ability to provide rate guarantees to individual flows
  - Bounded *irregularity* in how service is delivered, i.e., latency
  - Example: WFQ has a latency of $\frac{L_i}{r_i}+\frac{L_{\max}}{r}$

    (L_i is maximum packet size for session i, and r is link speed)
- Question is how do single node guarantees translate into end-to-end guarantees, i.e., end-to-end delay bound?

End-to-End vs Single Node Performance (2)

- Traffic envelope
  - In order to bound delay, we need to know how much traffic a flow is injecting
  - Traffic envelope specified through token bucket (P_i,ρ_i,σ_i), i.,e.,
    $A_i(\tau,t) \leq \min\left(\sigma_i + \rho_i(t-\tau), P_i(t-\tau)\right), \forall t,\tau \text{ and } \tau < t$
  - Basic result on concatenation of LRS
    - Two LRS in series with latencies Θ_1 and Θ_2 are equivalent to a single LRS with latency $\Theta = \Theta_1 + \Theta_2$
- Bound on end-to-end delay through N LRS is then given by

$$D_i \leq \left(\frac{P_i - r_i}{r_i}\right)\left(\frac{\sigma_i}{P_i - \rho_i}\right) + \sum_{j=1}^{N} \Theta_i^j$$

  where Θ_i^j is the latency of the *jth* LRS for flow i
- Bound on buffer requirements at *kth* LRS is also available

$$B_i \leq (P_i - r_i)\left(\frac{\sigma_i}{P_i - \rho_i}\right) + \rho_i \sum_{j=1}^{k} \Theta_i^j$$

Call Admission and Service Guarantees With WFQ

- Call admission decides if a new flow can be accepted
  - Rate guarantee: new flow is entitled to rate $r_i = \phi_i r$
  - Delay guarantee: new (P_i,ρ_i,σ_i) flow needs rate r_i to ensure its end-to-end delay bound
  - Call admission rule simply requires that $\sum_i r_i \leq r$
    - Buffer sizing also needed (depends on position in path - see Guaranteed Service model)
- Service guarantees
  - Rate guarantee
  - End-to-end delay guarantee
  - Fair access to excess bandwidth
  - But limited jitter control

Other Schedulers in The LRS Family (1)
Self-Clocked Fair Queueing (SCFQ)

- Similar to WFQ but aims at simpler computations of virtual finish times, i.e., avoids tracking GPS
  - Virtual time is *service tag* of packet in service
    - *O(1)* complexity of virtual time update
  - Service tag F_i^k computed as for PGPS

$$S_i^k = \max\{F_i^{k-1}, V(a_i^k)\}$$

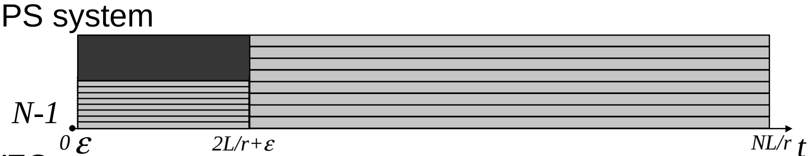
$$F_i^k = S_i^k + \frac{L_i^k}{r\phi_i}$$

- Main difference is in maximum discrepancy from GPS

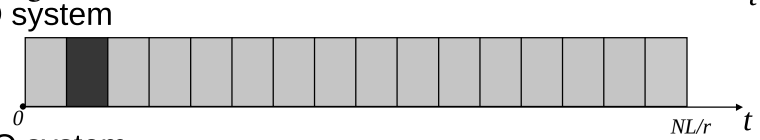
$$\forall p, \hat{F}_p - F_p < (N-1) \times \frac{L_{\max}}{r}, N \text{ is number of flows}$$

SCFQ Difference With GPS

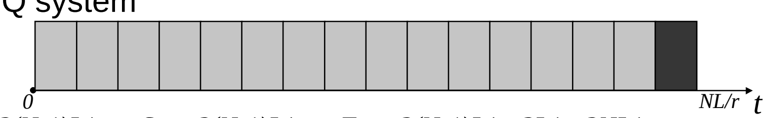
- *N* flows, with *N-1* flows at rate *r/2(N-1)* and 1 flow at rate *r/2*
  - *N-1* flows with rate *r/2(N-1)* and flow *N* with rate *r/2*
  - Packets for first *N-1* flows arrive at *t=0*, packet for flow *N* arrives at *t=ε>0*
- GPS system



- WFQ system



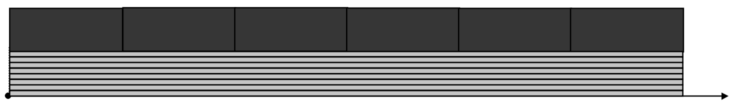
- SCFQ system



$V(\varepsilon)=2(N-1)L/r \Rightarrow S_N = 2(N-1)L/r \Rightarrow F_N = 2(N-1)L/r+2L/r=2NL/r$

Other Schedulers in The LRS Family (2)

- Worst case fair weighted fair queueing (WF^2Q)
  - *Service tag* is virtual service finish time, but *only among eligible* flows, i.e., flows with virtual start time <= current virtual time
  - Similar complexity and same difference with GPS as WFQ but better *fairness* properties
- GPS behavior



- WFQ behavior

- WF^2Q behavior

Fairness Criteria

- Ideally, each session should receive exactly its *normalized* amount of service, i.e., as per GPS
- Packet based transmission introduces discrepancies
- Goal is to estimate how much discrepancy each type of scheduler allows
- Some possible measures
  - Difference in service received For all time intervals $(\tau,t]$ where sessions i and j are backlogged

$$\left| \frac{W_i(\tau,t)}{r_i} - \frac{W_j(\tau,t)}{r_j} \right| \leq \Phi$$

  - Worst case Fair Index (WFI)

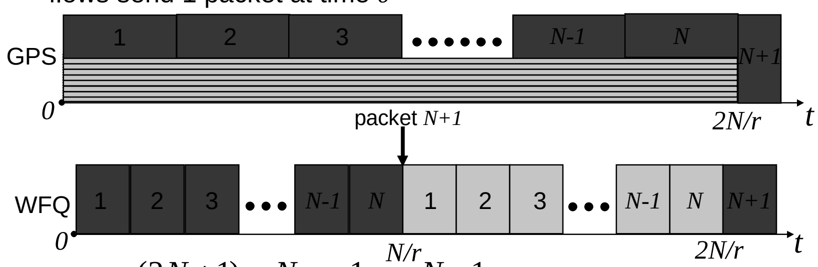
$$d_{i,S}^k \leq a_i^k + \frac{Q_{i,S}(a_i^k)}{r_i} + C_{i,S} \text{ and } c_{i,S} = \frac{r_i C_{i,S}}{r}$$

$$c_S = \max_i \{c_{i,s}\} \text{ is WFI for scheduler } S$$

- GPS is the *"benchmark"* as it has Φ and WFI of 0

Fairness Criteria - WFQ

- WFQ has a WFI that can grow linearly with the number of session
  - Flow 1 with rate *r/2* and *N* flows with rate *r/2N*
  - Flow 1 sends *N* packets at time *0* and 1 packet at time *N/r*, all other flows send 1 packet at time *0*



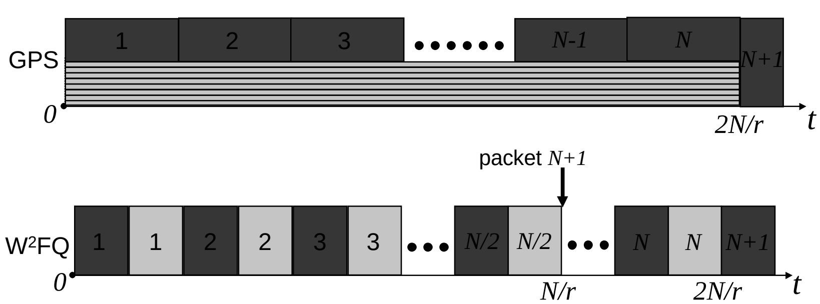
$$C_{1,WFQ} \geq \frac{(2N+1)}{r} - \frac{N}{r} - \frac{1}{r/2} = \frac{N-1}{r}$$

$$\Rightarrow c_{WFQ} \geq C_{1,WFQ}\frac{r_1}{r} = \frac{N-1}{2r}$$

Exact expression is available for c_{WFQ}

Fairness Criteria - WF²Q

- The WFI of WF²Q is *independent* of the number of sessions
  - For same example as with WFQ



$$C_{1,W^2FQ} \geq \frac{(2N+1)}{r} - \frac{N}{r} - \frac{N/2+1}{r/2} = -\frac{1}{r}$$

Fairness Criteria - WF²Q, contd.

- WFI of WF²Q?
  - Another example
    - Flow 1 with rate ε has packet at time 0, flow 2 with rate $r/2$ has packet at time 0^+, and flow 3 with rate $r/2$ has packet at time 0^{++}
    - Packets go out in order 1, 2, and 3, so that

$$d_3 = \frac{3L}{r} \Rightarrow C_{3,W^2FQ} \geq \frac{3L}{r} - \frac{L}{r/2} = \frac{L}{r}$$

- In general, it is possible to show that

$$C_{i,WF^2Q} = \frac{L}{r_i} \text{ and } c_{WF^2Q} = \frac{L}{r}$$

- As a result WF²Q is as fair as can be for a packet scheduler
- But remains complex because of Virtual Time computations
  - WF²Q+ provides the benefits of WF²Q with lower implementation complexity by using a modified Virtual Time function

WF²Q+

- Definition of Virtual Time function of WF²Q+

$$V_{WF^2Q+}(t+\tau) = \max\left(V_{WF^2Q+}(t) + \tau, \min_{i \in B(t)}\left(S_i^{h_i(t)}\right)\right)$$

  - where $B(t)$ is the set of backlogged flows at time t, and $S_i^{h_i(t)}$ is the virtual start time of the packet at the head of flow i queue
- Simplification of Virtual Start and Finish times
  - Updated only when a packet reaches the head of its queue

$$S_i = \begin{cases} F_i, & \text{if } Q_i(a_i^k -) \neq 0 \\ \max\{F_i, V(a_i^k)\} & \text{if } Q_i(a_i^k -) = 0 \end{cases}$$

$$F_i = S_i + \frac{L_i^k}{r\phi_i}$$

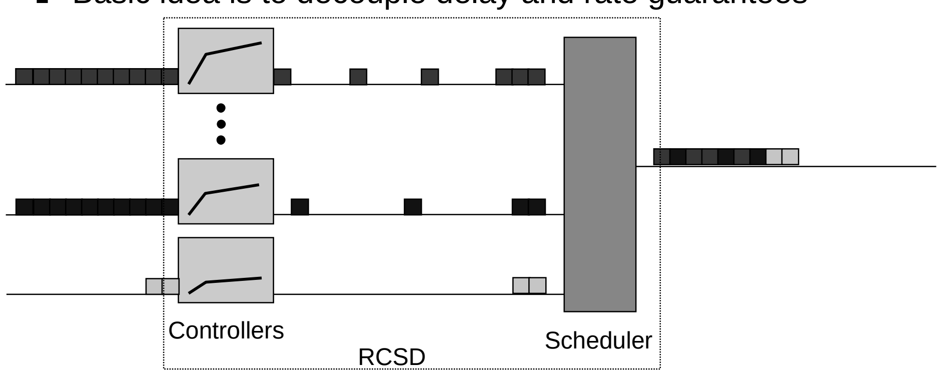
- Both Virtual Time updates and sorting of virtual finish times can be done in $O(\mathrm{Log}N)$ complexity

Summarizing Where We Are

- Identified "family" of *fair queueing* (FQ) packet schedulers with the following properties
  - Ability to guarantee transmission *rate* to individual flows *independent* of the behavior of other flows
  - Ability to guarantee fair (proportional to allocated rate) access to excess bandwidth with *bounded* deviation from ideal fluid model
  - Ability to guarantee local and end-to-end *delay* bound to policed (token bucket) flows *independent* of the behavior of other flows
  - Computation of required buffer sizes at each hop (function of burstiness increase as packets propagate through the network)
  - Various trade-offs between implementation complexity and tightness of delay bounds and fairness guarantees
- Some limitations of this family of schedulers
  - *Delay* guarantees provided through *rate* guarantees
    - Potentially inefficient for low bandwidth flows
  - Limited (no) control of jitter and burstiness increase

Rate Controlled Service Disciplines

- Basic idea is to decouple delay and rate guarantees



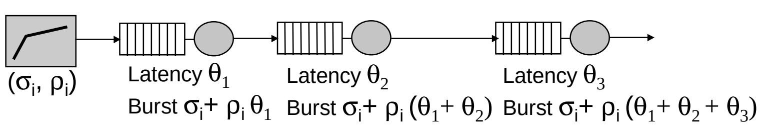
  - Rate controllers release packets to scheduler *only* when conformant with service contract
    - Enforces rate limitations and, therefore, ensures guarantees
  - Scheduler provides delay differentiation
    - Can decide which packet to send *independent* of rate

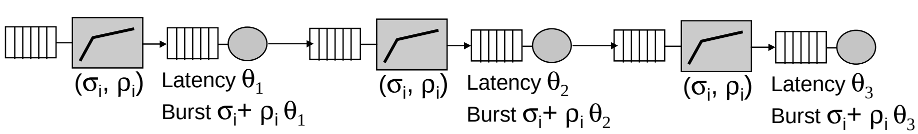
Some Properties of RCSD

- Capable of all the service guarantees of Fair Queueing packet schedulers
  - Rate guarantees
  - Tight local and end-to-end delay bounds
    - Introduction of rate controllers *does not* increase maximum delay
- And also
  - Tight jitter control
  - Burstiness control and, therefore, lower buffer requirements
  - Greater schedulability region, i.e., can accommodate more flows than FQ schedulers for a given set of guarantees
- But
  - Rate controllers increase *average* delay
  - No support for excess traffic and fair sharing in base version
    - Rate controllers make the system non-work-conserving
    - Possible extension (logical rate controller) provides some support for excess traffic

Why Lower Buffer Requirements?

- Rate control at each node means that the traffic is *reshaped* to its original envelope
  - Eliminates additional burstiness introduced by upstream nodes





  - Reshaping buffer at node k

$$B_k^{(R)} = \rho_i \theta_{k-1}$$

  - Scheduling buffer at node k

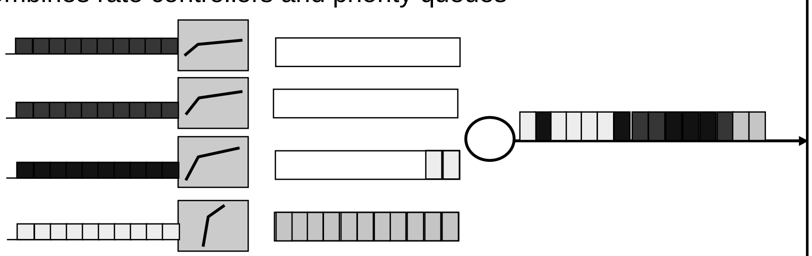
$$B_k^{(S)} = \sigma_i + \rho_i \theta_k$$

Why No Impact to Delay Bound?

- Only early packets are reshaped
  - delay added by reshaper is never more than available delay "budget"
- Illustration in the case of deadline based scheduler
  - Input is characterized by (σ_i, ρ_i) envelope
  - Scheduler guarantees local deadline d_i
  - Reshaping delay at next node?
- Informal "proof"
  - If *all* packets are delayed by d_i, traffic envelope remains unchanged
    - $\Rightarrow$ No need to reshape at next hop
  - Packets delayed (reshaped) at next node if they find no tokens
    - $\Rightarrow$ They arrived before the token was generated
    - But token is generated on time if they suffered the maximum delay
    - $\Rightarrow$ The time spent waiting for a token is no more than the difference between the worst case delay d_i and the actual delay experienced by the packet

Examples of RCSD

- Rate Controlled Static Priority (RCSP)
  - Combines rate controllers and priority queues



  - Main benefits
    - Avoids starvation of low priority classes
    - Allows low delay and low rate allocation
  - But
    - Delay guarantees are coarse
    - Call admission is complex (need to account for interactions between classes before making a decision)

Worst Case Delay for RCSP

- N priority classes with rate controllers (σ_i,ρ_i)
  - Let t_i be the time at which the server is finished serving the initial burst of class i assuming all classes are *greedy*
    - t_i is the worst case delay for class i

$$t_1 = \frac{\sigma_1}{C - r_1}$$

$$t_2 = \frac{\sigma_2 + r_2 t_1}{C - r_1 - r_2}$$

$$\Rightarrow t_j = \frac{\sigma_j + r_j t_{j-1}}{C - \sum_{i=1}^{j} r_i}$$

- Need to empty burst of higher priority classes, and after that service rate is reduced by rate of higher priority classes
- Substantial burstiness increase affects delay and buffering at subsequent nodes

A Better RSCD: RC-EDF

- Use a better scheduler, i.e., Earliest-Deadline-First (EDF)
  - *Optimal* policy for delay guarantees in the single node case
  - Each class (flow) is assigned a *deadline* representative of its local delay requirements
  - Packets released from the scheduler are marked for transmission based on their deadlines
    - All packets from the "*same*" burst have the same deadline
  - Scheduler picks for transmission packet with the smallest deadline
- Key properties
  - Decoupling of rate (shaper) and delay (scheduler) guarantees allows more efficient operation than rate based schedulers
    - Accept more flows
    - Lower buffer requirements
    - Lower end-to-end jitter (only delay variation is contributed by last node)
  - Shaper selection is key to tight end-to-end delay bounds
    - Typically stricter than original traffic envelope
  - Shaper is non-work-conserving
    - No native support for excess traffic
      - Possible *work conserving* extension

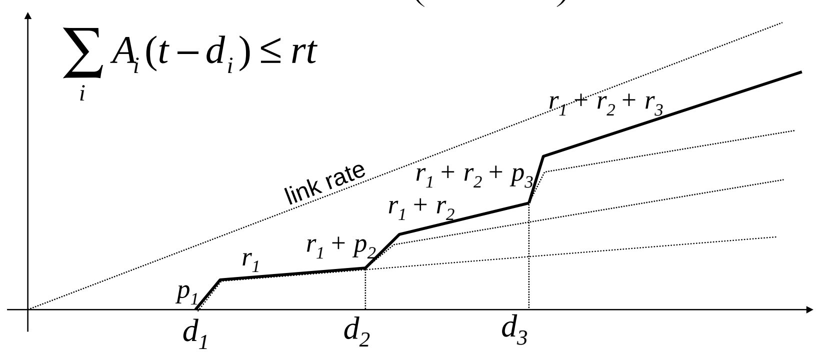
Impact of Shapers (1)

- End-to-End delay bound is of the form

$$D = \text{access shaping delay} + \sum_{i=1}^{N} d_i$$

- But local deadline d_i can be guaranteed to flow (σ_i, ρ_i) only if

$$\min\{k+1, N\}L + \sum_{n=1}^{k} \sigma_{i_n} \le d_{i_k}\left(r - \sum_{n=1}^{k-1} \rho_{i_n}\right) + \sum_{n=1}^{k-1} \rho_{i_n} d_{i_n}, 1 \le k \le N$$



Impact of Shapers (2)

- The smoother the traffic, the tighter the deadline the network can give
- The smoother the traffic, the bigger the access delay
- No known optimal partitioning of delay budget between access and network delay
  - Depends on number of nodes in network
    - Long paths make access smoothing more effective
  - Common approach is to use WFQ delay bounds as deadlines
- A work conserving shaper
  - Maintain two queues
    - One for non-eligible packets
    - One for eligible packets (sorted according to their deadline)
  - Select for transmission the packet from the eligible queue with the smallest deadline
  - If the eligible queue is empty select a packet from the non-eligible queue (according to some policy, e.g., sum of deadline and eligibility time)

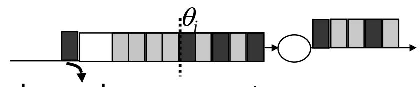
Buffer Management

- Scheduler manages access to link bandwidth while buffer manager controls access to storage resources
  - Not always enough room to store arriving packets
  - Needed to provide service guarantees
    - Permission to transmit is of no use if no packets are available for transmission in memory
- Two major decisions
  - When to discard a packet?
  - Which packet to discard?
- Classification of buffer management methods
  - Time of packet acceptance/discard decisions
    - At time of packet arrival or at any time
  - Granularity of information used for making decisions
    - per flow buffer count (stateful) vs global count (stateless)

Major Buffer Management Methods

- One time decision (no regrets allowed)
  - Discard/accept packet based on buffer state at arrival time
    - Simple on-line decision but off-line process pre-allocates resources ahead of time
  - Example: Threshold based schemes
    - Packets are discarded when the (total/class/flow) buffer occupancy (or count) exceeds a given limit (the threshold)
- Multiple decisions (change your mind each new packet)
  - For each packet arrival (departure) determine which packets are in and which are out
    - No off-line pre-allocation but complex on-line decisions and memory structure
  - Example: Pushout schemes
    - Remove low priority packet from memory to make room for arriving new high priority packet
- Trade-off between
  - Complexity, performance, and predictability

Threshold Schemes



- Accept/discard decision based on current resource usage
  - Deterministic
    - Incoming packet of type *i* is dropped when buffer content exceeds θ_i
    - Motivation
      - Threshold computed such that $B\text{-}\theta_i$ is sufficient to guarantee performance of type *i* (or higher) packets
  - Randomized
    - Type *j* packet is dropped with probability p_j when buffer content $\geq \theta_j$
    - Motivations
      - Preventive method, i.e., start dropping before onset of congestion
      - Likelihood of being dropped is *in proportion* to volume of traffic
      - Avoids possible synchronization, i.e., always dropping the same flow
    - Examples: Random Early Discard (RED) and Weighted RED
- Various granularity of allocation are possible
  - Global (priority classes), per flow (maintain count)

Random Early Detection (RED)

- The RED algorithm is a popular mechanism to implement congestion avoidance for adaptive traffic such as TCP
- RED is based on the following components
  - Computation of average queue size Q
    - Exponential averaging: $Q_n = (1-\alpha)\, Q_{n-1} + \alpha\, q_n$, $\alpha \ll 1$
  - Probabilistic dropping that varies with average queue size
    - No dropping if Q is below *minimum threshold* θ_{min}
    - All packets are dropped if Q is above *maximum threshold* θ_{max}
    - When $\theta_{min} \leq Q \leq \theta_{max}$ packet is dropped with probability $p(Q)$
      - p(Q) increases linearly from 0 to p_{max} between θ_{min} and θ_{max}

$$p(Q) = p_{max}(Q - \theta_{min})/(\theta_{max} - \theta_{min})$$

    - A *count* variable is used to track the number of packets since the last drop and ensure that drops occur without too much delay

$$p'(Q) = p(Q)/[1 - count \times p(Q)]$$

RED Status

- RED "home page" maintained by Sally Floyd
  - http://www.aciri.org/floyd/red.html
- RED has been implemented in a number of routers and recommended by the IETF as part of congestion avoidance schemes
  - RFC 2309 (Informational)
- Many pros and cons that are still being debated
  - Fairness
  - Lack of bias against bursty flows
  - Avoidance of synchronization
  - Difficulty in *properly* setting the many parameters that RED involves
    - Wide range of performance fluctuations
  - Check RED home page for information and updates

Pushout Schemes

- Pushout rules
  - Accept all packets when there is space in the buffer
  - When buffer is full, packet of type i is accepted by pushing out a packet of the lowest priority less than i, if any
  - Motivations
    - High priority packets have precedence over lower priority ones only in the presence of congestion
      - Favors efficiency by maximizing buffer utilization
  - Implementation issues in pushout schemes
    - Need to keep track of position of *all* packets of *all* types at *all* times (linked lists for each packet type)
    - Need to update linked lists for each packet arrival and departure
    - Difficult to accommodate variable size packets (may need to pushout several packets to accommodate a large packet)
    - Selection of which packet to discard (first, last, random)

Efficiency vs Accuracy

- Global schemes are simpler but may provide only loose service guarantees
  - Impact of excess traffic across streams
  - Fairness issue in accessing idle resources
- Per flow schemes can provide greater control
  - Tracking of individual flow resource usage

  at the cost of higher complexity and loss in efficiency
- Loss of efficiency is caused in part by resource partitioning
  - Range of possible trade-offs
    - Complete partitioning
    - Complete sharing
    - Mixed policies

Buffer Sharing Options

- Complete partitioning ($\sum_i B_i = B$)
  - Each flow gets its dedicated share of the total memory
  - Potentially inefficient as buffers are left unused
- Complete sharing ($\sum_i B_i = 0$)
  - Maximum efficiency but unable to ensure strong flow isolation
  - Minimum complexity (no per flow state)
- Limited sharing
  - Maximum per flow buffer content ($\sum_i B_i > B$)
    - Trade-off between efficiency and flow isolation
  - Minimum per flow allocation ($\sum_i B_i < B$)
    - Base guarantee with potential for improved efficiency
    - Still need to specify method for sharing excess buffer capacity
- How to control sharing across flows?
    - Preserve service guarantees while improving efficiency
    - What information to rely on?

Sample Buffer Sharing Choices

- Basic scheme has total memory size of B and per flow allocations of B_i
  - At each packet arrival (departure), update buffer count of flow and make packet acceptance
    - ```
      If (b_i + p_i,k <= B_i) {
          accept packet;
          b_i = b_i + p_i,k;
          }
      else drop packet;
      ```
- Sharing options
  - Aggressive ($\sum_i B_i > B$) design
    - Greater efficiency but lower protection
  - Conservative ($\sum_i B_i \le B$) design
    - Strong service guarantees, but need rule for fair access to excess buffers

Sample Buffer Sharing Rule

- Example
  - Flow buffer allocation is $B_i+f_i\,(B-B')$, where
    - $f_i = 1/N$, and N is the number of flows currently "active"
    - B' is the amount of reserved buffers, i.e., $B' = \sum_i B_i$
  - Issues
    - Definition of active flows, e.g., no packets in buffer, or $b_i \le B_i$?
    - Additional complexity of tracking active flows
- A simpler alternative
  - No flow can occupy *more* excess buffers than are left available
  - Only requires tracking of total excess buffer count
  - Can be shown to result in fair allocation of excess buffers
  - Some minor inefficiency when only few flows need excess buffers
    - For k active flows, the amount of unusable buffer space is $(B-B')/k+1$
      ```

Coupling Buffer Management & Scheduling

- Scheduling affects the *regularity* of service of a flow
- Buffers are used to absorb irregularities in *both* packet arrivals and transmission opportunities
  - The more regular the arrival process
    - The smaller the required buffer size
  - The more regular the service
    - The smaller the required buffer size, e.g., LRS
    - The smaller the required buffer size at the *next* node (more regular arrivals)
- Bottom line is that selection of scheduler and buffer management strategy needs to be done **jointly**

Call Admission

- Goals
  - Given
    - Desired service guarantees, e.g., loss or delay probability
    - Traffic characteristics, token bucket, max rate, etc., of new request
    - Current availability of network resources (bandwidth, buffer)
  - Determine
    - Availability of sufficient resources to accommodate new request
- General constraints
  - Keep it simple!
    - Minimize storage requirements and computational cost
    - Real time acceptance or rejection decisions

Design Issues

- Incremental decisions
  - Avoid computations that involve **all** flows when adding/removing one
- Accuracy
  - System resources, e.g., finite buffers
  - Traffic constraints
    - Progressive arrivals
    - Finite peak rates
- Complexity
  - Minimize on-line computations
  - Limit storage requirements
- Flexibility
  - Traffic patterns
  - Service guarantees

Effective or Equivalent Bandwidth

- Question
  - How much bandwidth given a desired loss probability ε, buffer size *X*, and source characteristics ρ, *b*, and R_{peak}
- Approach
  - Assume source can be viewed in isolation and use fluid flow representation
  - Invert expression for overflow probability and solve for the desired capacity *C*
    - Overflow probability is of the form $\varepsilon = \beta e^{-\delta x}$, where

$$\beta = \frac{(C - R_{peak}) + \varepsilon\rho(R_{peak} - C)}{(1-\rho)C}, \delta = \frac{C - \rho R_{peak}}{b(1-\rho)(R_{peak} - C)C}$$

    - Assume β≈1

$$\hat{C} = \frac{\alpha b(1-\rho)R_{peak} - X + \sqrt{[\alpha b(1-\rho)R_{peak} - x]^2 + 4X\alpha b\rho(1-\rho)R_{peak}}}{2\alpha b(1-\rho)}, \text{ where } \alpha = \mathrm{Ln}(1/\varepsilon)$$

Some Interesting Properties

- Effective bandwidths are (asymptotically) additive
  - For large enough buffers, the effective bandwidth of the superposition of *heterogeneous* ON-OFF sources is the *sum* of their *individual* effective bandwidth, i.e.,

$$\hat{C} \approx \sum_{i=1}^{N} \hat{c}_i$$

    - Based on separability results for the eigenvalues of the solution to the differential matrix equation describing the evolution of the queue length
- Benefit is that we can now consider each connection in isolation when making a call admission decision
  - Incremental decision process

Some Effective Bandwidth Limitations

- Based on exponential source assumption
  - Extensions to general sources are substantially more complex
  - Possible approximations
    - Moment matching
      - Compute *equivalent* exponential ON and OFF source by *resizing* ON and OFF periods based on (second) moment
    - Measurement based approach
      - Identify exponential source that yields the same measured queue size (based on measuring level crossing probability)
        - If bigger than expected, increase burst size, and recompute effective bandwidth
        - If smaller than expected, decrease burst size, and recompute effective bandwidth
        - Continue until convergence
- Does not capture well the effect of statistical multiplexing
  - Conservative for large number of high speed bursty sources
  - Some other (complementary) method is needed

A Stationary Bit Rate Approximation

- Goal is to complement effective bandwidth approach in scenarios where it is inefficient
  - Large number of bursty and high peak rate sources
- Basic idea is to ignore the potential usefulness of buffers when peak rate is high and bursts are large
  - Buffers are of little help when congestion can last
  - Focus instead on probability of congestion, i.e., probability that the aggregate rate R_{tot} of simultaneously active sources exceeds the available (allocated) bandwidth

$$\Pr(R_{tot} > \hat{C}_S) \le \varepsilon$$

  - Distribution of aggregate rate is *independent* of distribution of ON and OFF periods
    - Depends only of probability of being ON or OFF

A Stationary Bit Rate Approximation - contd.

- When many sources are multiplexed, we can use the law of large numbers to approximate the distribution of R_{tot} by a Gaussian distribution
- Computing $\hat{C}_S$ requires inverting a Gaussian distribution
  - Good and simple approximation can be obtained

$$\hat{C}_S \approx m + \alpha'\sigma, \text{ where } \alpha' = \sqrt{-2\ln(\varepsilon) - \ln(2\pi)}$$

  - where *m* is the sum of the average rates of all the connections multiplexed and σ^2 is the sum of the variances of all the connections

$$m_i = \rho_i R_{peak}^{(i)}$$

$$\sigma_i = \sqrt{m_i (R_{peak}^{(i)} - m_i)}$$

Combining the Two

- Which approach to use when?
  - They are complementary as they over-estimate capacity in different regions
  - Combined approach simply amounts to taking the minimum of both approaches

$$\hat{C} = \min\left\{ m + \alpha' \sigma, \sum_{i=1}^{N} \hat{c}_i \right\}$$

- Advantages
  - Reasonably accurate over wide range of connections
  - Incremental call admission process

$$\mathrm{L} = \left(m = \sum_{i=1}^{N} m_i, \sigma^2 = \sum_{i=1}^{N} \sigma_i^2, \hat{C}_F = \sum_{i=1}^{N} \hat{C}_i \right)$$

$$\mathrm{L}' = \mathrm{L} \pm \left(m_j, \sigma_j^2, \hat{C}_j \right)$$

Measurement Based Admission Control

- Basic premise
  - Some users may not require *absolute* performance guarantees
  - Those users may not have traffic descriptors for their sessions or more typically these will be inaccurate
    - Equation based methods are only as good (bad) as the information they rely on
      - This can result in highly inefficient call admission rules
- Basic idea
  - Take the traffic descriptors only as initial estimates of traffic characteristics and use *measurements* to accurately estimate network load and make admission decisions

Summary of MBCAC Approaches

- Two basic components
  - Measurements of current traffic and its required resources
    - How to measure?
    - How to derive required resources from measurements?
  - Accounting for a new request
    - What parameters (peak rate, average rate, burst size, etc.)?
    - How to incorporate requirements of new request?
- A variety of proposed approaches
  - From simple measurements
    - Time window estimator, exponential averaging of load estimates
  - To complex measurements
    - Effective bandwidth curves, maximal traffic envelopes
  - Incorporation of new flow based on
    - peak rate, token rate, “effective” bandwidth estimates
- Current status
  - No clear and proven choice
  - See Infocom'00 paper by Breslau et al. for a recent comparison of proposed schemes

IP QoS Approaches

IP Services

- Two broad families
  - Aggregate service (relative and qualitative guarantees)
    - Differentiated Services (RFC 2474, RFC 2475, RFC 2597, RFC 2598)
  - Per flow service (quantitative and qualitative guarantees)
    - Integrated Services (RFC 2211, RFC 2212)
- Differentiated Service (packets identified by DS field)
  - Pre-configured set of service classes (behaviors)
  - Expedited Forwarding (local behavior only)
      - Virtual leased line type of service
  - Assured Forwarding (local behavior only)
    - Several service classes with drop precedence within each class
- Integrated Services (flow identified by SA/DA & ports)
  - RSVP based signalling installs per flow state in routers
  - Controlled Load Service
    - Loose loss and delay guarantees
  - Guaranteed Service
    - Hard end-to-end delay bound and zero losses

Differentiated Service

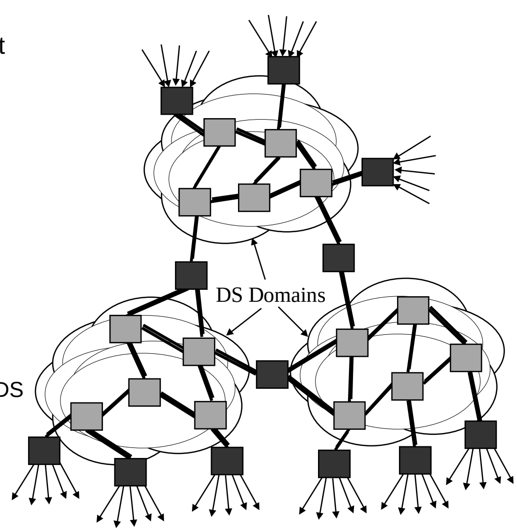
- Goals and motivations
  - Data path scalability
    - Coarse granularity *service classes* (no per flow state)
    - Minimum impact on packet forwarding performance
      - Realizable through simple mechanisms
  - Rapid deployment
    - Standardize service *codepoints* in IP header and associated expected **local** behavior (Per Hop Behavior - PHB)
      - Wide range of possible implementations
      - Avoid *"chicken and egg"* problem of signalling deployment and application/user support
- Status
  - Initial standardization effort complete
    - Definition of format of DS field (6 bits) in IP header (IPv4 and IPv6)
    - Two *behaviors:* Expedited Forwarding and Assured Forwarding
  - Interactions with Int-Serv and services definition in progress
  - Coarse signalling support (Bandwidth Broker) under investigation

Diff-Serv Terminology

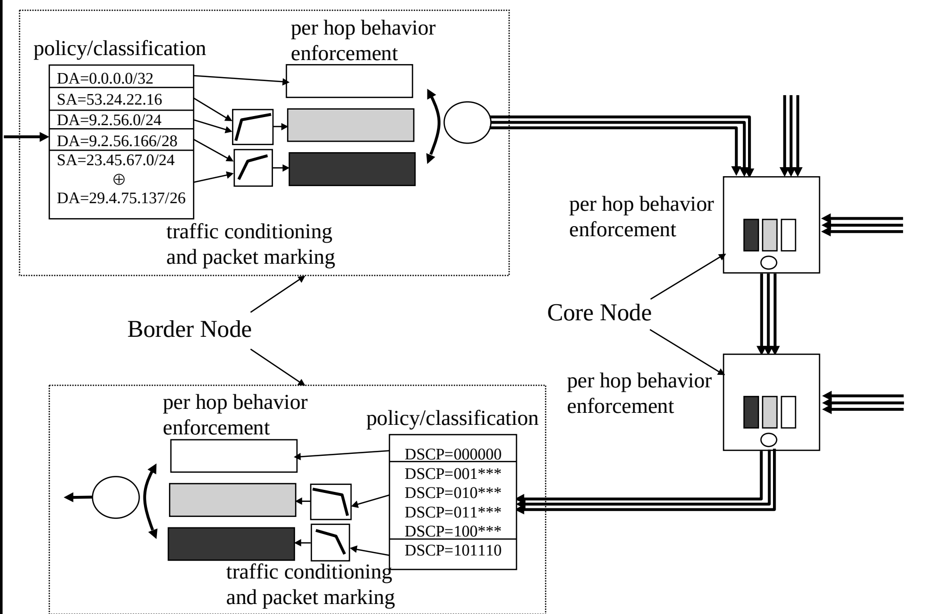
- DS field: First *six* bits of the IPv4 TOS octet or the IPv6 Traffic Class octet
- DS Code Point (DSCP): A specific value of the DS field
- Behavior Aggregate: A collection of packets (on a link) with the same DSCP
- Per Hop Behavior (PHB): The description of the forwarding treatment to be applied to a behavior aggregate
- PHB Group: A set of one or more PHBs that can only be specified and implemented simultaneously because of a common constraint
- Traffic conditioner: An entity that applies some traffic control function (metering, marking, policing, shaping) to incoming packets

Diff-Serv Components

- Edge functions
  - Flow classification and packet marking
  - Traffic conditioning
- Core functions
  - Enforcement of Per Hop Behaviors
- Boundary functions
  - Conformance enforcement
- Components
  - Classifiers
    - Select packets and assigns DS code point
  - Traffic conditioners
    - Enforces rate limitations
  - Per Hop Behaviors
    - Differentiated packet treatments



More on DS Functions and Components



DS Standardization Status

- Assignment of Code Points in DS field (DSCP)
  - Space is partitioned in three pools

Pool	Codepoint Space	Assignment Policy
1	xxxxx0	Standards Action
2	xxxx11	EXP/LU
3	xxxx01	EXP/LU (*)

(*) may be utilized for future Standards Action allocations as necessary

- DSCP 000000 is the *recommended* value for the default PHB to be used for current best-effort traffic
- DSCP values xxx000 have been reserved as a set of *Class Selector Codepoints* to define up to 8 PHBs
  - Aimed as some backward compatibility with previous usage of IP precedence field, i.e., bits 0-2 of IPv4 TOS octet
  - DSCP 11x00 has preferential forwarding treatment over 000000
  - The 8 PHBs *must* yield at least to independent forwarding classes
  - Packet forwarding treatment *should* improve the higher the numerical value of the DSCP of a PHB

Expedited Forwarding (EF) PHB

- Status
  - Standardized in RFC 2598
  - Recommended DSCP is 10110
- Goals
  - Enable deployment of low loss, low latency, low jitter, assured bandwidth service
  - Emulation of a *virtual leased line*
- Definition of EF PHB
  - Minimum service rate *must* equal or exceed a configurable rate settable by a network administrator
  - Ability to ensure service *should* be independent of other traffic
  - Minimum service rate *should* average the configured rate over any time interval longer than the time it takes to send an MTU sized packet at the configured rate
- Requirements
  - Maximum rate of EF traffic *must* be limited if EF traffic can preempt other traffic
    - Ingress traffic conditioners discard (shape?) “excess” traffic

Sample Mechanisms for Supporting EF

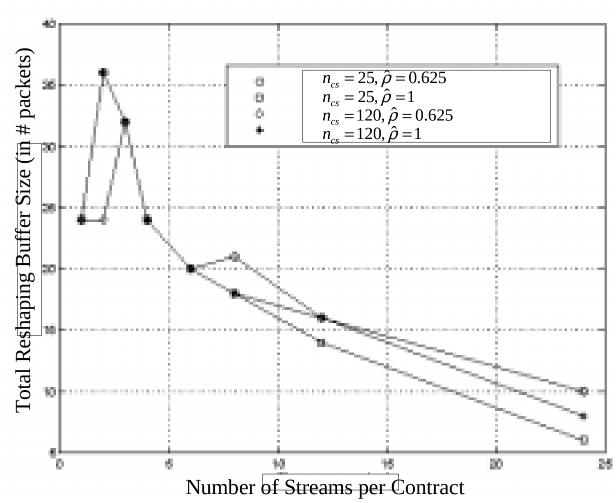
- Priority queue
  - EF traffic has precedence $\Rightarrow$ low loss, low latency, low jitter
  - Service rate is link speed whenever EF queue is non-empty
  - Ingress traffic policing is required to avoid starvation of other traffic classes
- Weighted Fair Queueing
  - EF is guaranteed a rate and delay bound
    - Low loss and low latency
    - Minimum rate is configured rate over time interval larger than one MTU transmission time
  - Potential for burstiness increase because of “aggregation”
    - Configure rate to be higher than incoming EF rate
      - How much higher?

Some Basic Issues with EF

- Impact of aggregation on
  - End-to-end performance
  - Policing functions at domain boundaries
- What mechanism?
  - Priority queue vs WFQ
  - Shaping at domain boundaries
- What EF load?
- Individual vs aggregate contracts
- Some preliminary answers (no deployment experience)
  - Reshaping is nearly a must to avoid substantial non-conformance dropping at domain boundaries
    - Aggregate contracts are more efficient in terms of both amount of buffering and reshaping delay
  - Aggregation requires low load to avoid potential increases in burstiness, delay variations, etc.

On The Efficiency of Aggregate Contracts

- *N* = 10 network hops
- n_{ts} = 24 tagged streams
- n_{cs} = # cross-streams
- Desired level of non-conformance P_D=10^{-5}



Aggregation reduces the *total* amount of buffering needed. Furthermore, higher speeds should also reduce reshaping delays

Assured Forwarding (AF) PHB

- Status: Standardized in RFC 2597
- Goals
  - Ensure high probability of packet delivery up to a committed rate
  - Support *excess* traffic with a lower probability of delivery
- Definition of AF PHB group
  - Four separate AF *classes* are currently defined
  - Each class is allocated its *own* amount of resources (buffer & bw)
  - Each class specifies three drop precedence values (DSCP)
    - Low drop precedence packets are *protected* from loss by preferentially discarding higher drop precedence packets
- Requirements
  - Two of more AF classes *must* not be aggregated together
  - A class *must* be allocated a configurable amount of resources and *should* achieve its rate over small and large time scales
  - Packet forwarding probability *must* be inversely proportional to drop precedence
  - A DS node *must* accept all three drop precedence values and *must* yield at least two levels of loss probability
  - A DS node *must not* reorder packets from the same microflow that belong to the same AF class

Specification of AF PHB Group

- Recommended values for AF codepoints

	AF1	AF2	AF3	AF4
Low drop prec.	001010	010010	011010	100010
Medium drop prec.	001100	010100	011100	100100
High drop prec.	001110	010110	011110	100110

- Each class, if supported, has its own resources
  - No specific performance relationship between classes
    - Performance depends on *relative* ratio between resources allocated to each class and traffic volume assigned to it
  - Traffic conditioning on ingress can provide desired packet marking
  - Note that fourth bit in DS field can be used for simple high/low priority identification
    - Implementation simplicity

Sample AF Implementations

- FIFO scheduler with buffer management
  - Assign buffer shares to each AF class in proportion to its committed rate
  - Specify thresholds within each class for discarding based on drop precedence
    - Deterministic or randomized a la RED
- WFQ scheduler with buffer management
  - Assign scheduler weight to each AF class in proportion to its committed rate
  - Provide per class buffer management, e.g., through the specification of drop precedence based thresholds
  - Smoother service characteristics of WFQ can lower likelihood of dropping high drop precedence packets

Diff-Serv Summary

- Standardization effort complete
  - Several proposed standard RFCs
- Initial deployments and vendor support
  - Internet2 QBone effort (`http://www.internet2.edu/qos/qbone`)
  - DS field based classification and prioritization becoming available from most router vendors
- Some ongoing/missing pieces
  - Signalling support
    - Bandwidth broker?
  - Sensitivity to route changes
    - Single change can impact many flows on unaffected links
  - Coexistence with other Internet technologies under development
    - Integrated services and MPLS
  - What end-to-end services can be built based on PHBs?

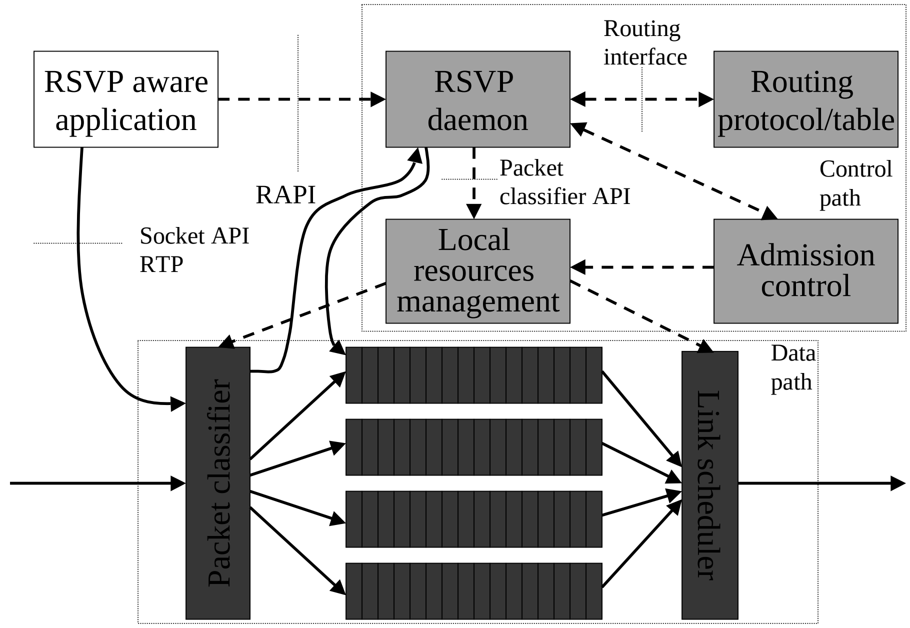
Integrated Services

- Goals and motivations
  - End-to-end guarantees for individual flows
    - From end-system to end-system
  - Range of service guarantees
    - From deterministic performance to loose bandwidth/delay guarantees
  - Tight coupling with signalling (RSVP)
- Status
  - Initial standardization effort complete
    - Two services have been standardized
      - Controlled Load Service
      - Guaranteed Service
    - Standardized signalling mechanism (RSVP)
  - Limited deployment experience
    - Scalability concern (especially in the backbone)
    - Few applications exist that can invoke the services (this is changing)
    - Issues regarding end-to-end availability

What is RSVP

- RSVP is a signalling protocol to request allocation of resources to a *flow* in an IP network
- Major characteristics of RSVP
  - Application initiated (fine *granularity* of reservation)
  - Designed for *scalability* to **very large** multicast group
    - Receiver oriented model, e.g., as with ATM LIJ
  - Reservations are for *simplex* flows
  - Support for *heterogeneous* reservations (multicast) and *renegotiation* of reservations
  - Allows *sharing* of reservations across multiple flows
  - *Soft state* approach for simple error recovery

Overview of RSVP Role and Interfaces



Some RSVP Definitions

- Session: set of packets addressed to a particular destination and transport protocol
- Flow descriptor: Flowspec + Filter spec
  - Flowspec: Reservation request (RSpec & TSpec)
  - Filter spec: Sender address and TCP/UDP port number
- RSVP flow: Session + Filter spec
- Reservation style: distinct/shared; explicit/implicit

Sender selection	Reservations: Distinct	Reservations: Shared
Explicit	Fixed-Filter (FF) style	Shared-Explicit (SE) style
Implicit	none defined	Wildcard-Filter (WF) style

RSVP Messages

- PATH: sets up state along path followed by packets
- RESV: Request for reservation back along setup path

- PATH_TEAR: Explicit removal of state along path
- RESV_TEAR: Explicit removal of reservation

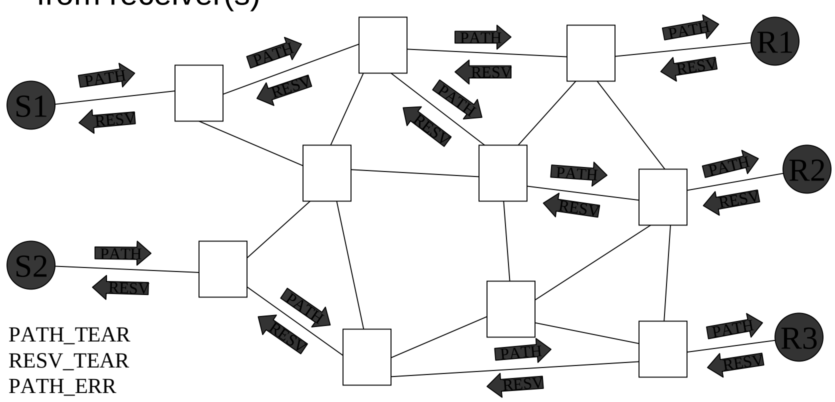
- RESV_ERR: Reservation failure & errors
- PATH_ERR: Path error

- RESV_CONFIRM: Reservation confirmation*

* Not an end-to-end guarantee

Basic RSVP Operation

- Setup path in the network through PATH messages from sender(s)
- Reserve resources on path through RESV messages from receiver(s)



PATH_TEAR
RESV_TEAR
PATH_ERR
RESV_ERR
RESV__CONFIRM

RSVP Soft State

- Soft state means that any RSVP related state (PATH and RESV states) is temporary (timer)
  - Needs to be *refreshed* in order to stay
  - Will (eventually) disappear if not refreshed
- Refresh and state clean-up
  - Send new message every *t* sec (30sec)
  - If fail to receive any refresh in *n**t* sec, delete state (*n=3)*
- Simplification of error recovery as you know you wont stay forever in a bad state

  ⟹ Don't need to be "too" picky in taking care of **all** error scenarios
- Trade-off between additional protocol overhead and simpler processing rules
  - RSVP messages are sent *unreliably*

RSVP PATH Message

- From sender to receiver(s) to establish path state in network elements (routers) between sender and receiver(s)
- Addressed directly to destination
  - Router alert option ensures interception at each RSVP hop
- Includes
  - Previous hop (PHOP): The previous RSVP aware entity on the path
  - Sender template: Filter spec for the sender
  - Sender TSpec: Traffic characteristics of sender
  - Sender ADSPEC: Used to capture path characteristics
- PATH message is processed and updated at each RSVP aware hop on the path
  - Create or refresh path state
  - Update ADSPEC

Format of Sender TSpec

31 ... 24	23 ... 16	15 ... 0
0	reserved	7
1	0 reserved	6
127	0	5
Token bucket rate [r] (32-bit IEEE floating point number)		
Token bucket size [b] (32-bit IEEE floating point number)		
Peak data rate [p] (32-bit IEEE floating point number)		
Minimum policed unit [m] (32-bit integer)		
Maximum packet size [M] (32-bit integer)		

0: Message format version number
7: Overall length (7 words without header)
1: Service header, service number 1 (default/global information)
6: Length of service 1 data (6 words without header)
127: Parameter ID, parameter 127 (Token_Bucket_TSpec)
0: Parameter 127 flags (none)
5: Parameter 127 length (5 words without header)

TSpec Conformance Rules

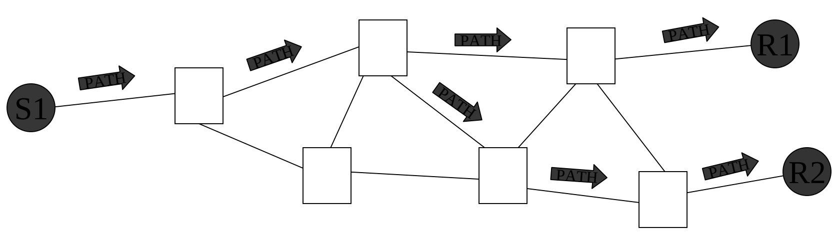
- Limitation on when and how much data to send
  - $\forall t \ \ A(t) <= M + \min[pt, rt+b-M]$
- Limitation on minimum packet size
  - Per packet processing time τ, e.g., lookup, must be smaller than packet transmission time
    $$\tau \leq \frac{m}{c}, c = \text{input link speed}$$
  - Packet smaller than *m* are counted as being of size *m*
    $\Rightarrow$ Lower effective rate if smaller packets are used
    $$m' \leq m \Rightarrow r' = r\frac{m'}{m}$$
- Limitation on maximum packet size (*M*)
  - Bound the maximum transmission time of packets
  - Packets larger than *M* are deemed non-conformant

Content of ADSPEC

- Two components
  - General parameters
  - Service specific parameters (absent ⇒ ineligible service) possible means for coordinating service selection between receivers
- General parameters
  - Global break bit
  - Hop count for Integrated Services network elements
  - Path bandwidth estimate
  - Minimum path latency
  - Path MTU
- Service specific parameters
  - Service break bit
  - Overrides of general parameters (MTU, bandwidth estimate)
  - Additional service specific quantities

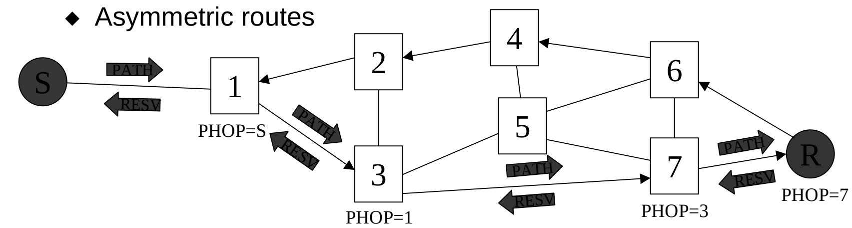
Processing of PATH Messages

- Generate PATH message (sender template, Tspec, ADSPEC)
- Query routing to identify NHOP(s) (forwarding)
- PATH state processing
  - Create PATH state (TSpec, PHOP, NHOP, etc.) if not present
  - Update ADSPEC (hop count, MTU, latency, bandwidth, etc.)
  - Refresh state if PATH state present
  - Send immediate refresh if state changed

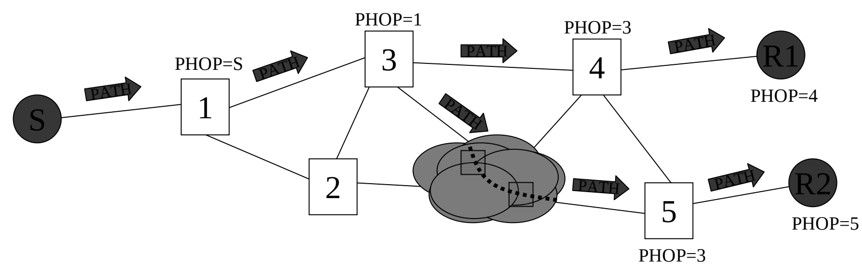


Why Do We Need a PHOP?

- Multiple reasons
  - Asymmetric routes



  - Non-IS hops/clouds (tunneling)



One Pass With Advertising (OPWA)

- Benefits
  - Discover characteristics of path before making reservation, i.e., ADSPEC
    - Improves odds of successful reservation (bandwidth estimates)
    - Hints to receiver for improved configuration (playback buffer based on latency estimate)
    - Can compensate for path characteristics during reservation phase (Guaranteed Service queueing delay adjustment)
- Disadvantage
  - Path selection (routing) done without knowing exact service requirements

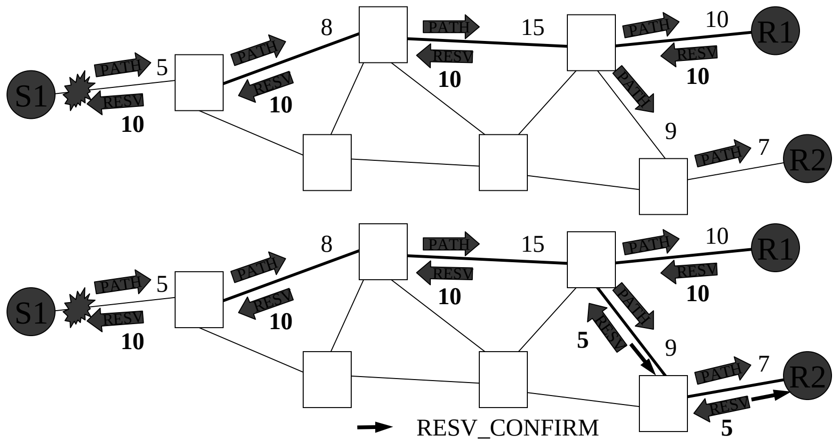
RSVP RESV Message

- From receiver to sender(s) to reserve resources
- Sent hop-by-hop using PHOP information
- Includes
  - NHOP, i.e., where it came from (interface address)
  - Reservation style (FF, SE, WF) and scope object if needed
    - Scope object: List of senders to which **implicit** reservation applies
  - Reservations style and flow descriptor list
    - Sender(s) to which reservation applies (filter spec's)
    - Flowspec (service specific parameters)
      - RSpec, i.e., QoS specific requirements
      - TSpec, i.e., sender traffic to which reservation applies
  - RESV_CONFIRM (optional)
- RESV message processing at each hop
  - Create or modify RESV state
  - Merging of RESV message (avoids RESV implosion)
  - Forward upstream (PHOPs) after successful reservation

Processing of RESV Messages

- Generate RESV message (Flowspec, Filter spec)
- Query admission control (new/modified reservation)
- RESV state processing
  - Create RESV state (Flowspec, filter spec, etc.) if not present
  - Coordinate merging of Flowspec's (from multiple RESV)
  - Refresh state if RESV state present
    - Send immediate refresh if state changed
  - Send RESV_CONFIRM if necessary
    - End point or larger reservation already in place
  - Determine where to propagate RESV messages
    - Filter spec (and scope object if necessary)
    - list of PHOPs
  - Blockade state (avoidance of "killer" reservation)

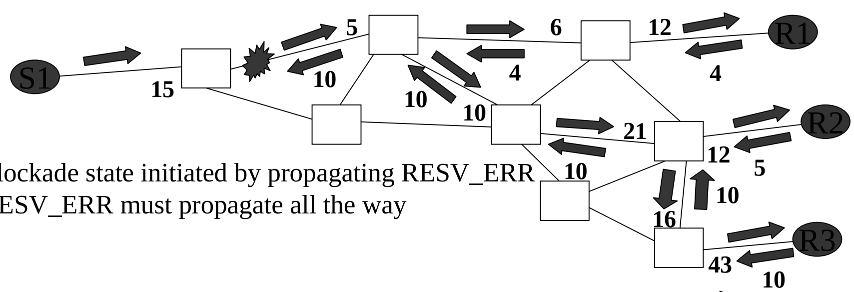
Why is RESV_CONFIRM Not An End-to-End Confirmation?



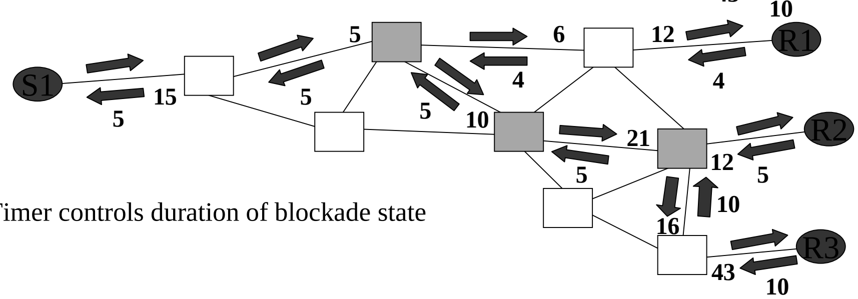
- RESV_CONFIRM sent if larger reservation already in place at merge point
  - No guarantee that reservation is in place end-to-end
- Blockade state will eventually allow end-to-end reservation, but after additional delay

Killer Reservation Problem

- Large unsuccessful reservation prevents smaller one(s)
- Blockade state installed to block offending reservation

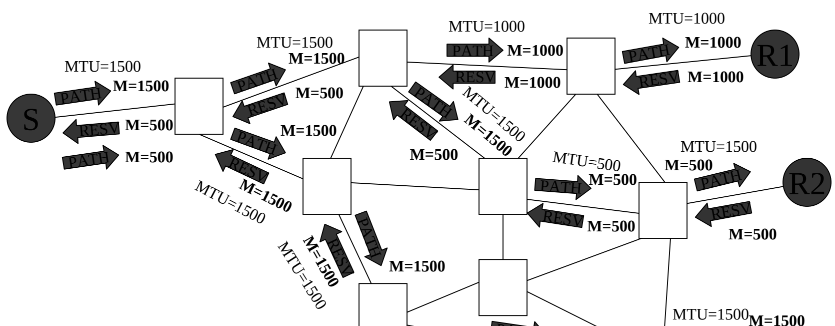


Blockade state initiated by propagating RESV_ERR
RESV_ERR must propagate all the way



Timer controls duration of blockade state

RSVP as an MTU Discovery and Traffic Negotiation Mechanism



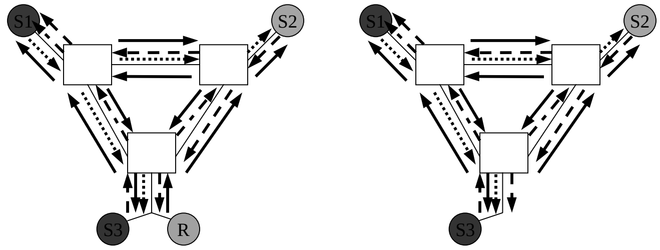
- MTU discovery
  - PATH: Propagate min(M,MTU)
  - RESV: Propagate M=min(Mi)
- Traffic negotiation
  - Receiver specifies desired TSpec, e.g., smaller *b* and larger *r*
- Some dangers
  - Lowest common denominator rules...

Merging of Flowspec and Sender TSpec

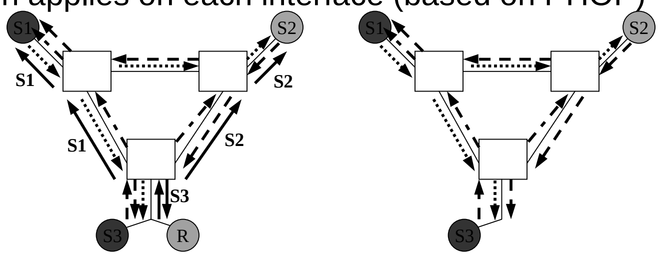
- Two types of merging operations
  - Least Upper Bound (LUB): As "*good*" as each Flowspec for any individual parameter (merging of reservations)
    - Flowspec TSpec1: r1=1000, b1=3500, p1=10,000, m1=64, M1=1500
    - Flowspec TSpec2: r2=800, b2=5000, p2=10,000, m1=32, M2=1000
    - Merged TSpec : r=1000, b=5000, p1=10,000, m=32, M=1000
  - Greatest Lower Bound (GLB): As big a reservation as *needed* given aggregate traffic (comparing reservation to sum of sender traffic)
    - Sender_TSpec: r1=1000, b1=3500, p1=10,000, m1=64, M1=1500
    - Flowspec_TSpec: r2=800, b2=5000, p2=10,000, m1=32, M2=1000
    - Merged_TSpec : r=800, b=3500, p1=10,000, m=64, M=1000

Avoiding Looping with Wildcard Filters

- Wildcard filter introduce the possibility of *auto-refresh* loops



- Scope object specifically identifies senders to which reservation applies on each interface (based on PHOP)



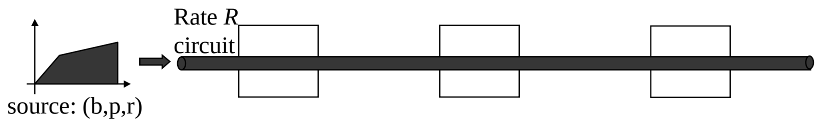
Controlled Load (CL) Service

- Input traffic defined by token bucket
  - Conformant packets are eligible for service guarantees
  - Non-conformant packets are treated as best effort packets
- Conformant packets see delay and losses equivalent to what they would *experience* in an unloaded network
- Implementing the CL service guarantee requires
  - Dedicated resources for CL flows (data path mechanism)
  - Limit on the number of CL flows based on available resources (control path mechanism)
- Issue
  - Difficulty in identifying non-conformant packets inside the network (no packet marking ability as in ATM to "carry" the result of the token bucket computations)
  - Efficient admission control algorithm to control the number of accepted flows

Guaranteed Service (GS)

- Service goals
  - Emulate the service achievable through a dedicated line
    - Challenge is to achieve it over a shared packet network
  - Provide deterministic service guarantees
    - Zero (congestion) packet losses and hard end-to-end delay bound
- Traffic descriptor in the form of a token bucket
- Service guarantees of a dedicated rate (*R*) circuit

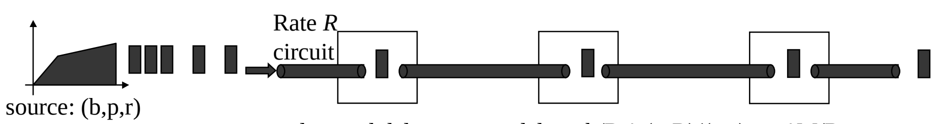
end-to-end delay = prop. delay +b/R * (p-R)/(p-r)



Max time at peak rate: t0 = b/(p-r) ⇒ pb(p-r) is amount received by t0
Amount transmitted by t0 is Rb/(p-r) ⇒ b(p-R)/(p-r) is amount left by t0
⇒ Max access delay is b/R * (p-R)/(p-r)

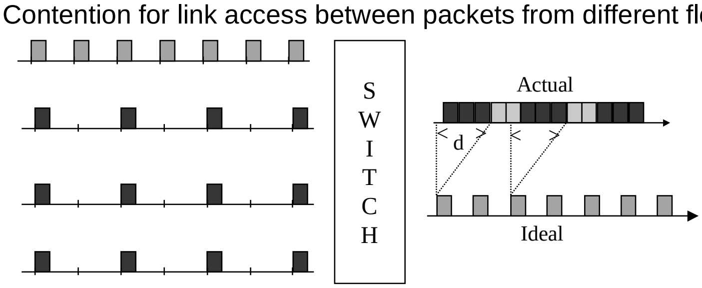
Emulating a Dedicated Rate Circuit

- Impact of packet based transmissions
  - Packetization delay at each hop (*n* hops)



end-to-end delay = prop. delay +b/R * (p-R)/(p-r) + *n**M/R

  - Contention for link access between packets from different flows



    - Added delay component at each hop
    - How does it affect delay at *next* hop

The Guaranteed Service Model

- Basic idea is to characterize the divergence from a perfect *fluid* (fixed rate circuit) model at each hop
  - A rate dependent *error term (C)*
  - A rate independent *error term (D)*
- Error terms are specific to the scheduling mechanism used by each router, but are *additive*
  - Error terms accumulated in ADSPEC (RSVP PATH)
  - End-to-end delay expression

$$\Delta = \left[\frac{(b-M)}{R} \times \frac{(p-R)}{(p-r)} \right] + \frac{(M + C_{tot})}{R} + D_{tot}, \; p > R \geq r$$

  - C_{tot} and D_{tot} correspond to accumulated values of C and D error terms

The Guaranteed Service Model - contd.

- Error terms also provide information to compute buffer requirements at each hop
  - Initial buffer requirement is $\frac{(b-M)}{R} \times \frac{(p-R)}{(p-r)} + \frac{M}{R}$
  - $C_{tot}/R + D_{tot}$ gives the maximum amount of time a packet can have been held up, so that during that time additional data can have been accumulated (increased burst size)
    - Start with $(b-M) \times \frac{(p-R)}{(p-r)} + M$
    - Need to add $\left(\frac{C_{tot}}{R} + D_{tot} \right) \times r$ to account for possible additional accumulation
  - Note: Some adjustments needed to handle different sub-cases

Guaranteed Service Usage (with RSVP)

- Sender advertises TSpec in PATH message
- Router updates ADSPEC based on characteristics of local scheduler (add C and D error terms to C_{tot} and D_{tot})
- Receiver determines reservation rate R needed to satisfy end-to-end delay requirements Δ

$$R = \frac{(b-M)p + (M + C_{tot})(p-r)}{(p-r)(\Delta - D_{tot}) + (b-M)}$$

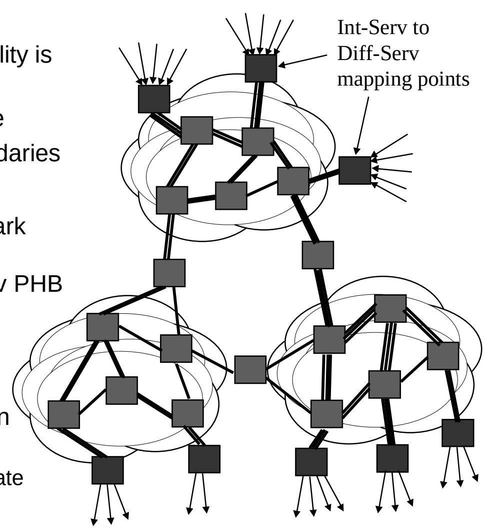
- Routers allocate requested rate (if available) and determine necessary buffer allocation for zero loss
  - Note: Reshaping at a node "resets" the original traffic envelope and, therefore, lowers buffer requirements at downstream nodes. Add only error terms from reshaping point on

$$\left(\frac{C_{sum}}{R} + D_{sum}\right) \times r$$

Note: Reshaping does not affect the end-to-end delay bound (if you are reshaped, it's because you were early)

Int-Serv/Diff-Serv Interactions

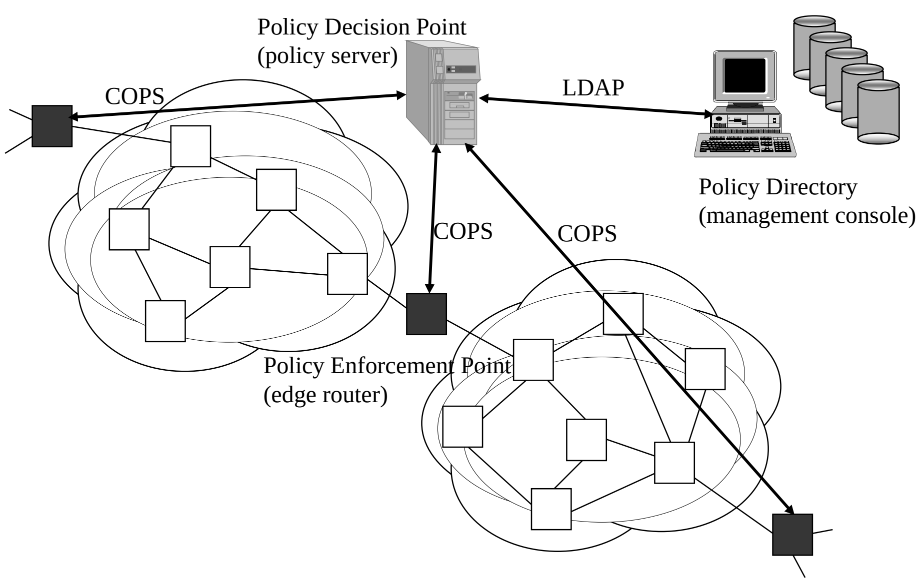
- Basic premises
  - Int-Serv where data path scalability is not an issue
  - Diff-Serv support in network core
  - Mapping points at network boundaries
- Data path aggregation
  - Perform in gress policing and mark packets from Int-Serv flows with DSCP of corresponding Diff-Serv PHB
    - draft-ietf-issll-ds-map-00.txt
- Control Path aggregation
  - Aggregate RSVP signalling establishes reservations between ingress and egress
    - New session and sender_template objects
    - draft-ietf-issll-rsvp-aggr-02.txt



Controlling Access to QoS

- If you offer different services you need mechanisms to verify who can access them
- *Policy* encompasses a set of solutions to this problem
  - Policy specification
    - Language (schema) to describe policy rules
    - Protocols to distribute/download policy rules
  - Policy decision
    - Receiving and processing service requests
    - Controlling and monitoring access to resources
      - Time dependent decisions
  - Policy enforcement
    - Intercepting service requests
    - Implementing policy decisions

Policy Architecture Overview



Summary

- Internet QoS is becoming a reality
  - The technology is there
  - User demand is there
  - Standards are there
- But the future remains blurred
  - Multiple competing solutions
    - Diff-Serv vs Int-Serv
    - Diff-Serv vs MPLS+traffic engineering
  - Some missing pieces
    - Diff-Serv signalling and service definitions
    - Coupling to routing protocols
    - Interactions between technologies
- The outcome will be driven by deployment issues
  - Standardized policy support
  - Interactions between providers (peering agreements)

References

- Differentiated Services
  - Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers: rfc 2474
  - An Architecture for Differentiated Services: rfc 2475
  - Assured Forwarding PHB Group: rfc 2597
  - An Expedited Forwarding PHB: rfc 2598
  - A Two-bit Differentiated Services Architecture for the Internet: rfc 2638
  - A Single Rate Three Color Marker: rfc 2697 (informational)
  - A Two Rate Three Color Marker: rfc 2698 (informational)
  - Internet Drafts
    - Management Information Base for the Differentiated Services Architecture

References, contd.

- RSVP (and other signalling protocols)
  - Resource ReSerVation Protocol (RSVP): rfc 2205
  - Resource ReSerVation Protocol (RSVP) Version 1 Message Processing Rules: rfc 2209
  - Resource ReSerVation Protocol (RSVP) Version 1 Applicability Statement Some Guidelines on Deployment: rfc 2208
  - The Use of RSVP with IETF Integrated Services: rfc 2210
  - RSVP over ATM Implementation Requirements: rfc 2380
- Integrated Services
  - Specification of the Controlled-Load Network Element Service: rfc 2211
  - Specification of Guaranteed Quality of Service: rfc 2212
  - General Characterization Parameters for Integrated Service Network Elements: rfc 2215
  - Network Element Service Specification Template: rfc 2216 I
  - nteroperation of Controlled-Load Service and Guaranteed Service with ATM: rfc 2381

References, contd.

- Routing
  - A Framework for QoS-based Routing in the Internet: rfc 2386
  - QoS Routing Mechanisms and OSPF Extensions: rfc 2676
  - The OSPF Opaque LSA Option: rfc 2370
  - Internet Drafts
    - OSPF Optimized Multipath (OSPF-OMP)
    - IS-IS Optimized Multipath (ISIS-OMP)
    - IS-IS extensions for Traffic Engineering
- MPLS
  - Requirements for Traffic Engineering Over MPLS: rfc 2702
  - Internet Drafts
    - Extensions to RSVP for LSP Tunnels
    - Constraint-Based LSP Setup using LDP
    - MPLS Traffic Engineering Management Information Base Using SMIv2
    - MPLS Support of Differentiated Services
    - A Proposal to Incorporate ECN in MPLS
    - Applicability Statement for Extensions to RSVP for LSP-Tunnels

References, contd.

- Policy
  - A framework for policy-based admission control: rfc 2753
  - The COPS (Common Open Policy Service) protocol: rfc 2748
  - COPS usage for RSVP: rfc 2749
  - Signaled preemption priority policy element: rfc 2751
  - Identity representation for RSVP: rfc 2752
  - Multiple drafts on policy schemas and frameworks
- Integrated Services over Specific Link Layers
  - RSVP over ATM Implementation Requirements: rfc 2380
  - RSVP over ATM Implementation Guidelines: rfc 2379
  - Interoperation of Controlled-Load Service and Guaranteed Service with ATM: rfc 2381
  - A Framework for Integrated Services and RSVP over ATM: rfc 2382
  - Internet Drafts
    - SBM (Subnet Bandwidth Manager): A Protocol for RSVP-based Admission Control over IEEE 802-style networks
    - Integrated Service Mappings on IEEE 802 Networks
    - A Framework For Integrated Services Operation Over Diffserv Networks
    - Aggregation of RSVP for IPv4 and IPv6 Reservations