Scheduling and queue management

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
- Traffic patterns

Questions

- How do we modify router scheduling behaviour to support QoS?
- What are the alternatives to FCFS?
- How do we deal with congestion?

Scheduling mechanisms

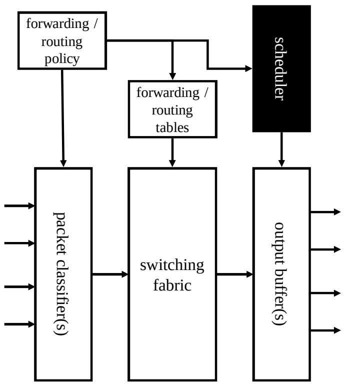
Scheduling [1]

- Service request at server:
  - e.g. packet at router inputs
- Service order:
  - which service request (packet) to service first?
- Scheduler:
  - decides service order (based on policy/algorithm)
  - manages service (output) queues
- Router (network packet handling server):
  - **service:** packet forwarding
  - **scheduled resource:** output queues
  - **service requests:** packets arriving on input lines

Scheduling [2]

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



FCFS scheduling

- Null packet classifier
- Packets queued to outputs in order they arrive
- Do 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

Conservation law [1]

- FCFS is work-conserving:
  - not idle if packets waiting
- Reduce delay of one flow, increase the delay of one or more others
- We can not give *all* flows a lower delay than they would get under FCFS

$$\sum_{n=1}^{N} \rho_n q_n = C$$

$\rho_n = \lambda_n \mu_n$

ρ_n : mean link utlisation

q_n : mean delay due to scheduler

C : constant [s]

λ_n : mean packet rate [p/s]

μ_n : mean per – packet service rate [s/p]

Conservation law [2]

Example

- μ_n : 0.1ms/p (fixed)
- Flow f1:
  - λ_1 : 10p/s
  - q_1 : 0.1ms
  - $\rho_1 q_1 = 10^{-7}$s
- Flow f2:
  - λ_2 : 10p/s
  - q_2 : 0.1ms
  - $\rho_2 q_2 = 10^{-7}$s
- $C = 2\times10^{-7}$s

- Change f1:
  - λ_1 : 15p/s
  - q_2 : 0.1s
  - $\rho_1 q_1 = 1.5\times10^{-7}$s
- For f2 this means:
  - decrease λ_2?
  - decrease q_2?
- Note the trade-off for f2:
  - **delay vs. throughput**
- Change service rate (μ_n):
  - change service **priority**

Non-work-conserving schedulers

- Non-work conserving disciplines:
  - can be idle even if packets waiting
  - allows “smoothing” of packet flows
- Do not serve packet as soon as it arrives:
  - what until packet is **eligible** for transmission
- Eligibility:
  - fixed time per router, or
  - fixed time across network

- ✓ Less jitter
- ✓ Makes downstream traffic more predictable:
  - output flow is controlled
  - less bursty traffic
- ✓ Less buffer space:
  - router: output queues
  - end-system: de-jitter buffers
- ✗ Higher end-to-end delay
- ✗ Complex in practise
  - may require time synchronisation at routers

Scheduling: requirements

- Ease of implementation:
  - simple → fast
  - high-speed networks
  - low complexity/state
  - implementation in hardware
- Fairness and protection:
  - local fairness: **max-min**
  - local fairness → global fairness
  - protect any flow from the (mis)behaviour of any other

- Performance bounds:
  - per-flow bounds
  - deterministic (guaranteed)
  - statistical/probabilistic
  - data rate, delay, jitter, loss
- Admission control:
  - (if required)
  - should be easy to implement
  - should be efficient in use

The max-min fair share criteria

- Flows are allocated resource in order of increasing demand
- Flows get no more than they need
- Flows which have not been allocated as they demand get an equal share of the available resource
- Weighted max-min fair share possible
- If max-min fair → provides protection

$$m_n = \min(x_n, M_n) \quad 1 \le n \le N$$

$$M_n = \frac{C - \sum_{i=1}^{n-1} m_i}{N - n + 1}$$

C : capacity of resource (maximum resource)
m_n : actual resource allocation to flow n
x_n : resource demand by flow $n, x_1 \le x_2 \cdots \le x_N$
M_n : resource available to flow n

Example:
C = 10, four flow with demands of 2, 2.6, 4, 5
actual resource allocations are 2, 2.6, 2.7, 2.7

Scheduling: dimensions

- Priority levels:
  - how many levels?
  - higher priority queues services first
  - can cause starvation lower priority queues
- Work-conserving or not:
  - must decide if delay/jitter control required
  - is cost of implementation of delay/jitter control in network acceptable?
- Degree of aggregation:
  - flow granularity
  - per application flow?
  - per user?
  - per end-system?
  - cost vs. control
- Servicing within a queue:
  - "FCFS" within queue?
  - check for other parameters?
  - added processing overhead
  - queue management

Simple priority queuing

- *K* queues:
  - $1 \leq k \leq K$
  - 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:
  - not max-min fair: starvation of low priority queues

Generalised processor sharing (GPS)

- Work-conserving
- Provides max-min fair share
- Can provide weighted max-min fair share
- Not implementable:
  - used as a reference for comparing other schedulers
  - serves an infinitesimally small amount of data from flow *i*
- Visits flows round-robin

$\phi(n) \quad 1 \leq n \leq N$

$S(i,\tau,t) \quad 1 \leq i \leq N$

$$\frac{S(i,\tau,t)}{S(j,\tau,t)} \geq \frac{\phi(i)}{\phi(j)}$$

$\phi(n)$: weight given to flow n

$S(i,\tau,t)$: service to flow i in interval $[\tau\tau,t$

flow i has a non – empty queue

GPS – relative and absolute fairness

- Use fairness bound to evaluate GPS emulations (GPS-like schedulers)
- Relative fairness bound:
  - fairness of scheduler with respect to other flows it is servicing
- Absolute fairness bound:
  - fairness of scheduler compared to GPS for the same flow

$$RFB = \left| \frac{S(i,\tau,t)}{g(i)} - \frac{S(j,\tau,t)}{g(j)} \right|$$

$$AFB = \left| \frac{S(i,\tau,t)}{g(i)} - \frac{G(i,\tau,t)}{g(i)} \right|$$

$S(i,\tau,t)$: actual service for flow i in $[\tau,t]$

$G(i,\tau,t)$: GPS service for flow i in $[\tau,t]$

$g(i) = \min\{g(i,1), \cdots, g(i,K)\}$

$$g(i,k) = \frac{\phi(i,k) r(k)}{\sum_{j=1}^{N} \phi(j,k)}$$

$\phi(i,k)$: weight given to flow i at router k

$r(k)$: service rate of router k

$1 \leq i \leq N$ flow number

$1 \leq k \leq K$ router number

Weighted round-robin (WRR)

- Simplest attempt at GPS
- Queues visited round-robin in proportion to weights assigned
- Different means 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?

DigiComm II

Deficit round-robin (DRR)

- DRR does not need to know mean packet size
- Each queue has deficit counter (dc): initially zero
- Scheduler attempts to serve one quantum of data from a non-empty queue:
  - packet at head served if size ≤ quantum + dc
    dc ← quantum + dc – size
  - else dc += quantum
- Queues not served during round build up "credits":
  - only non-empty queues
- Quantum normally set to max expected packet size:
  - ensures one packet per round, per non-empty queue
- RFB: 3T/r (T = max pkt service time, r = link rate)
- Works best for:
  - small packet size
  - small number of flows

DigiComm II

Weighted Fair Queuing (WFQ) [1]

- 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
- **Round-number**:
  - execution of round by bit-by-bit round-robin server
  - finish-number calculated from round number
- If queue is empty:
  - finish-number is: *number of bits in packet + round-number*
- If queue non-empty:
  - finish-number is: *highest current finish number for queue + number of bits in packet*

DigiComm II

Weighted Fair Queuing (WFQ) [2]

$$F(i,k,t) = \max\{F(i,k-1,t), R(t)\} + P(i,k,t)$$

$F(i,k,t)$: finish-number for packet k on flow i arriving at time t

$P(i,k,t)$: size of packet k on flow i arriving at time t

$R(t)$: round-number at time t

$$F_{\phi}(i,k,t) = \max\{F_{\phi}(i,k-1,t), R(t)\} + \frac{P(i,k,t)}{\phi(i)}$$

$\phi(i)$: weight given to flow i

- Rate of change of $R(t)$ depends on number of active flows (and their weights)
- As $R(t)$ changes, so packets will be served at different rates
- Flow completes (empty queue):
  - one less flow in round, so
  - R increases more quickly
  - so, more flows complete
  - R increases more quickly
  - etc. …
  - **iterated deletion** problem
- WFQ needs to evaluate R each time packet arrives or leaves:
  - processing overhead

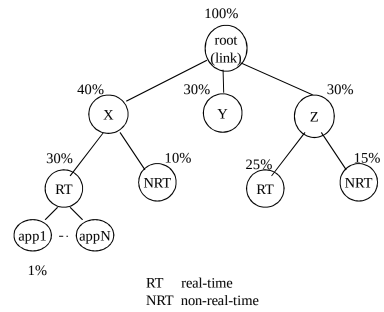
DigiComm II

Weighted Fair Queuing (WFQ) [3]

- 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”
- Alternatives also available:
  - self-clocked fair-queuing (SCFQ)
  - worst-case fair weighted fair queuing (WF^2Q)

Class-Based Queuing

- Hierarchical link sharing:
  - link capacity is shared
  - class-based allocation
  - policy-based class selection
- Class hierarchy:
  - assign capacity/priority to each node
  - node can “borrow” any spare capacity from parent
  - fine-grained flows possible
- Note: this is a queuing mechanism: requires use of a scheduler



Queue management and congestion control

Queue management [1]

- Scheduling:
  - which output queue to visit
  - which packet to transmit from output queue
- Queue management:
  - ensuring buffers are available: memory management
  - organising packets within queue
  - **packet dropping when queue is full**
  - **congestion control**

Queue management [2]

- Congestion:
  - misbehaving sources
  - source synchronisation
  - routing instability
  - network failure causing re-routing
  - congestion could hurt many flows: aggregation
- Drop packets:
  - drop "new" packets until queue clears?
  - admit new packets, drop existing packets in queue?

Packet dropping policies

- Drop-from-tail:
  - easy to implement
  - delayed packets at within queue may "expire"
- Drop-from-head:
  - old packets purged first
  - good for real time
  - better for TCP
- Random drop:
  - fair if all sources behaving
  - misbehaving sources more heavily penalised
- Flush queue:
  - drop all packets in queue
  - simple
  - flows should back-off
  - inefficient
- Intelligent drop:
  - based on level 4 information
  - may need a lot of state information
  - should be fairer

End system reaction to packet drops

- Non-real-time – TCP:
  - packet drop → congestion → slow down transmission
  - slow start → congestion avoidance
  - network is happy!
- Real-time – UDP:
  - packet drop → fill-in at receiver → ??
  - application-level congestion control required
  - flow data rate adaptation not be suited to audio/video?
  - real-time flows may not adapt → hurts adaptive flows
- Queue management could protect adaptive flows:
  - smart queue management required

RED [1]

- Random Early Detection:
  - spot congestion before it happens
  - drop packet → pre-emptive congestion signal
  - source slows down
  - prevents real congestion
- Which packets to drop?
  - monitor flows
  - cost in state and processing overhead vs. overall performance of the network

RED [2]

- Probability of packet drop ∝ queue length
- Queue length value – exponential average:
  - smooths reaction to small bursts
  - punishes sustained heavy traffic
- Packets can be dropped or marked as “offending”:
  - RED-aware routers more likely to drop offending packets
- Source must be adaptive:
  - OK for TCP
  - real-time traffic → UDP ?

TCP-like adaptation for real-time flows

- Mechanisms like RED require adaptive sources
- How to indicate congestion?
  - packet drop – OK for TCP
  - packet drop – hurts real-time flows
  - use ECN?
- Adaptation mechanisms:
  - layered audio/video codecs
  - TCP is unicast: real-time can be multicast

Scheduling and queue management: Discussion

- Fairness and protection:
  - queue overflow
  - congestion feedback from router: packet drop?
- Scalability:
  - granularity of flow
  - speed of operation
- Flow adaptation:
  - non-real time: TCP
  - real-time?
- Aggregation:
  - granularity of control
  - granularity of service
  - amount of router state
  - lack of protection
- Signalling:
  - set-up of router state
  - inform router about a flow
  - explicit congestion notification?

Summary

- Scheduling mechanisms
  - work-conserving vs. non-work-conserving
- Scheduling requirements
- Scheduling dimensions
- Queue management
- Congestion control