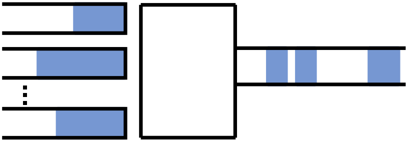
O(1) packet scheduling at high data rates

Luigi Rizzo, Università di Pisa
(joint work with Fabio Checconi and Paolo Valente)

March 30, 2010

O(1) packet scheduling at high data rates

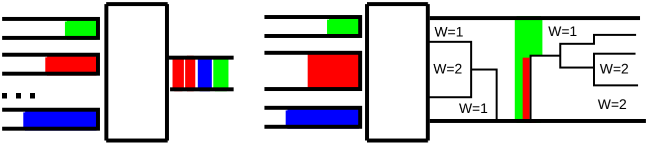


Why do we care about packet scheduling ?

- arbitrate access to common resources;
- provide service guarantees and resource isolation;
- overprovisioning is not always possible/desirable, today's CPUs are too fast;
- links are very fast too, so schedulers must keep up with high data rates and number of flows.

Problem setting and definitions

Many definitions for Service Guarantees. We consider the deviations of our actual scheduler (Packet System) from the service offered by an Ideal Fluid System.



- each flow has a weight Φ_i, and *should* receive a fraction $\Phi_i / \sum_j \Phi_j$ of the total link capacity at any time;
- the Fluid System serves all flows simultaneously;
- the Packet System serves one packet at a time, is non preemptable, online, and possibly work-conserving;

Service Guarantees

Because of its nature, a Packet System cannot guarantee perfect sharing at all times. The magnitude of deviations is an indicator of the quality of the scheduler.

- various quality metrics including

$$\text{B-WFI} = \max_{k,\Delta t} [\Phi_k W(\Delta t) - W_k(\Delta t)]$$

- in the best possible Packet System (e.g. WF^2Q), B-WFI = 1 MSS (*Optimal B-WFI*);
- tradeoff between guarantees and complexity:
  Xu-Lipton 2002: optimal B-WFI requires $\Omega(\log N)$ time;
  Valente 2004: an $O(\log N)$ version of WF^2Q;
- breaking the $O(\log N)$ barrier implies relaxed guarantees.

State of the art of fast schedulers

- Priority-based schedulers are fast but give no guarantees except to the flow with highest priority;
- Round Robin schedulers have $O(1)$ time but poor guarantees ($O(N)$ B-WFI);
- some *timestamp-based* schedulers such as WF^2Q give optimal service guarantees in $O(\log N)$ time;
- approximated variants of timestamp-based schedulers (KPS - Karsten 2006; GFQ - Stephens,Bennet,Zhang 1999) have near-optimal guarantees and $O(1)$ time complexity (but several times slower than RR).

Our result

QFQ is a practical O(1) approximated timestamp-based scheduler with

- near-optimal guarantees (B-WFI $\sim$5 MSS);
- truly constant complexity, independent of number of flows and configuration parameters;
- uses very simple CPU instructions;
- 110 ns/pkt on common workstations, compared to 55 ns for DRR and 400 ns for KPS.

Fair Queueing in software (or inexpensive hardware) is feasible at GBit/s rates.

QFQ overview

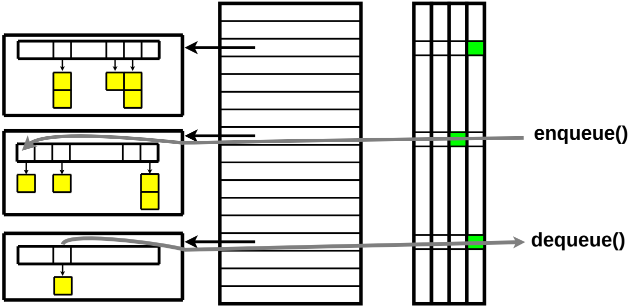
QFQ operates as other timestamp-based schedulers:

- track the behaviour of a Fluid System;
- for each packet, compute *Virtual* Start and Finish times;
- schedule in Finish time order among packets that are i) available and ii) already started in the Fluid Server (*Eligible*)

The sorting steps imply a $O(logN)$ complexity.

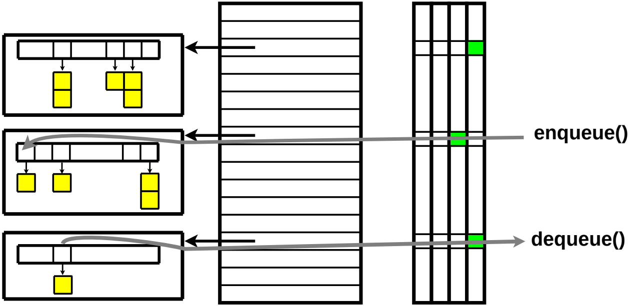
- use approximated sorting to reduce complexity;
- use careful approximations to preserve guarantees;
- use extra data structures to reduce constants.

QFQ data structures



- Approximated sorting based on rounded timestamps and splitting flows into a constant number of groups;
- flow i belongs to group $\lceil \log_2 L_i/\Phi_i \rceil$;
- rounding intervals grow exponentially.

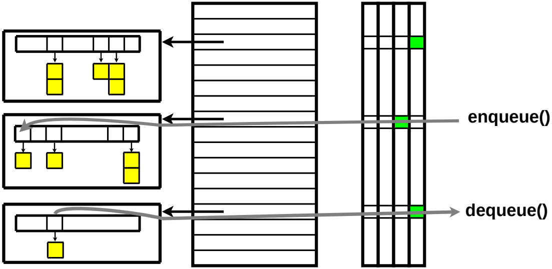
QFQ data structures – sorting



- Use approximate timestamps for sorting, but keep exact values internally;
- within each group, there is only a finite number of slots, so we can use bucket sort;
- for selection purposes, use same $(F - S)$ for all flows in a group, so the order on F and S is the same.

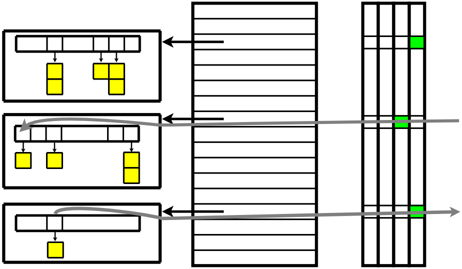
QFQ data structures – selection (1)

The goal is to select the eligible flow with smallest F.



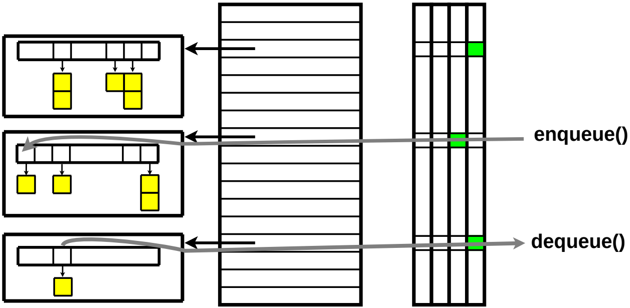
- GFQ needs to iterate on groups to find the candidate, hence $O(G)$ complexity;
- QFQ organizes groups into four Sets, such that group index reflects the Finish time order;
- one of the groups contains all interesting candidates, so a single FFS instruction replaces the scan.

QFQ data structures – selection (2)



- partitioning is done on *Eligibility* and *Readyness* (groups that violate the ordering are put in a different set);
- on packet arrivals, finding the right set for a group requires only one FFS instruction;
- on packet departure, moving multiple groups between sets is also done without loops, using MASK/AND/OR ops.

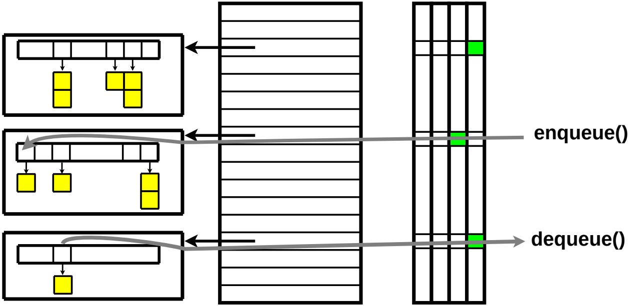
QFQ – enqueue



Nothing to do if flow is already backlogged. Otherwise:

- bucket-insert the flow in its group;
- update group state;
- put the group in the correct set.

QFQ – dequeue



- locate first bit set in ER;
- serve the head flow in the corresponding group;
- possibly put the flow in a new slot;
- update group state;
- move groups between sets, due to changes in Virtual time and Readiness.

Service guarantees

Service guarantees for QFQ:

$$\text{B-WFI}^k = 3\phi^k\sigma_i + 2\phi^k L$$

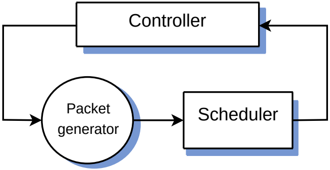
(remember that $L^k/\Phi_k < \sigma_i \leq 2L^k/\Phi_k$)

$$\text{T-WFI}^k = \left(3\left\lceil\frac{L^k}{\phi^k}\right\rceil + 2L\right)\frac{1}{R}$$

(R is the link's rate).

Experimental results

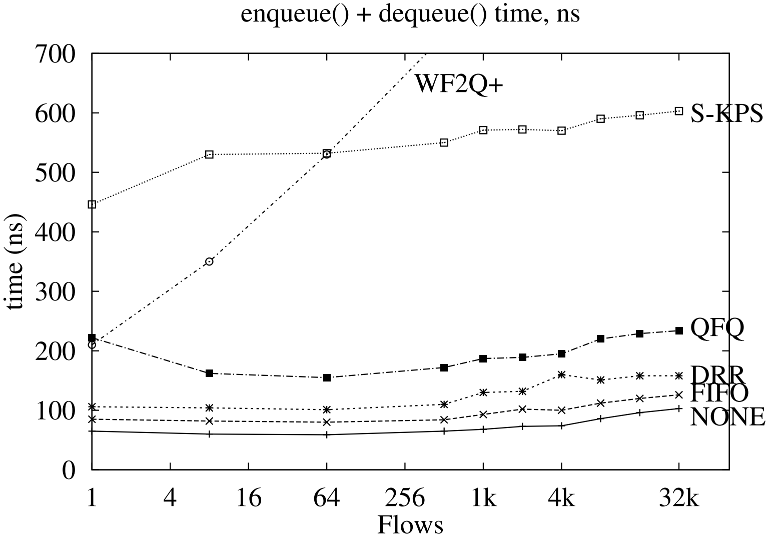
Measurements taken by running the kernel code in userspace:



- generate traffic for a programmable number of flows, packet size and weight distribution;
- carefully control the operating point of the scheduler;

```
./test -alg rr -qmin 4n -qmax 30n -flowsets 1::512,8::64
dn_rr    n 5004288 10000000 time 0.683968  136.676
./test -alg qfq -qmin 4n -qmax 30n -flowsets 1::512,8::64
dn_qfq   n 5004288 10000000 time 0.974142  194.661
./test -alg kps -qmin 4n -qmax 30n -flowsets 1::512,8::64
dn_kps   n 5004288 10000000 time 2.855963  570.703
```

Performance comparison – scalability



Mixed workloads

Measurement results in ns for an enqueue()/dequeue() pair and packet generation. Standard deviations are within 3% of the average.

Flows	NONE	FIFO	DRR	QFQ	KPS	WF2Q+
1	62	83	105	**221**	450	210
8	60	80	102	**163**	543	344
64	59	80	100	**158**	540	526
512	64	85	110	**175**	560	740
4k	74	102	157	**197**	590	1110
32k	62	117	147	**222**	601	1690
1:32k,2:4k,4:2k,8:1k,128:16,1k:1 flows						
mix	92	119	160	**255**	612	1715

Conclusions

- QFQ is a Timestamp-based scheduler with near optimal service guarantees and true $O(1)$ run time;
- 110 ns/pkt, only 2 times slower than DRR, and 4 times faster than comparable algorithms;
- already available as part of dummynet, together with several other schedulers: `http://info.iet.unipi.it/~luigi/dummynet/`
- technical report and code at `http://info.iet.unipi.it/~luigi/qfq/`
- soon available as a Click module.

Future work

Future work:

- detailed performance analysis on low-end hardware (OpenWRT platforms);
- identify performance bottlenecks, memory access patterns;
- investigate feasibility of hardware implementations (including NETFPGA).