Z24: Packet Scheduling and Enhanced Quality of Service

Mark Handley

Traditional queuing behaviour in routers

- Data transfer:
  - datagrams: individual packets
  - no recognition of **flows**
  - connectionless: no signalling
- Forwarding:
  - based on per-datagram, forwarding table look-ups
  - no examination of “type” of traffic – no **priority** traffic

Packet Scheduling

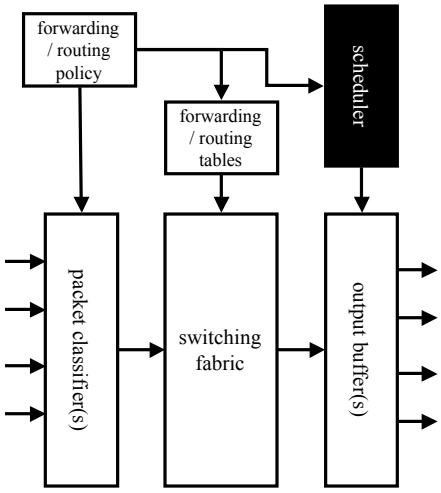
Scheduling

- A server receives a stream of requests.
  - which request to service first?
- Scheduler:
  - decides service order (based on policy/algorithm)
  - manages queues for service.
- Router (network packet handling server):
  - **service:** packet forwarding
  - **scheduled resource:** output queues
  - **service requests:** packets arriving on input lines

Scheduling

Simple router schematic

- Input lines:
  - no input buffering
- Packet classifier:
  - policy-based classification
- Correct output queue:
  - forwarding/routing tables
  - switching fabric
  - output buffer (queue)
- Scheduler:
  - which output queue serviced next



First come, first served (FCFS) scheduling

- Null packet classifier
- Packets queued to outputs in order they arrive
- No packet differentiation
- No notion of flows of packets
- Anytime a packet arrives, it is serviced as soon as possible:
  - FCFS is a *work-conserving* scheduler (not idle if packets waiting)
    - Reducing the delay of one flow, implies increasing the delay of one or more others.
    - We can not give *all* flows a lower delay than they would get under FCFS

Non-work-conserving schedulers

Non-work conserving disciplines can be idle even if packets are waiting.

- This allows "smoothing" of packet flows.
- Do not serve packet as soon as it arrives - wait until packet is *eligible* for transmission.

- ✓ Less jitter
- ✓ Makes downstream traffic more predictable and less bursty.
- ✓ Less buffer space:
  - router: output queues
  - end-system: de-jitter buffers
- × Higher end-to-end delay
- × Complex in practice.

Simple priority queuing

K queues:

- $1 \le k \le K$
- queue $k + 1$ has greater priority than queue k
- higher priority queues serviced first.

- ✓ Very simple to implement
- ✓ Low processing overhead
- ■ Relative priority:
  - no deterministic performance bounds
- × Fairness and protection:
  - starvation of low priority queues

Fair Queuing

- Allocate each flow to a separate queue.
  - What is a flow? Policy issue.
- Max-min fair share:
  - Allocate bandwidth equally between flows.
  - If a flow can't use its bandwidth (because of constraints elsewhere), the excess is divided equally amongst the other unconstrained flows.
- True fair queuing (aka Generalized Processor Sharing) is not implementable in practice.
  - Assumes bit-by-bit forwarding.

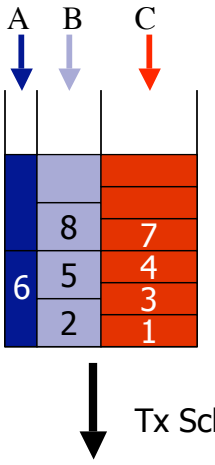
Weighted round-robin (WRR)

- Simplest attempt at GPS
- Queues visited round-robin in proportion to weights assigned
- Different mean packet sizes:
  - weight divided by mean packet size for each queue
- Mean packets size unpredictable:
  - may cause unfairness
- Service is fair over long timescales:
  - must have more than one visit to each flow/queue
  - short-lived flows?
  - small weights?
  - large number of flows?

Weighted Fair Queuing

- Based on GPS:
  - GPS emulation to produce *finish-numbers* for packets in queue
  - Simplification: GPS emulation serves packets bit-by-bit round-robin
- ***Finish-number:***
  - the time packet would have completed service under (bit-by-bit) GPS
  - packets tagged with finish-number
  - smallest finish-number across queues served first

Weighted Fair Queuing



3 flows A, B, C
Weights:
A: 1
B: 2
C: 3
Assume same packet sizes.

Tx Schedule: C B C C B A C B ...

Weighted Fair Queuing

- Buffer drop policy:
  - packet arrives at full queue
  - drop packets already in queued, in order of decreasing finish-number.
- Can be used for:
  - best-effort queuing
  - providing guaranteed data rate and deterministic end-to-end delay
- WFQ used in “real world”
  - Cisco implementation: hash flows across 256 queues.

Fair Queuing: Pragmatic Issues

- Per-flow (src, dest, sport, dport, proto) fair queuing:
  - Technically feasible.
  - Lots of state in the fast path.
  - Very fast memory is expensive.
- Probably not needed in high-speed routers!
  - At the edges, would be a big benefit.
- **Warning: DoS attacks imminent**.
  - An attacker may be able to spoof a lot of different low-rate flows and cause the legitimate flows to go very slowly.

Enhanced Quality of Service

Questions

- Can we do better than **best-effort**?
- What support do real-time flows need in the network?
- What support can we provide in the network?
- QoS for many-to-many communication?
- Application-level interfaces?
- Signalling

Better Service

- RSVP/Intserv
- Diffserv
- where's it going???

Isn't Best Effort Service Sufficient?

In theory, yes.

- If there's sufficient capacity to accommodate all the real-time flows (as there is in the phone network) then best effort is sufficient.
  - Queues do not build
  - No packet loss occurs
- If there's not sufficient capacity to accommodate them, calls will either block if we have reservations or give degraded service if we don't.
  - Neither of these is acceptable.
  - Thus there must be sufficient capacity.

Isn't Best Effort Service Sufficient?

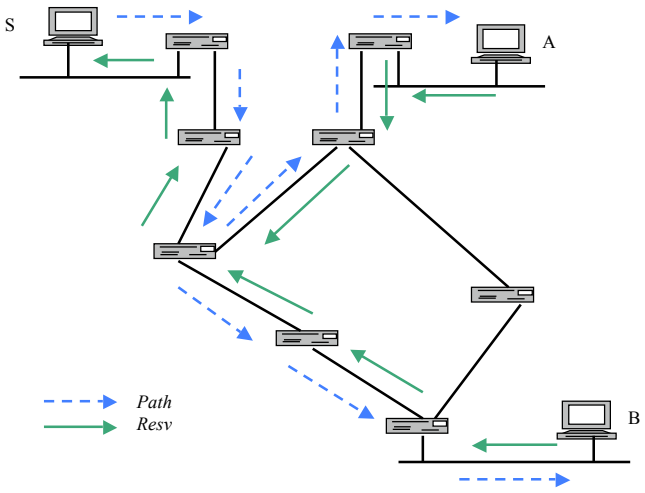
- In practice, probably not.
  - When demand grows exponentially, ISPs trail the demand curve at least some of the time.
  - TCP traffic expands to fill available bandwidth and produces loss in doing so.
  - Simple prioritization of real-time traffic leads to falsely described traffic.
  - Getting from here to there is difficult - someone has to pay for the infrastructure.

The Goal

- The trick is to deploy mechanisms that are:
  - not required to obtain service, but which can be used to obtain better service if best-effort isn't adequate,
  - require minimal network state so we can build fast routers,
  - can be charged for to improve the network for everyone,
  - require billing arrangements that are feasible.

RSVP and Intserv

RSVP: Details



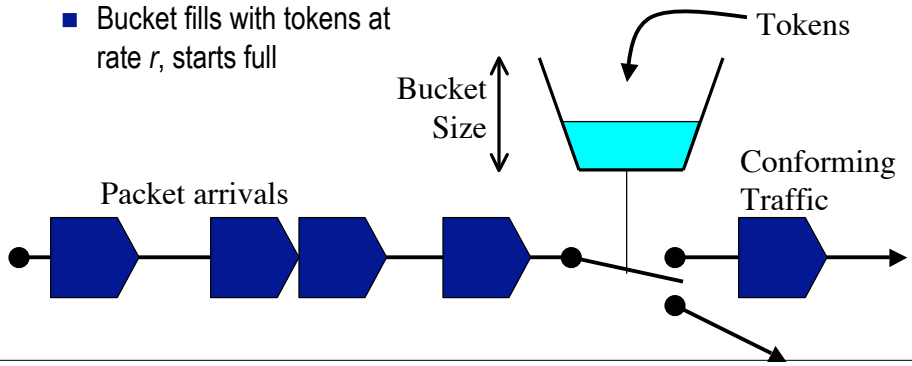
RSVP: Reservation Styles

Several styles of reservation are supported:

- Fixed Filter
  - separate reservations for each listed sender.
  - E.g.: several video streams.
- Shared Explicit
  - one reservation shared between several listed senders.
  - E.g.: video with floor control
- Wildcard
  - one reservation for any senders.
  - E.g.: audio with silence detection in a large group

Token bucket

- Three parameters:
  - *b*: bucket size [B]
  - *r*: bucket rate [B/s]
  - *p*: peak rate [B/s]
- Bucket fills with tokens at rate *r*, starts full
- Tokens allow transmission
- Burst allowed at rate *p:*
  - data sent < *rt* + *b*



Intserv: Integrated Services

Two Intserv service models were standardized:

- Controlled Load Service
  - This is the one you want.
  - If you want Intserv at all.
- Guaranteed Service
  - Practically no-one needs this.

Controlled Load Service

- The goal is to make it look like the network is unloaded.
  - It does not guarantee jitter bounds or no loss
  - both are very low though.
- Traffic is policed at the network edges and split/merge points.
  - If it exceeds the reservation, it is treated as best effort.
  - A token-bucket is used for policing and specified in reservation requests.
  - Admission control ensures that reservations do not exceed the available bandwidth.
- Controlled Load packets get priority over Best Effort
  - Best Effort packets are not pre-empted, so some jitter is seen.
  - Cumulative jitter can lead to small, temporary queues.

Guaranteed Service

- Both bandwidth and delay bounds are guaranteed.
  - Traffic is policed at the network edges and split/merge points.
  - If it exceeds the reservation, it is treated as best effort.
  - A token-bucket is used for policing and specified in reservation requests.
- Admission control ensures that reservations do not exceed the available bandwidth.
  - In addition, *buffer slots are scheduled*.
  - Guaranteed Service packets get priority over Controlled Load.
  - If a packet arrives before it's buffer slot, *it is delayed until that slot*. In this way jitter does not accumulate, so no temporary GS queues form (other than for shaping).
- Delay is normally longer than with Controlled Load, but there's no distribution tail.

Why isn't everyone doing it?

The protocols and mechanisms work OK.
It solves the problem people *thought* they wanted solved.

- Some minor issues:
  - Extra traffic due to soft-state refreshes
  - Route changes & router failure:
    - QoS degrades to best-effort, need to re-negotiate QoS
- Two Serious Problems:
  - Charging/authentication
  - Router State

RSVP/Intserv Charging

- A reservation goes hop-by-hop across many ISPs.
  - Why should I reserve bandwidth for some receiver I've never heard of?
- Need negative feedback to discourage reservations or everything gets reserved.
  - Essentially this means charging.
  - Vanilla RSVP needs n^2 billing arrangements between n ISPs.

Router State

- Backbone routers currently handle O(1,000,000) simultaneous connections.
- We don't want a significant proportion of these to have reservation state:
  - Fast router memory is very expensive.
  - CPU Cycles to check the flow spec are in very short supply.
    - Bandwidth is growing faster than Moore's Law
    - In the future we'll have less cycles per packet than we have now.

Router State: Solutions?

- Only police/install state at the edges.
  - Most of the congestion is at the edges.
  - Do something different (or nothing at all) in the backbone.