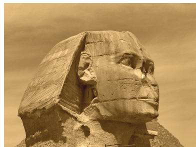
Providing Near-Optimal Fair-Queueing Guarantees at Round-Robin Amortized Cost

Paolo Valente

Department of Physics, Computer Science and Mathematics
Modena - Italy

UNIMORE
UNIVERSITÀ DEGLI STUDI DI MODENA E REGGIO EMILIA

Workshop PRIN SFINGI – October 2013



Contributions

- A **modification scheme** for *fair-queueing* packet schedulers
  - To reduce amortized computational cost
- A scheduler obtained by applying this scheme to the Quick Fair Queueing (QFQ) Linux scheduler
  - **Quick Fair Queueing Plus** (**QFQ+**)
    - Replaced QFQ in Linux from 3.7

Motivation

- A **very efficient** fair-queueing packet scheduler exists: Deficit Round Robin (**DRR**)
- But it suffers from high packet delay and jitter
  - With respect to a perfectly fair, and smooth, ideal service
- Fair-queueing schedulers providing much better service guarantees exist
  - Optimal (i.e., minimum possible), or near-optimal, deviation from ideal service
  - But even **the fastest** of them, **QFQ**, is at least **twice as slow as DRR**

Demos

- Instead of diagrams and further explanations
  - QFQ+ in action ...
- Downside
  - Results shown imprecisely and incompletely
- Notes
  - QFQ+ named QFQ, because in Linux it replaced QFQ, retaining its same name
  - QFQ not considered
    - Lives now only in older versions of Linux
    - Different environment
    - Hard to port to newer kernels

Computational cost 1/2

- Pair of VMs: a generator sending packets to a sink
- Host
  - Intel Core i7-2760QM, 4 cores, 6MB cache
  - 16 GB RAM
- Guests
  - 1 CPU, 2 GB RAM
- Bandwidth of output link on generator limited to 300Mb/s, to not saturate (virtual) CPU bandwidth

Computational cost 2/2

Service guarantees 1/2

Service guarantees 2/2

Unscheduled program ...

- In a virtualized environment
- With any pair of (existing) host-and-guest schedulers,
- and independently of how fast the code of, e.g., an interrupt handler is
  - the latency of the handlers in the guest
  - may be **thousands of times** as high as on bare metal
  - (it happens if the host is not idle)

Simple demo

- Realized in collaboration with Virtual Open Systems ...
- Two clips
    - Guest-latency statistics with an idle host
    - Guest-latency statistics with a non-idle host (e.g., executing some other service or VM)
- HW details
    - ARM Versatile Express CoreTile
    - Cortex-A15x2 TC2 @ 1.0 GHz

Any interest in collaborations?

- VoSys is a member of the ETSI NFV working group
  - Working on the
    - creation of a pool of partners (Vosys, system makers, operators, SoC's makers, universities ..)
    - to propose an SDN/NFV Proof of Concept

The end

Thanks for your attention

Questions?

Idea

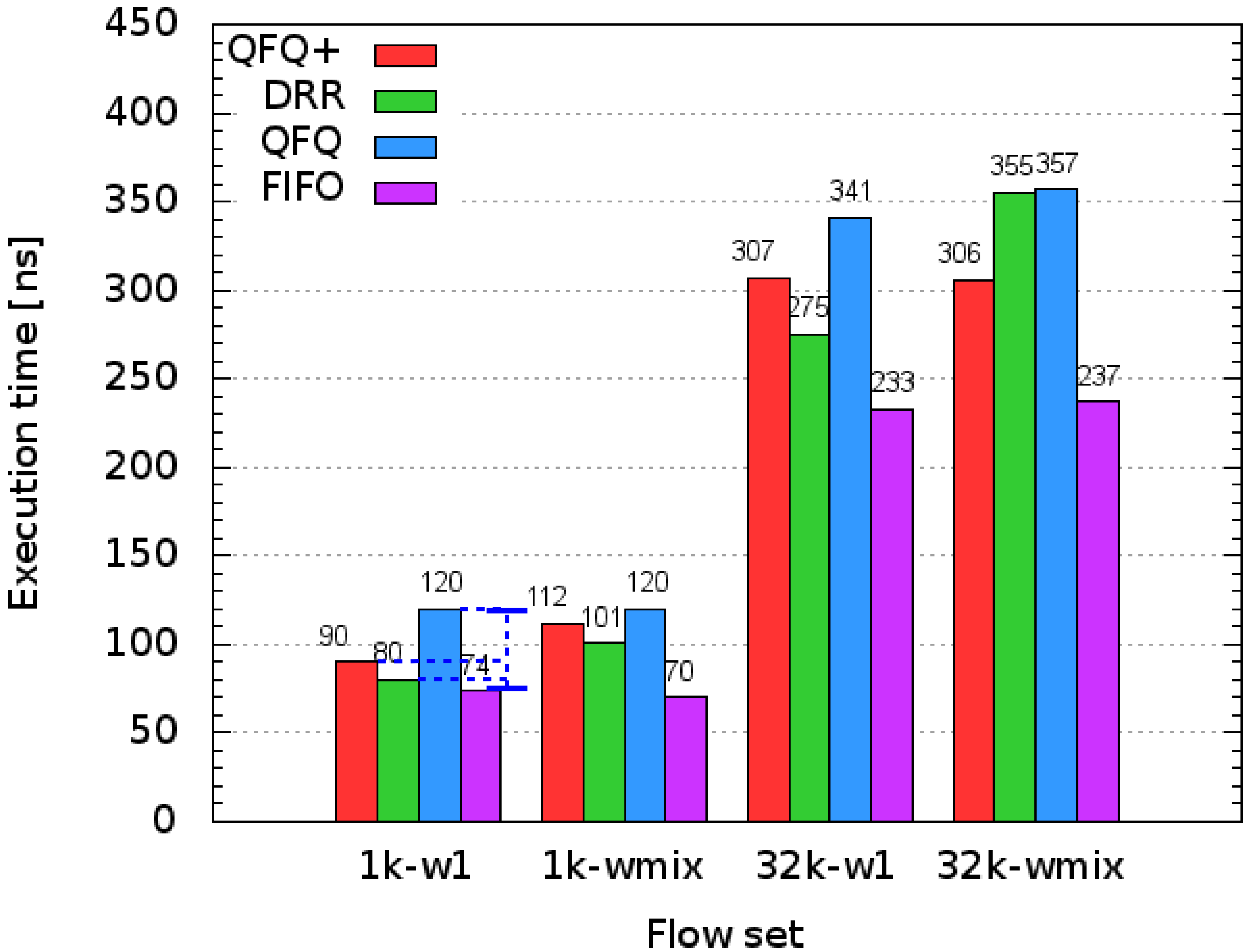
- Group packet flows into *aggregates*
- Use the costly operations of the original schedu-lers to schedule aggregates and not single flows
- To **reduce the frequency** of costly operations:
  - Serve each aggregate for a relatively long time
    - Proportional to the number of flows in the aggregate
- Inside aggregates, schedule flows with DRR
  - Cheap operations

Benefits and drawbacks

- The higher the number of flows in each aggregate is, ...
  - ..., the less frequently costly operations are executed during overload periods
    - Hence the closer the amortized execution time becomes to DRR
  - ..., the more the service of the modified schedulers deviate from the one of the original scheduler
    - Original per-flow guarantees are now provided to aggregates
      - Become coarser and coarser as the number of flows in each aggregate grows
    - Final, per-flow guarantees depend on DRR service properties

Computational cost: diagram

- For a more precise comparison, including also (the old) QFQ



Service guarantees: diagram

- For a more precise comparison, including also (the old) QFQ