Flow and Congestion Control

Based on A. Mediano's lecture notes

Flow Control

- Flow control: end-to-end mechanism for regulating traffic between source and destination
- Congestion control: Mechanism used by the network to limit congestion
- The two are not really separable, and we will refer to both as flow control
- In either case, both amount to mechanisms for limiting the amount of traffic entering the network
- Sometimes the load is more than the network can handle

Flow Control Absent

- **When overload occurs**
  - **queues build up**
  - **packets are discarded**
  - **Sources retransmit messages**
  - **congestion increases => instability**
- **Flow control prevents network instability by keeping packets waiting outside the network rather than in queues inside the network**
  - **Avoids wasting network resources**
  - **Prevent disasters**

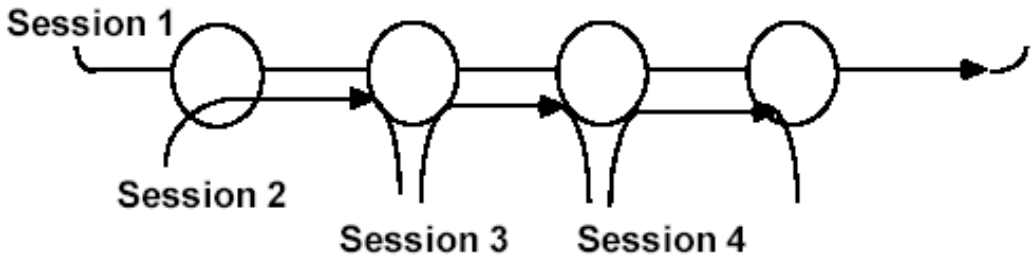
Means of Flow Control

- Call blocking
  - Flow admission control
- Packet discarding
  - If a packet has to be discarded anyway, it might as well be discarded as early as possible to avoid wasting additional network resources unnecessarily
- Packet scheduling
  - A network node can selectively expedite or delay of the transmission of certain packets

Goals of Flow Control

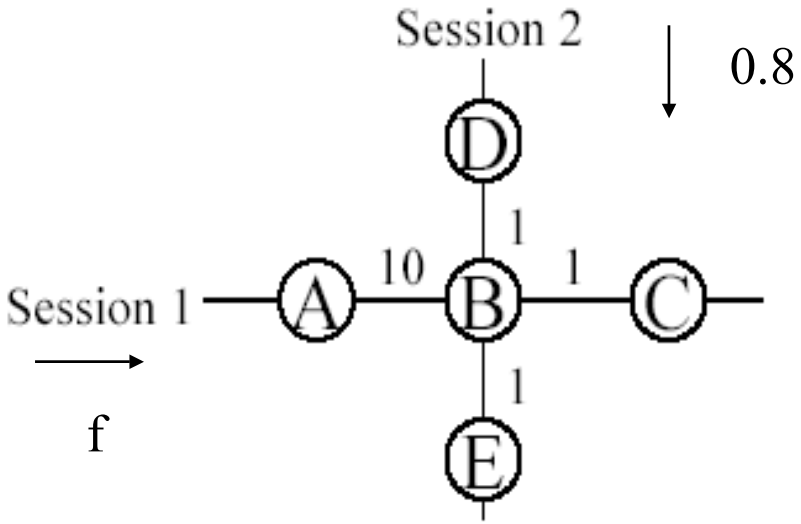
- Maximize network throughput
- Reduce network delays
- Maintain quality-of-service parameters
- Fairness, delay, etc..
- Tradeoff between fairness, delay, throughput…

Fairness



- **If link capacities are each 1 unit, then**
  - **Maximum throughput is achieved by giving short session one unit and zero units to the long session; total throughput of 3 units**
  - **One concept of fairness would give each user 1/2 unit; total throughput of 2 units**
  - **Alternatively, giving equal resources to each session would give single link users 3/4 each, and 1/4 unit to the long session**
- **Generally a compromise is required between equal access to a network resource and throttling those sessions that are most responsible for using that resource**

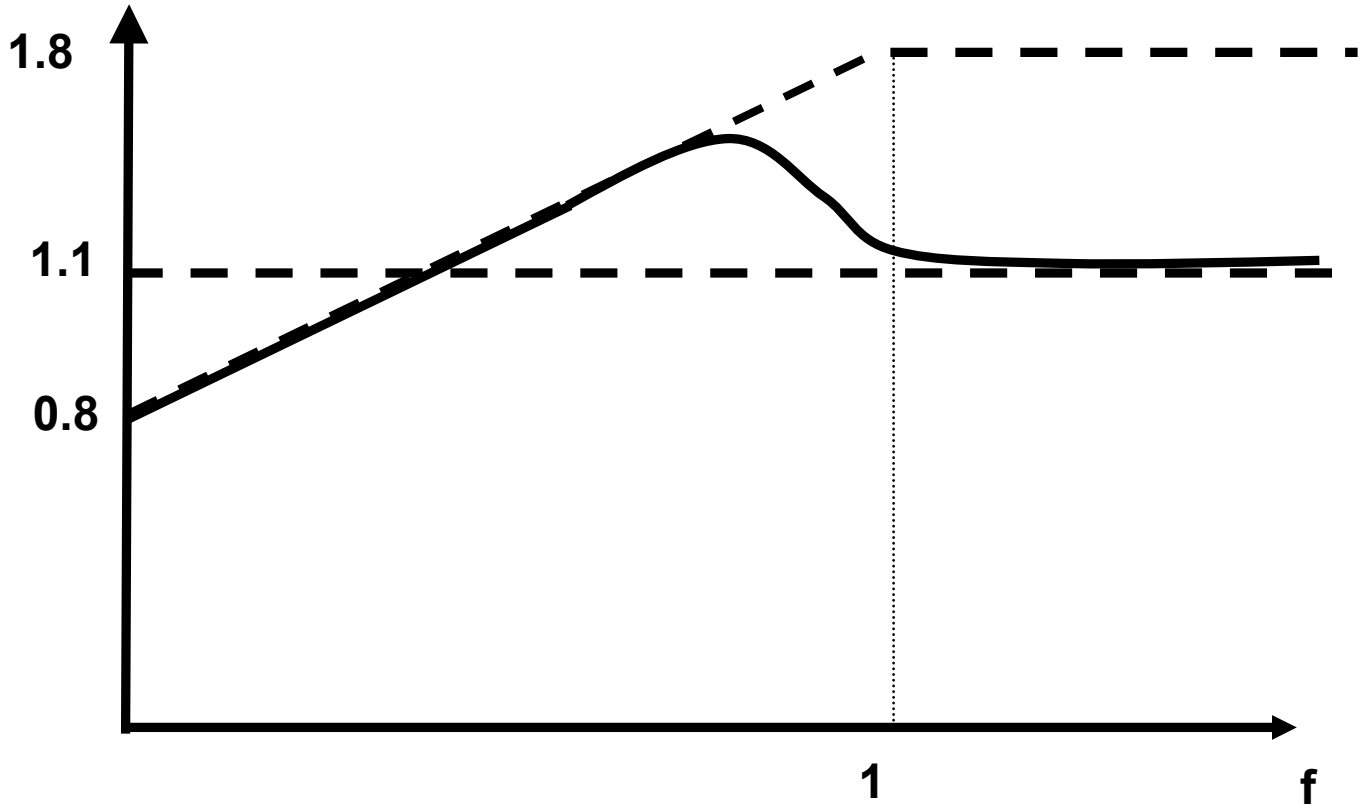
Limiting Buffer Overflow



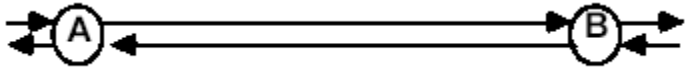
- **Clearly both sessions are limited to 1 unit of traffic**
- **Without flow control, session 1 can dominate the buffer at node B**
  - **Since 10 session 1 packets arrive for each session 2 packet, 10/11 packets in the buffer will belong to session 1**

Example

Total Throughput

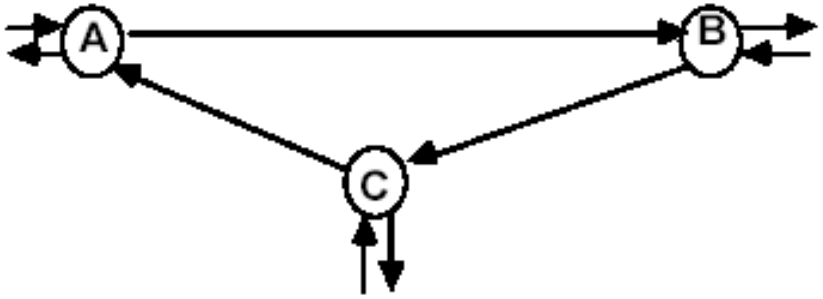


Deadlocks



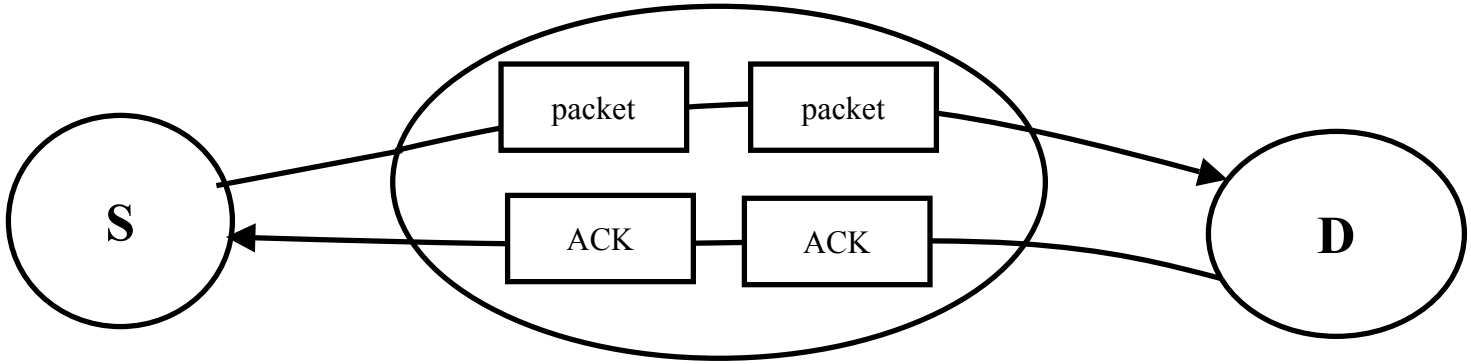
If buffers at A fill up with traffic to B and vice versa, then A can not accept any traffic from B, and vice versa causing deadlock

- **A cannot accept any traffic from B**
- **B cannot accept any traffic from A**



A can be full of B traffic, B of C traffic, and C of A traffic.

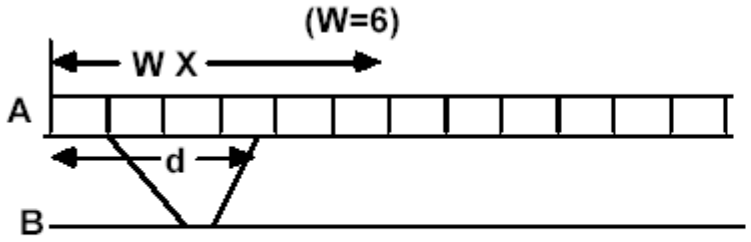
Window Flow Control



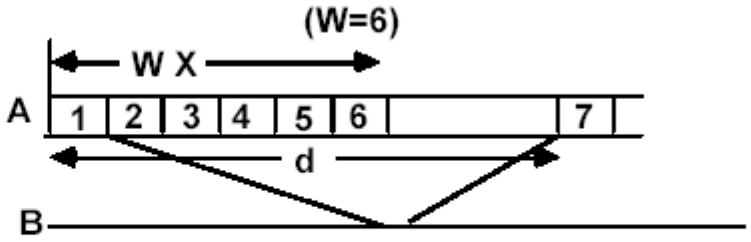
- **Similar to Window based ARQ**
  - **End-to-end window for each session, Wsd**
  - **Each packet is ACK'd by receiver**
  - **Total number of un-ACK'd packets <= Wsd**
  - **Window size is an upper-bound on the total number of packets and ACKs in the network**
  - **Limit on the amount of buffering needed inside network**

End-to-end Flow Control

- Let x be expected packet transmission time, W be size of window, and d be the total round trip delay for a packet
  - Ideally, flow control would only be active during times of congestion
  - Therefore, Wx should be large relative to the total round trip delay d in the absence of congestion If d <= Wx, flow control in-active and session rate r = 1/x If d > Wx, flow control active and session rate r = W/d packets per second

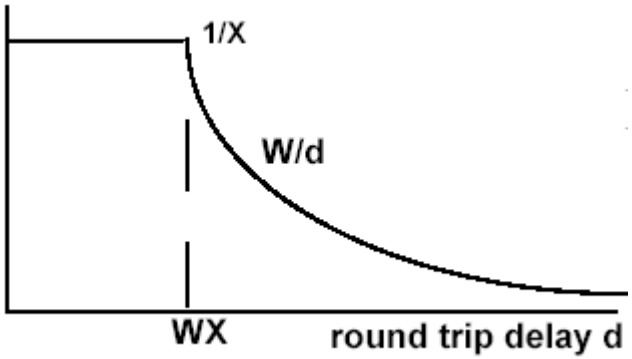


Flow control not active



Flow control active

End-to-end Flow Control



R = min {1/x, W/d} packets/second

- **As d increases, flow control becomes active and limits the transmission rate**
- **As congestion is alleviated, d will decrease and r will go back up**
- **Flow control has the affect of stabilizing delays in the network**

Choice of Window Size

- **Without congestion, window should be large enough to allow transmission at full rate of 1/x packets per second**
  - **Let d' = the round-trip delay when there is no queueing**
  - **Let N = number of nodes along the path**
  - **Let Dp = the propagation delay along the path**
  - **d' = 2Nx + 2 Dp (delay for sending packet and ack along N links)**
  - **Wx > d' => W > 2N + Dp/x**
  - **When Dp < x, W ~ 2N (window size is independent of prop. Delay)**
  - **When Dp >> Nx, W ~ 2Dp/x (window size is independent on path length)**

Impact of Congestion

- **Without congestion d = d'**
- **With congestion d > d'**
- **Problem:**
- **When d' is large (e.g., Dp is large) queueing delay is smaller than propagation delay and hence it becomes difficult to control congestion**
  - **increased queueing delay has a small impact on d and hence a small impact on the rate r**

Problems with Window Control

- **Window size must change with congestion level Difficult to guarantee delays or data rate to a session**
- **For high speed sessions on high speed networks, windows must be very large**
  - **E.g., for 1 Gbps cross country each window must exceed 60Mb**
  - **Window flow control becomes in-effective**
  - **Large windows require a lot of buffering in the network**
- **Sessions on long paths with large windows are better treated than short path sessions. At a congestion point, large window fills up buffer and hogs service (unless round robin service used)**

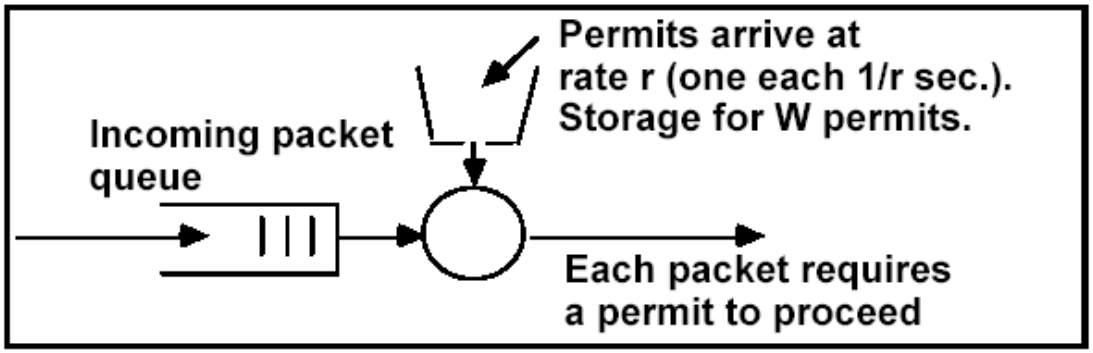
Node-by-node Window Control

- **Separate window (w) for each link along the sessions path**
  - **Buffer of size w at each node**
- **An ACK is returned on one link when a packet is released to the next link**
  - **buffer will never overflow**
- **If one link becomes congested, packets remain in queue and ACKs don't go back on previous link, which would in-turn also become congested and stop sending ACKs (back pressure)**
  - **Buffers will fill-up at successive nodes**
    - » **Under congestion, packets are spread out evenly on path rather than accumulated at congestion point**
- **In high-speed networks this still requires large windows and hence large buffers at each node**

Rate Control

- **Window flow control cannot guarantee rate or delay**
- **Requires large windows for high (delay * rate) links**
- **Rate control schemes provide a user a guaranteed rate and some limited ability to exceed that rate**
  - **Strict implementation: for a rate of r packets per second allow exactly one packet every 1/r seconds**
    - » **TDMA => inefficient for bursty traffic**
  - **Less-strict implementation: Allow W packets every W/r second**
    - » **Average rate remains the same but bursts of up to W packets are allowed**
    - » **Typically implemented using a "leaky bucket" scheme**

Leaky Bucket



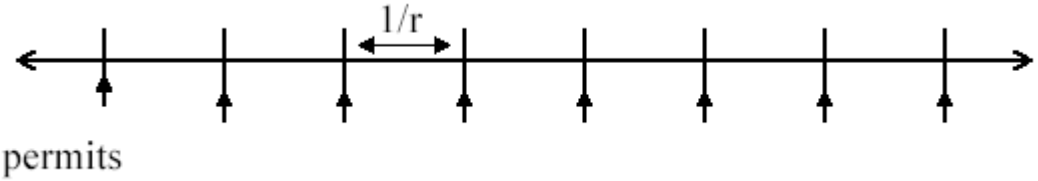
Session bucket holds W permits
- In order to enter the network, packet must first get a permit
- Bucket gets new permits at a rate of one every 1/r seconds

When the bucket is full, a burst of up to W packets can enter the network
- The parameter W specifies how bursty the source can be
    - Small W => strict rate control
    - Large W allows for larger bursts
- r specifies the maximum long term rate
- An inactive session will earn permits so that it can burst later

Leaky Bucket Flow Control

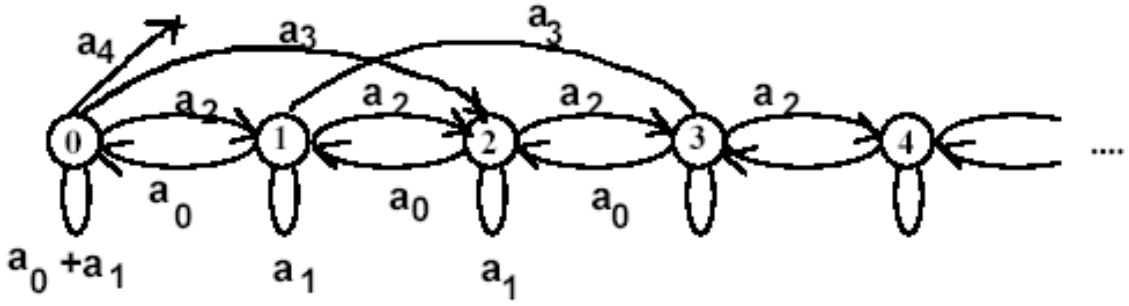
- **Leaky bucket is a traffic shaping mechanism**
- **Flow control schemes can adjust the values of W and r in response to congestion**
  - **E.g., ATM networks use RM (resource management) cells to tell sources to adjust their rates based on congestion**

Queueing Model



- **Slotted time system with a state change each 1/r seconds**
  - **A permit arrives at start of slot and is discarded if the bucket is full**
  - **Packets arrive according to a Poisson process of rate λ**
  - **a_i = Prob(i arrivals) = $(\lambda/r)^i e^{-\lambda/r} / i!$**
  - **P = number of packets waiting in the buffer for a permit**
  - **B = number of permits in the buffer**
  - **W = bucket size**
- **State of system: K = W + P - B**
  - **State represents the "permit deficit" and is equal to the number of permits needed in order to refill the bucket**
    - » **State 0 => bucket full of permits State W => no permits in buffer**
    - » **State W + j => j packets waiting for a permit**

Queueing Analysis



- **Note that this is the same as M/D/1 with slotted service**
  - **In steady-state the arrival rate of packets is equal to the arrival rate of permits (permits are discarded when bucket full, permits don't arrive in state 0 when no packets arrive)**

$$\Rightarrow \lambda = (1 - P(0)a_o)r, \quad \Rightarrow P(0) = (r-\lambda)/(a_0 r)$$

Queueing Analysis

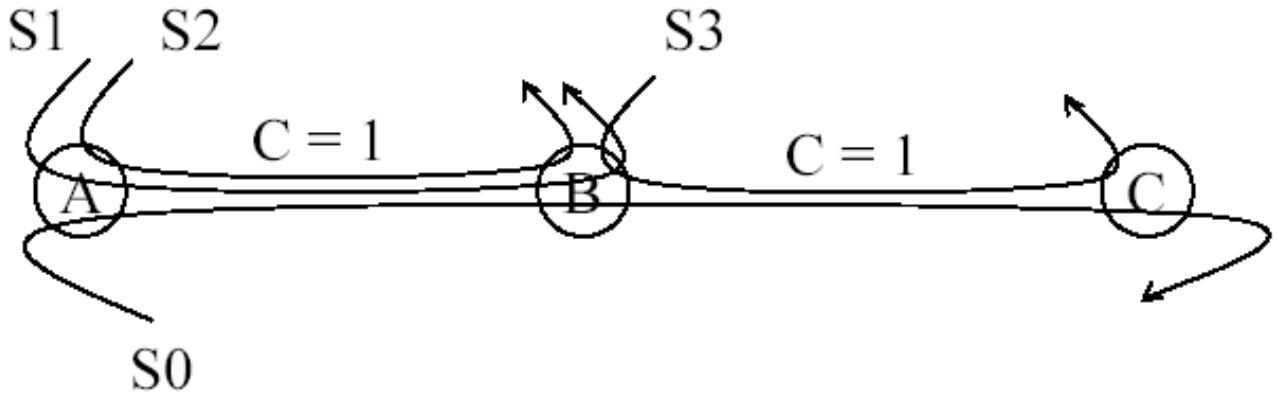
- **Now from global balance eqns:**
  - » **P(0) [1-a0 -a1] = a0 P(1)**
  - » **P(1) = [(1-a0-a1)/a0]P(0) => can solve for P(1) in terms of P(0)**
  - » **P(1)[1-a1] = a2P(0) + a0P(2) => obtain P(2) in terms of P(1)**
  - » **Recursively solve for all P(i)'s**
- **Average delay to obtain a permit**

$$T = \left[\sum_{j=W+1}^{\infty} (j - W) P(j) \right] \frac{1}{r}$$

How do you allocate the rate *r*

- **How do we decide on the rate allocated to a session?**
- **Approaches**
  - **Optimal routing and flow control**
    - » **Tradeoff between delay and throughput**
  - **Max-Min fairness**
    - » **Fair allocation of resources**
  - **Contract based**
    - » **Rate negotiated for a price (e.g., Guaranteed rate, etc.)**

Max-min fairness



- **Sessions S0, S1, S2 share link AB and each gets a fair share of 1/3**
- **Sessions S3 and S0 share link BC, but since session S0 is limited to 1/3 by link AB, session S3 can be allocated a rate of 2/3**

Max-min fairness

- **The basic idea behind max-min fairness is to allocate each session the maximum possible rate subject to the constraint that increasing one session's rate should not come at the expense of another session whose allocated rate is not greater than the given session whose rate is being increased**
  - **i.e, if increasing a session's rate comes at the expense of another session that already has a lower rate, don't!**

- **Given a set of session requests P and an associated set of rates RP, RP is max-min fair if,**
  - **For each session p, rp cannot be increased without decreasing r for some session p' which rp' <= rp**

Max-min fairness

- Let r_p be the allocated rate for session p, and consider a link a with capacity C_a
- The flow on link a is given by: $$F_a = \sum_{\substack{\forall p \; cross \sin g \\ link \; a}} r_p$$

- A rate vector R is feasible if:
  - $R_p >= 0$ for all p in P (all session requests) and
  - $F_a <= C_a$ for all a in A (where A is the set of links)

- R is max-min fair if it is feasible <u>and</u>

  For all p, if there exists a feasible R^1 such that $r_p < r^1_p$

  Then there exists a session p' such that $r_{p'} > r^1_{p'}$ and $r_{p'} <= r_p$

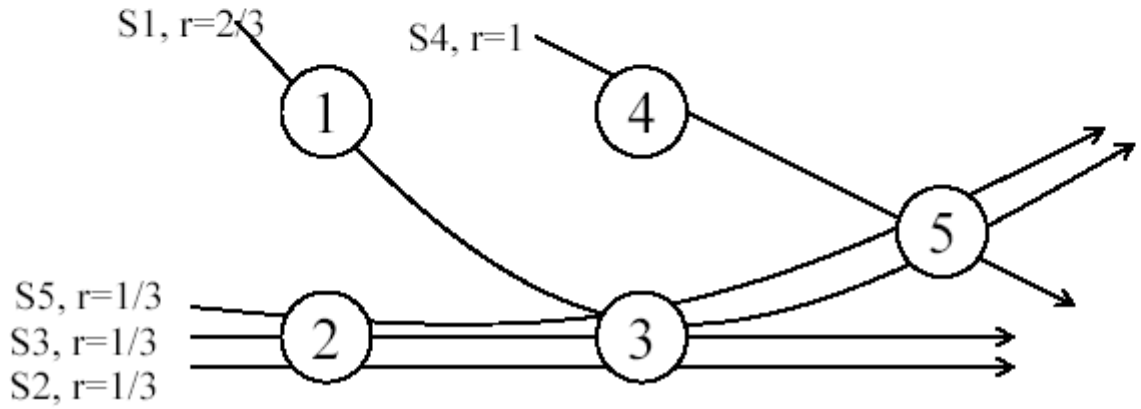
- In other words, you can only increase the rate of a path by decreasing the rate of another path that has been allocated no more capacity

Max-min Fairness

- Given a rate vector R, a link 'a' is a bottleneck link for session p if:
  - $F_a = C_a$ and $r_p >= r_{p'}$ for all sessions p' crossing link 'a'
  - Notice that all other sessions must have some other bottleneck link for otherwise their rate could be increased on link 'a'

- Proposition: A feasible rate vector R is max-min fair if and only if each session has a bottleneck link with respect to R

- Example (C=1 for all links)



Bottleneck links

S1 <=> (3,5)
S2,S3,S5 <=> (2,3)
S4 <=> (4,5)

Max-min fair algorithm

Start all sessions with a zero rate

Increment all session rates equally by some small amount δ

- Continue to increment until some link reaches capacity ($F_a = C_a$)
  - All sessions sharing that link have equal rates
  - Link is a bottleneck link with respect to those sessions
  - Stop increasing rates for those sessions (that is their Max-Min allocation)
- Continue to increment the rate for all other sessions that have not yet arrived at a bottleneck link
  - Until another bottleneck link is found
- Algorithm terminates when all sessions have a bottleneck link

In practice sessions are not known in advance and computing rates in advance is not practical

Outline

- Introduction
  - Multipath Traffic Engineering (TE)
  - Problem Statement
- Proposed Multipath-TE Architecture
  - Establishment of LSPs
  - MPLS Queueing
  - MPLS Feedback Mechanism
  - Flow-based Splitting
- Simulation Results
- Conclusions and Future Work

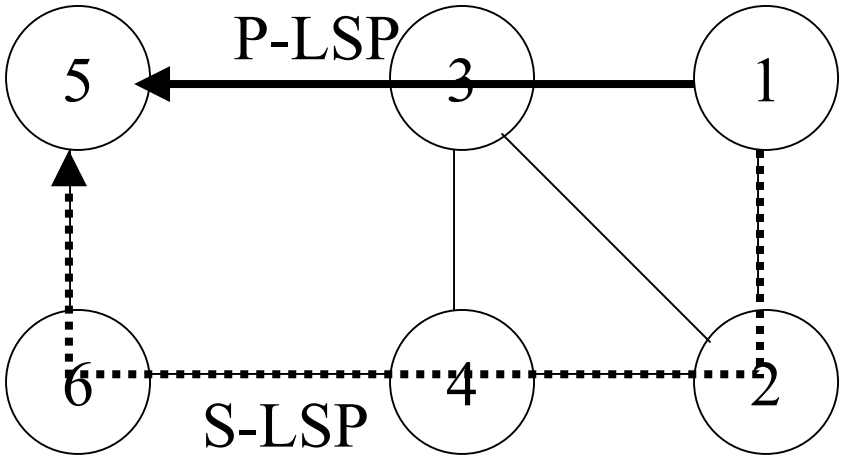
Multipath TE

- Multipath TE aims to exploit resources of the underlying physical network by splitting total traffic among multiple paths between source-destination pairs to enhance the performance of an operational network
  - Load Balancing. It could however lead to additional propagation delay if the alternate routes are poorly chosen
  - Knock-on effect. Using alternate paths by some sources force other sources whose minhop paths share links with these alternate paths to also use alternate paths ➔ performance degradation if the alternate paths use more resources (hops)
  - Distributed Algorithms. We avoid centralized algorithms that rely on the availability of a traffic matrix
  - Out-of-order packet delivery. It would be desirable to keep packet orders within a flow.

Proposed Multipath TE

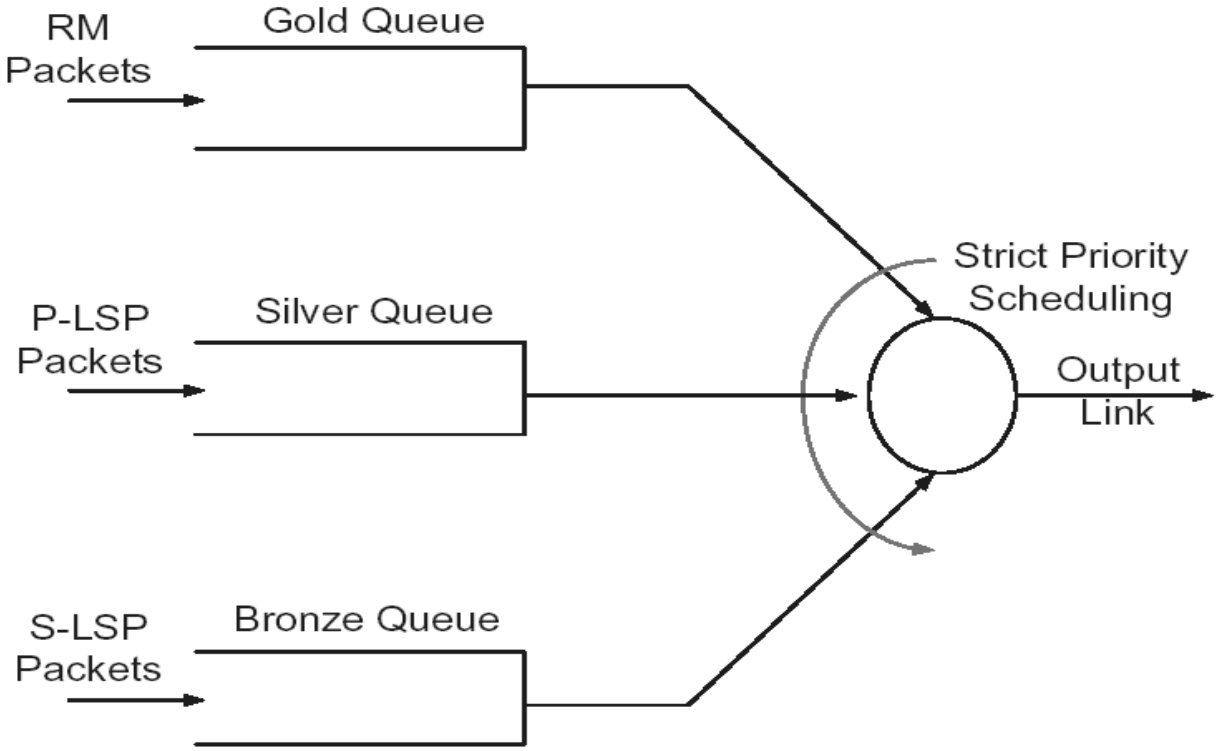
- We propose a novel multipath TE architecture for best-effort IP networks with MPLS backbones
- Our main goal is to increase the total amount of carried traffic by using two paths although the ideas are extendible to more than two paths
- No a-priori information on the traffic matrix required
- Effective in short and long time scales
- Knock-on effect is eliminated
- Flow-based routing ensures that out of order packet arrivals would not take place
- We provide network-wide simulation studies for moderately sized networks
- We assume UDP transport and TCP is left for future research

Establishment of LSPs



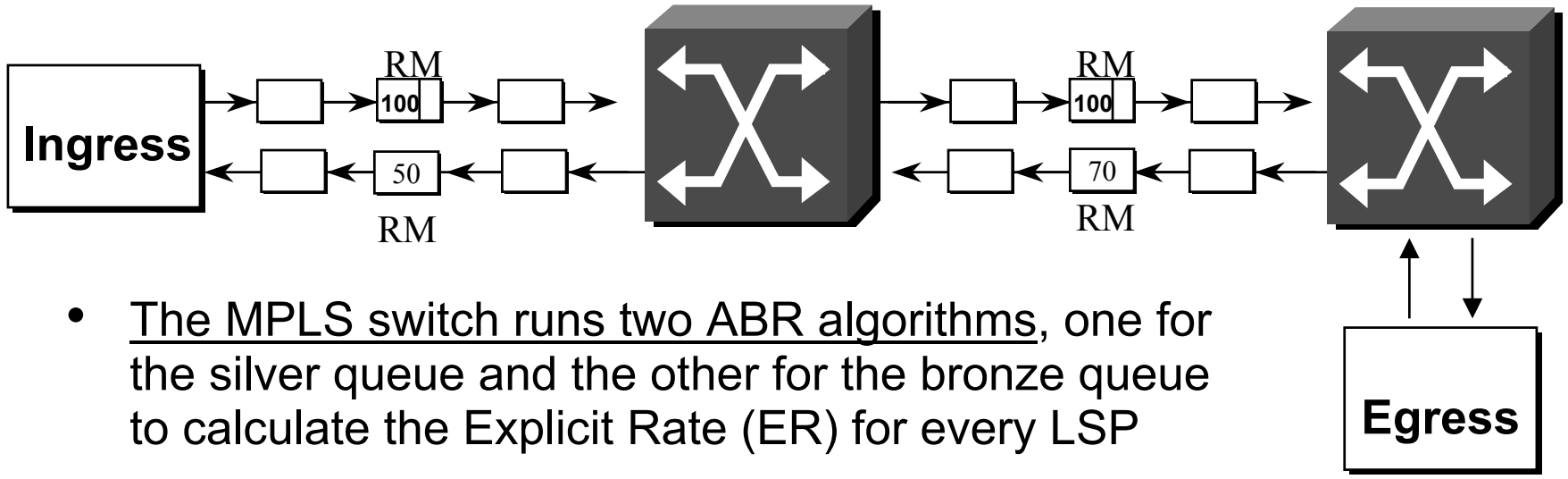
- Primary LSP (P-LSP) uses the minimum hop path found using Dijkstra's algorithm
- Secondary LSP (S-LSP) is found by pruning the links used by P-LSP and using Dijkstra's algorithm on the remaining network graph
- Other algorithms to find two link-disjoint paths and their impact on throughput left for future research

MPLS Queueing



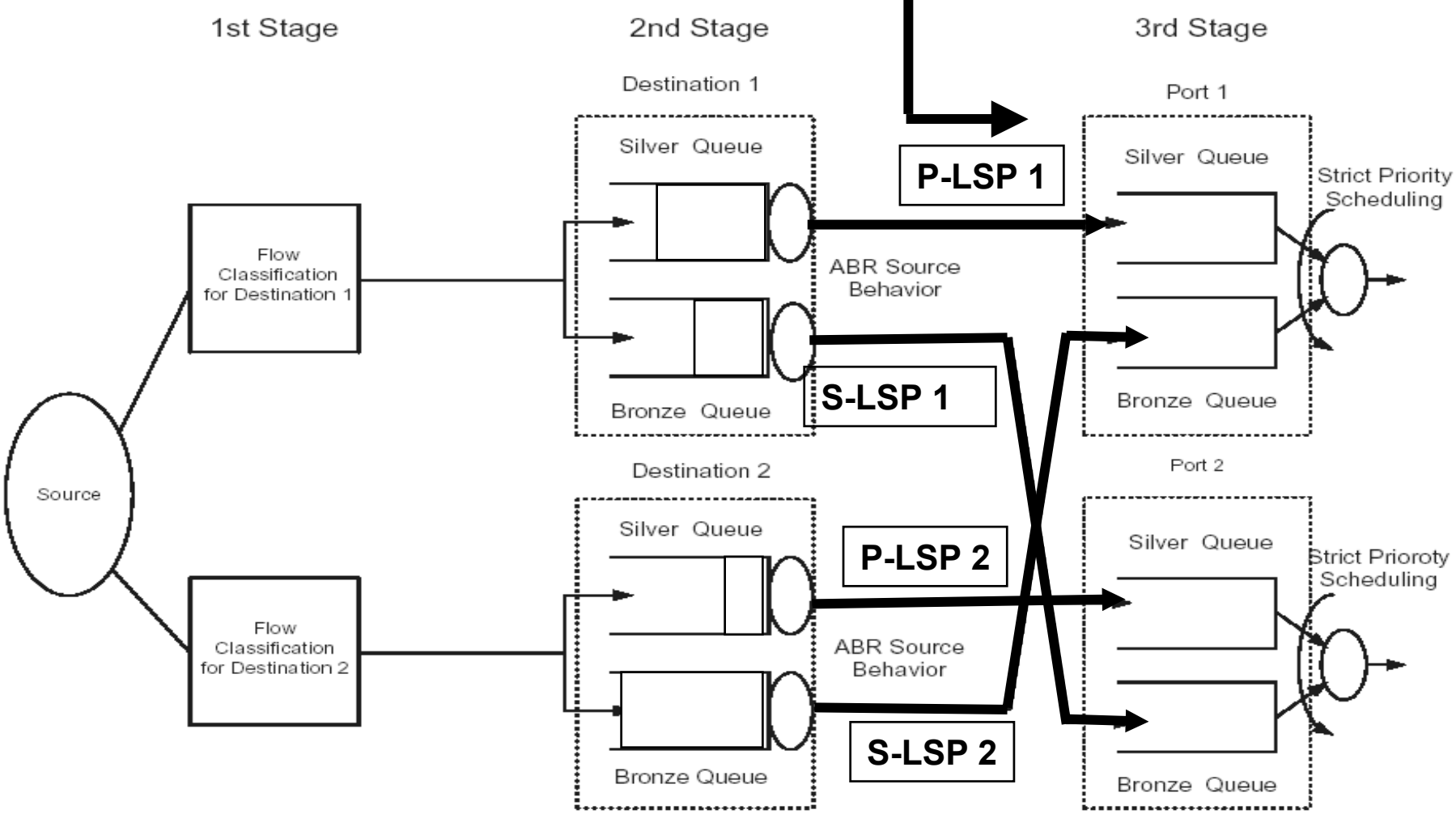
- RM packets have strict priority over P-LSP packets and P-LSP packets have strict priority over S-LSP packets
- EXP-inferred-LSP (E-LSP) method is used for tagging the packets

MPLS Feedback Mechanism



- The MPLS switch runs two ABR algorithms, one for the silver queue and the other for the bronze queue to calculate the Explicit Rate (ER) for every LSP
- We use the variable packet version of ERICA ABR explicit rate algorithm
- Resource Management (RM) packet to network (once in 15 packets)
- When RM is on way back from egress to ingress:
  - Each switch sets ER field to minimum of ER in RM and the rate switch can support
  - CI and NI are set according to buffer occupancy

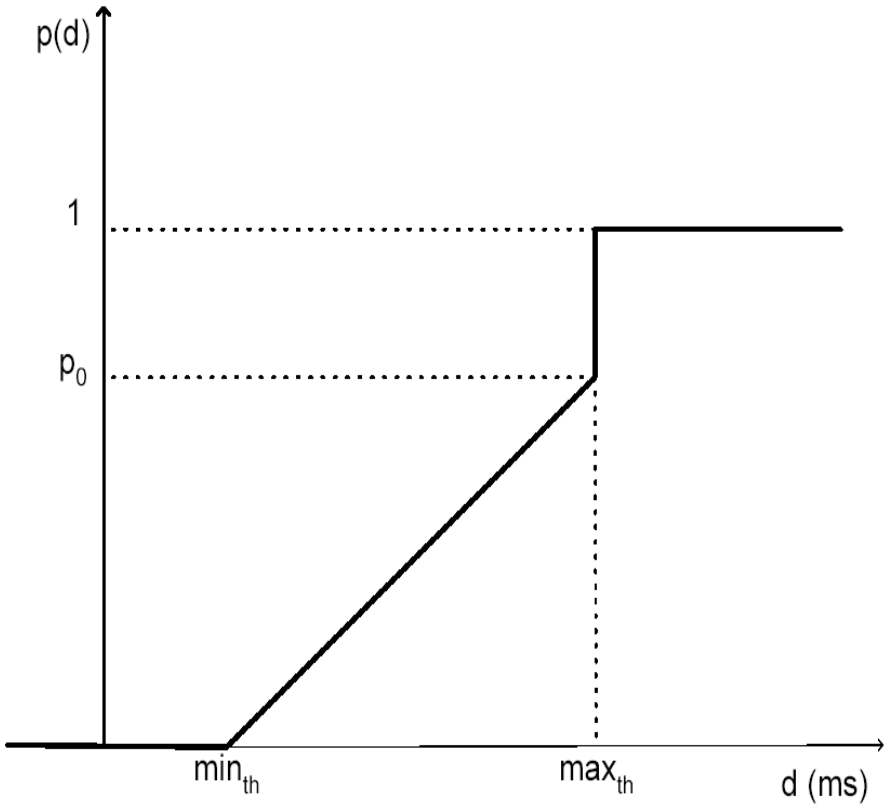
MPLS Edge Switch



Flow-based Routing and Traffic Splitting

- Silver and bronze queues are drained using Allowed Traffic Rate (ATR) via standard ABR source behavior
- $D_{P\text{-}LSP}$ and $D_{S\text{-}LSP}$ denote delay estimates for P-LSP and S-LSP
- These delay estimates are calculated dividing the corresponding queue occupancy by ATR of that queue
- Δ denotes smoothed difference between delay estimates
- $\Delta_{new}=\gamma(D_{P\text{-}LSP}-D_{S\text{-}LSP})+(1-\gamma)\,\Delta_{old}$
- D_{max} denotes the maximum allowable delay
- We adopt a variant of Random Early Discard (RED) algorithm
- We control average delay difference

Random Early Reroute (RER)

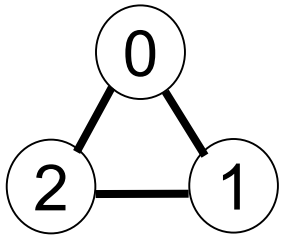


- Discard the packet if
  - $D_{P\text{-}LSP}, D_{S\text{-}LSP} \geq D_{max}$
- Forward flow over P-LSP if
  - $D_{P\text{-}LSP} < D_{max}, D_{S\text{-}LSP} \geq D_{max}$
- Forward flow over S-LSP if
  - $D_{P\text{-}LSP} \geq D_{max}, D_{S\text{-}LSP} < D_{max}$
- Forward flow over S-LSP with probability $p(\Delta)$

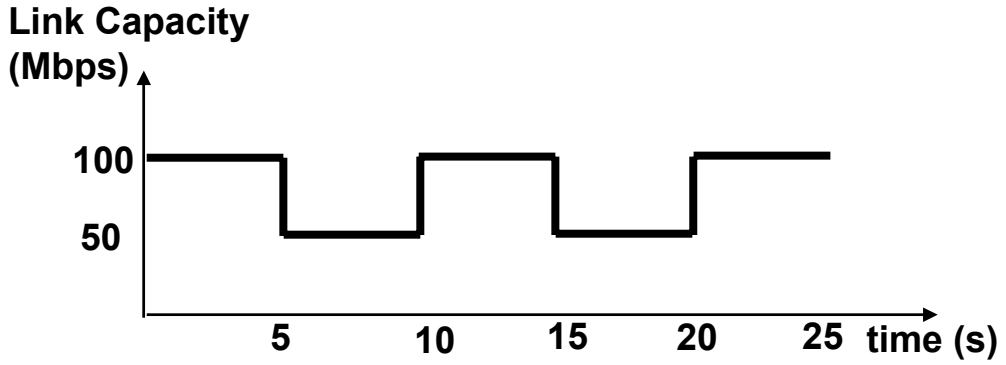
Traffic Model

- An event-driven special purpose flow/packet-based MPLS simulator using the Java programming language
- M/G/∞ model is used to model incoming flow/packet arrival process
  - Flow arrivals are Poisson
  - Packet interarrivals within a flow are exponentially distributed
  - Packet lengths are fixed (128 bytes)
  - RM packets are 50 bytes
  - Flows consist of a random number of packets (e.g., geometrically distributed) with mean M_f

Transient Performance



3 node ring topology

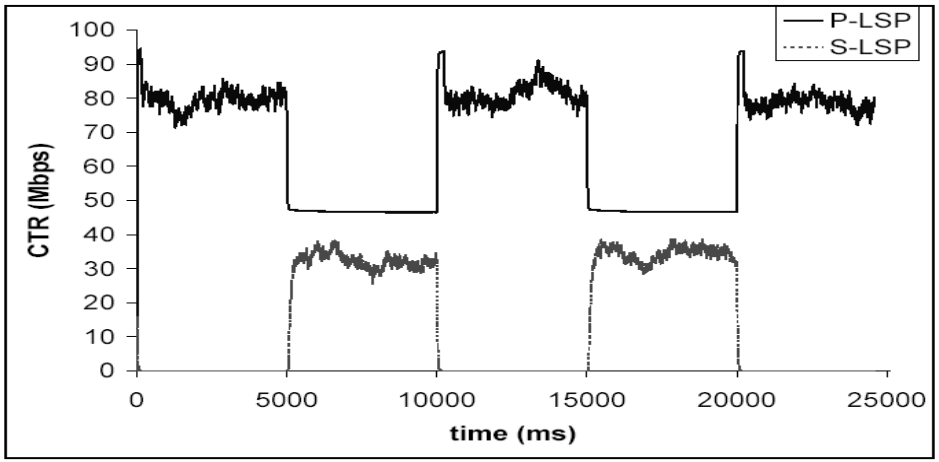


Link capacity from node 1 to node 0

- 80 Mbps traffic demand from node 1 to 0
- All link capacities 100 Mbps except link capacity between node 1 and 0 alternates periodically between 100 Mbps and 50 Mbps
- We compare and contrast
  - Single Path Routing (SPR)
  - Flow-based Multipath Routing (FBMPR)
  - Packet-based Multipath Routing (PBMPR)

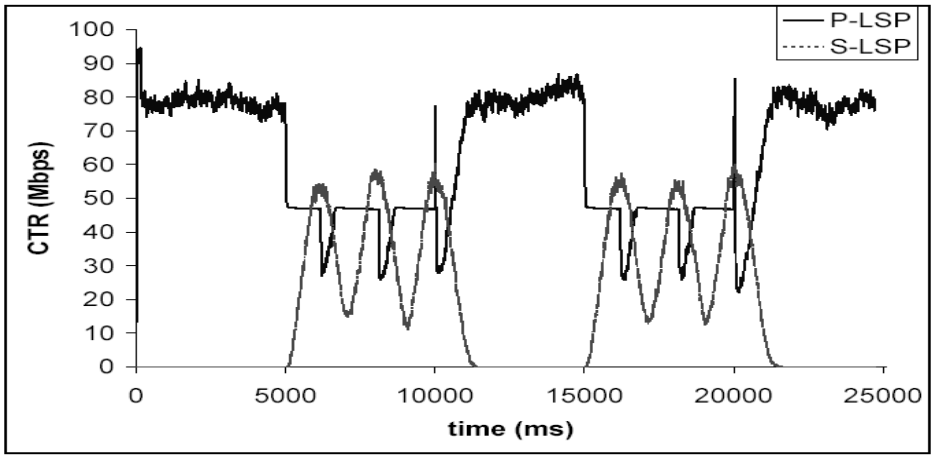
Transient Performance

PBMPR



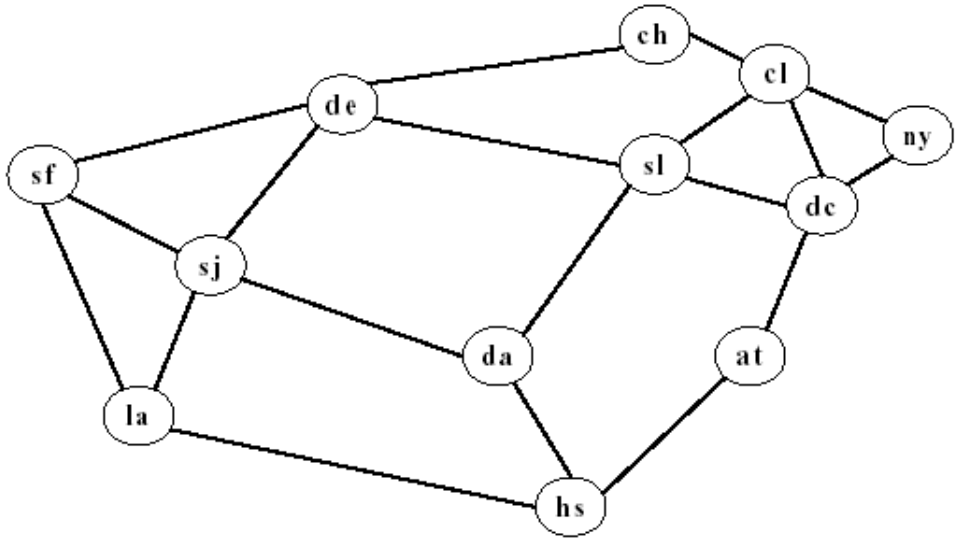
- Perfect packet-level throughput but
- Network Reordering Rate (NROR) ≈ 10%

FBMPR



- Underutilization in one queue and overutilization in the other queue due to flow-based routing
- Price we pay for in-order packet delivery

Mesh Network Topology



- The topology and the traffic demand matrix are used from the data given in ***www.fictitious.org/omp***
- Each link is bi-directional and has 45 Mbps capacity in both directions except the links between de-ch and ch-cl have 90 Mbps capacity in both directions

NLR (Network Loss rate) and NROR (Network Reordering Rate)

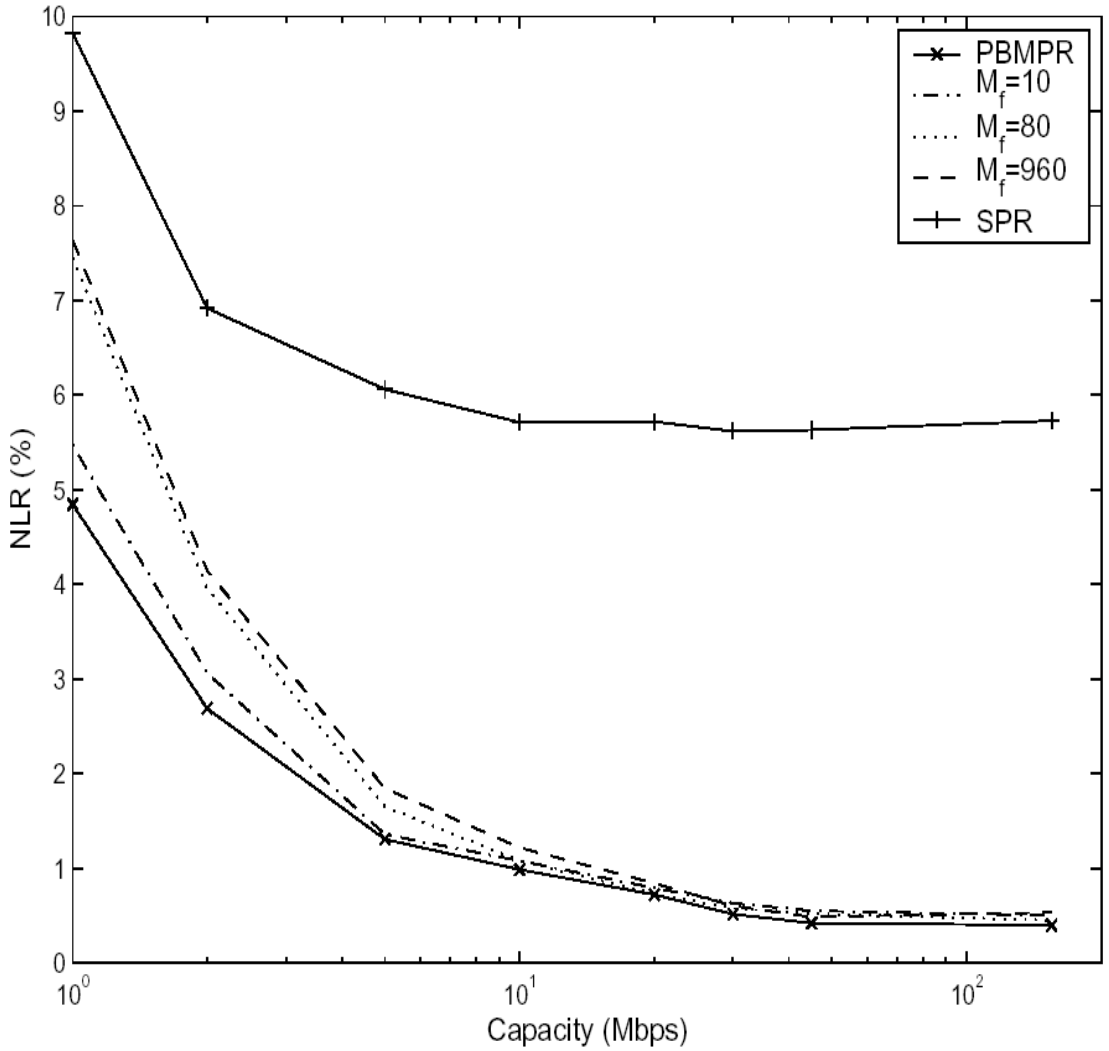
Method	NLR (%)	NROR (%)
SPR	5.559	-
FBMPR	0.783	-
PBMPR	0.445	9.169

- NLR is reduced when multipath is used
- BPMPR slightly outperforms FBMPR
- NROR is zero for SPR and FBMPR
- NROR is 9.169 % for PBMPR

Impact of Flow Parameters

- What happens when we increase the link capacities and flow arrival rates together
  - Move towards the core regime and see if the proposed scheme gets more effective
- What is the impact of the average flow size
  - NLR as a function of M_f

Impact of Flow Model Parameters



- Reduction in NLR for SPR and PBMPR can be explained through statistical multiplexing
  - Larger the flow arrival rates, more number of decision epochs to split traffic
  - Increasing the control frequency improves performance of system
  - FBMPR is almost indistinguishable from PBMPR when C> 45 Mbps, M_f < 960 packets

Knock-on Effect

- Vary the topology from a densely connected mesh topology to a ring topology
  - A: original hypothetic US topology (densely connected)
  - B: delete links dc-sl and dc-cl from A
  - C: delete links da-hs and sl-cl from B
  - D: delete links sf-sj and la-sj from C
  - E: connect all nodes via a ring (sparsely connected – ring)
- We compare and contrast
  - FBMPR: MPLS strict priority queueing + RER
  - Method I: FIFO queueing + RER
  - Method II: FIFO queueing + Choose the LSP with lower delay

Knock-on Effect

Topology	PLR	FBMPR	Method I	Method II
A	1.514	99.351	99.857	88.460
B	1.614	94.363	95.905	83.023
C	2.043	87.282	82.020	66.078
D	2.254	74.527	61.378	49.273
E	2.670	61.675	39.261	34.594

Network Throughput (100 – NLR) as a Function of the PLR (Path Length Ratio)

- PLR = Avg. S-LSP hops / Avg. P-LSP hops
- FBMPR > Method I > Method II in general where the performance gain in using FBMPR is proportional with PLR

Conclusions

- We proposed a novel available bit rate traffic engineering methodology in best effort IP networks using Diffserv/MPLS backbones
- Using two disjoint paths in our proposed architecture provides significantly and consistently better results than the case of single LSP
- Isolation of P-LSPs from S-LSPs at the data plane using strict priority is key to the success of this approach
- Distributed multipathTE in short time scales is possible
- Flow-based multipath routing handles packet reordering
- The proposed architecture is most effective in flow-rich backbones ($C > 45$ Mbps, $M_f < 960$ packets)

Future Work

- We are currently working on extending/testing our TE architecture using TCP flows
- We are working on testing our TE approach using more realistic traffic models (self-similar ?) and different network topologies
- What to do with excessively long flows? Split or not?
- How critical are the choice of link-disjoint paths?
- 2 paths or more?
- ER feedback is not standards based for MPLS. We are also working on TE with standards-friendly feedback mechanisms (e.g., one bit feedback)