Lectures 22 & 23

Flow and congestion control

Eytan Modiano

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 I 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**

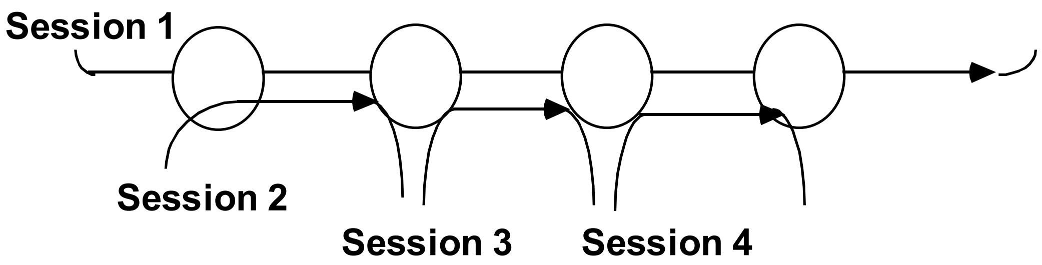
WITHOUT FLOW CONTROL

- 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”

OBJECTIVES OF FLOW CONTROL

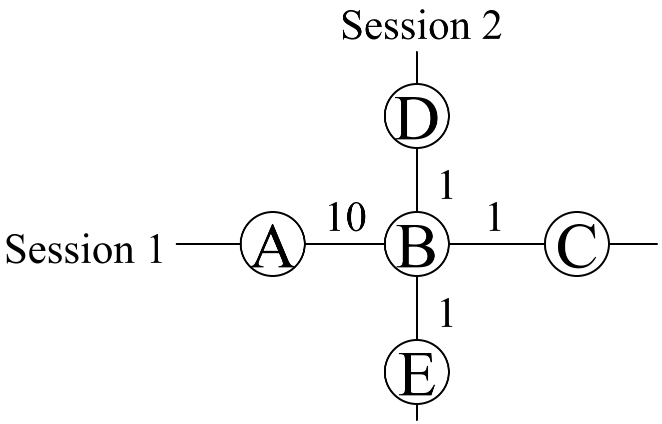
- **Maximize network throughput**
- **Reduce network delays**
- **Maintain quality-of-service parameters**
  - **Fairness, delay, etc..**
- **Tradeoff between fairness, delay, throughput…**

FAIRNESS



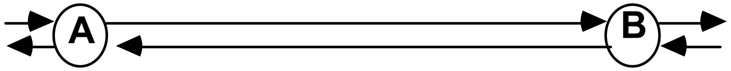
- If link capacities are each 1 unit, then
  - Maximum throughput is achieved by giving short session one unit and zero units to the long session; total throughput of 3 units
  - One concept of fairness would give each user 1/2 unit; total throughput of 2 units
  - Alternatively, giving equal resources to each session would give single link users 3/4 each, and 1/4 unit to the long session

FAIRNESS



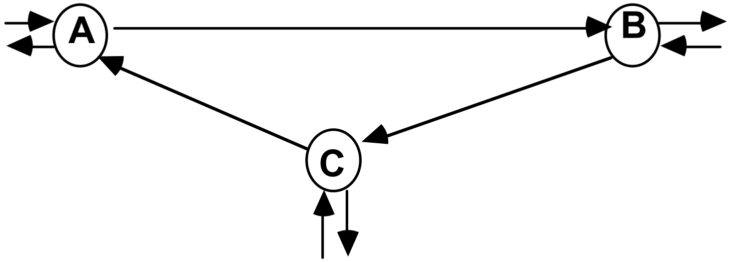
- Limited buffer at node B
- Clearly both sessions are limited to 1 unit of traffic
- Without flow control, session 1 can dominate the buffer at node B
    - Since 10 session 1 packets arrive for each session 2 packet, 10/11 packets in the buffer will belong to session 1

DEADLOCKS FROM BUFFER OVERFLOWS



- 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.

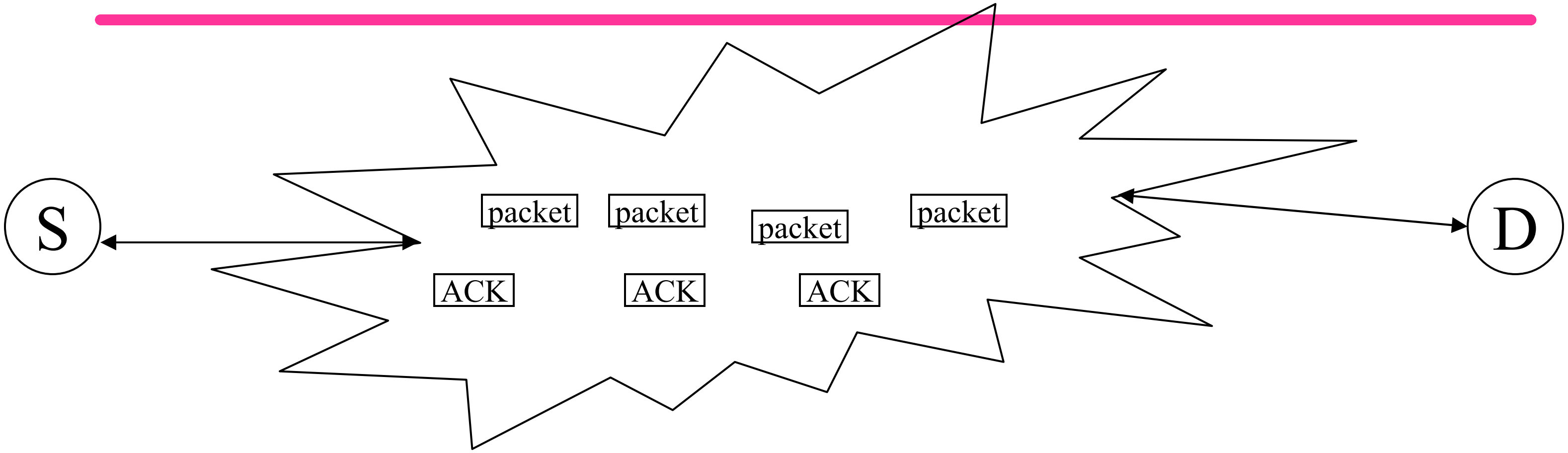


QuickTime™ and a Photo - JPEG decompressor are needed to see this picture.

WINDOW FLOW CONTROL

QuickTime™ and a GIF decompressor are needed to see this picture.

LIDS



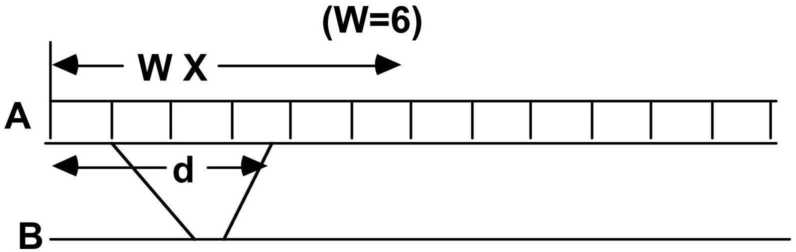
- Similar to Window based ARQ
  - End-to-end window for each session, Wsd
  - Each packet is ACK'd by receiver
  - Total number of un-ACK's packets <= Wsd

⇒ Window size is an upper-bound on the total number of packets and ACKs in the network

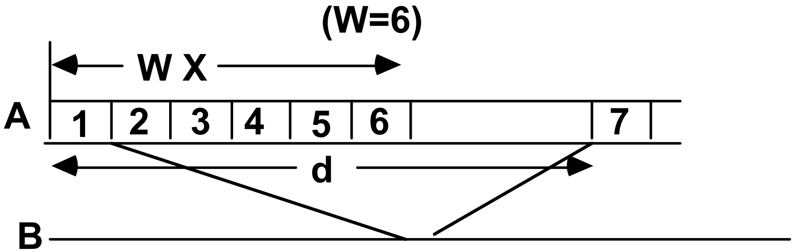
⇒ Limit on the amount of buffering needed inside network

END TO END WINDOWS

- **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**

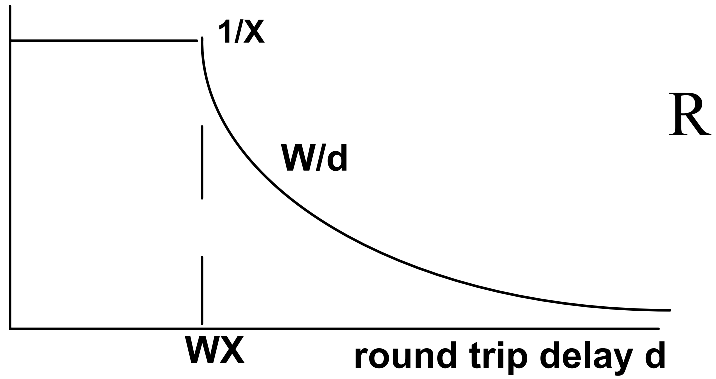


Flow control not active



Flow control active

Behavior of end-end windows



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

- **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)**

⇒ **$Wx > d' \Rightarrow W > 2N + Dp/x$**

- **When $Dp < x$, $W \sim 2N$ (window size is independent of prop. Delay)**
- **When $Dp \gg Nx$, $W \sim 2Dp/x$ (window size is independent on path length**

Impact of congestion

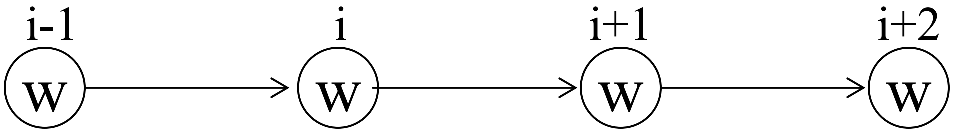
- Without congestion d = d’ and flow control is inactive
- With congestion d > d’ and flow control becomes active

- 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 WINDOWS

- 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 WINDOWS



- 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 BASED FLOW 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

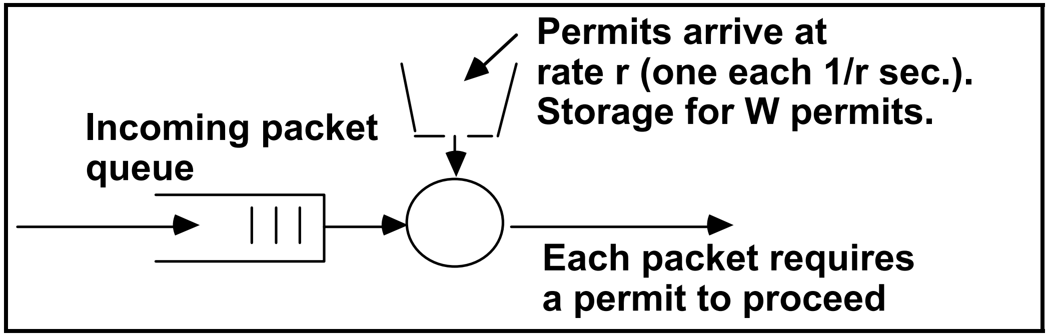
    => TDMA => inefficient for bursty traffic

  – Less-strict implementation: Allow W packets every W/r seconds

    Average rate remains the same but bursts of up to W packets are allowed

    Typically implemented using a “leaky bucket” scheme

LEAKY BUCKET RATE CONTROL

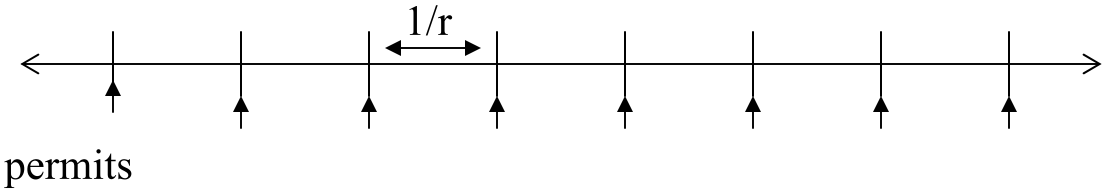


- Session bucket holds W permits
  - In order to enter the network, packet must first get a permit
  - Bucket gets new permits at a rate of one every 1/r seconds
- When the bucket is full, a burst of up to W packets can enter the network
  - The parameter W specifies how bursty the source can be
    - Small W => strict rate control
    - Large W supports 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 ANALYSIS OF LEAKY BUCKET



- 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

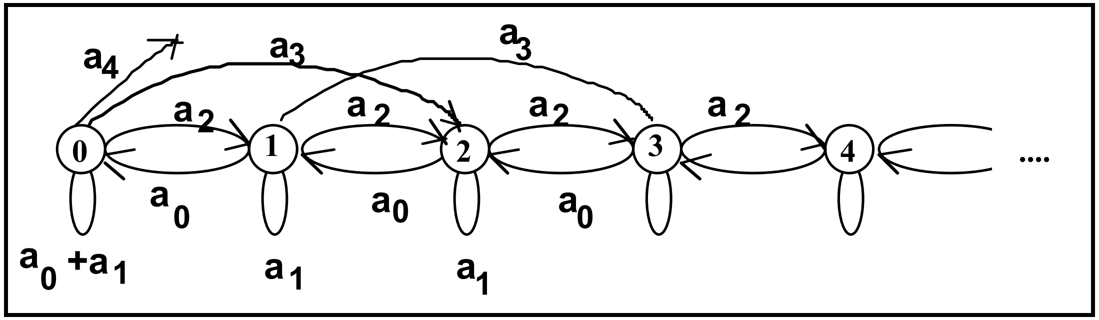
QuickTime™ and a Photo - JPEG decompressor are needed to see this picture.

QUEUEING ANALYSIS, continues

QuickTime™ and a GIF decompressor are needed to see this picture.

LIDS

System Markov Chain:



- 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)

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

- Now from global balance eqns:
  - $P(0)\,[1-a_0 - a_1] = a_0\,P(1)$
  - $P(1) = [(1-a_0-a_1)/a_0]P(0)$ => can solve for P(1) in terms of P(0)
  - $P(1)[1-a_1] = a_2P(0) + a_0P(2)$ => obtain P(2) in terms of P(1)
  - Recursively solve for all P(i)'s
- Average delay to obtain a permit = $T = \left[\sum_{j=W+1}^{\infty}(j-W)P(j)\right]\frac{1}{r}$

Choosing a value for r

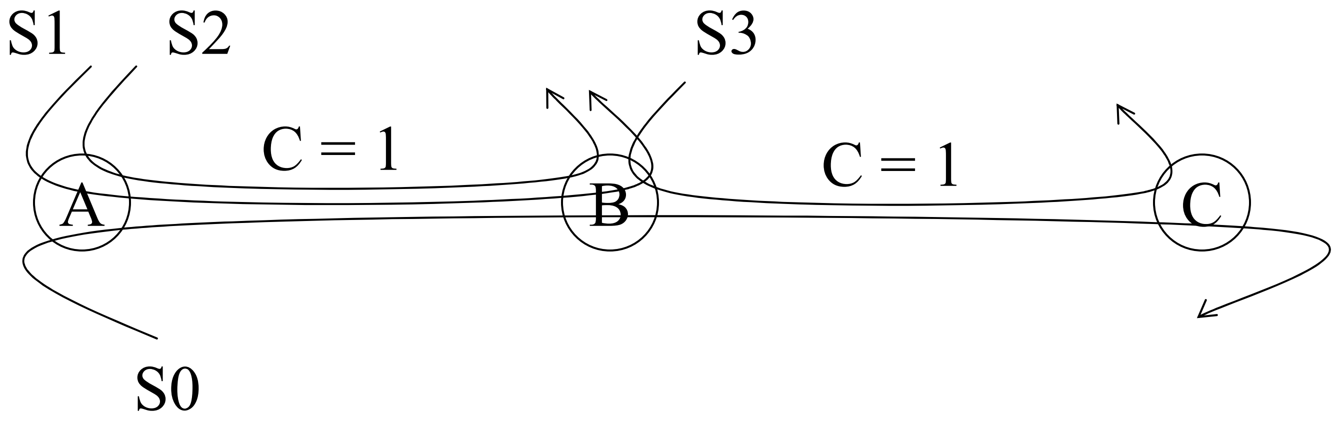
- **How do we decide on the rate allocated to a session?**

- **Approaches**

  1. **Optimal routing and flow control**
     - **Tradeoff between delay and throughput**
  2. **Max-Min fairness**
     - **Fair allocation of resources**
  3. **Contract based**
     - **Rate negotiated for a price (e.g., Guaranteed rate, etc.)**

Max-Min Fairness

- Treat all sessions as being equal
- Example:



- 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 notion

- 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 do it!

- Given a set of session requests P and an associated set of rates R_P, R_P is max-min fair if,
  - For each session p, r_p cannot be increased without decreasing $r_{p'}$ for some session p’ for which $r_{p'} <= r_p$

Max-Min fair definition

- 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:

$$F_a = \sum_{\substack{\forall p\ cros\sin g \\ link\ a}} r_p$$

- 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 and

  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 $r_{p'} <= r_p$

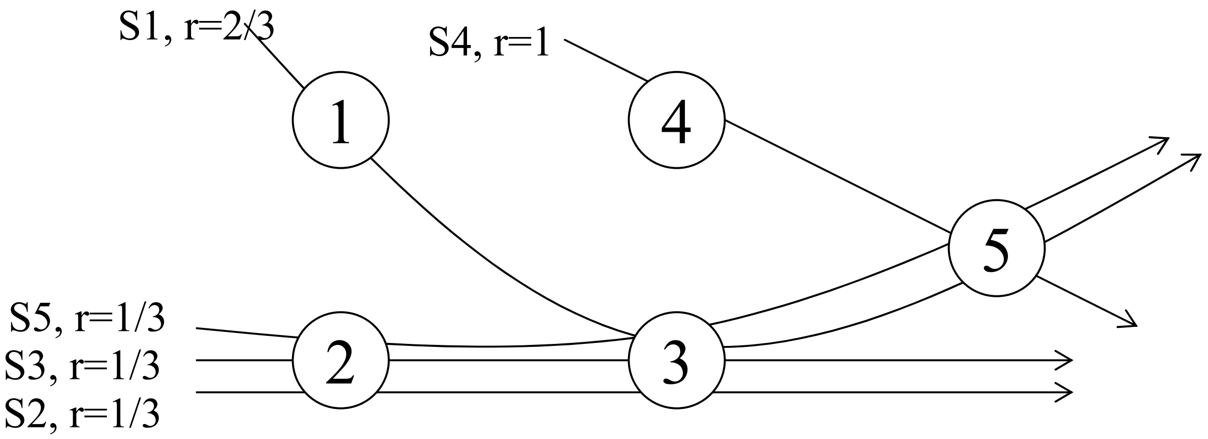
- In other words, you can only increase the rate of a path by decreasing the rate of another path that has been allocated no more capacity

Bottleneck link

- **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

Generalized processor sharing (AKA fair queueing)

- **Serve session in round-robin order**
  - **If sessions always have a packet to send they each get an equal share of the link**
  - **If some sessions are idle, the remaining sessions share the capacity equally**

- **Processor sharing usually refers to a “fluid” model where session rates can be arbitrarily refined**

- **Generalized processor sharing is a packet based approximation where packets are served from each session in a round-robin order**