TEMPEST: Test EnvironMent for Performance Evaluation of the Scheduling of packeTs

Carlo Augusto Grazia, Maurizio Casoni, Paolo Valente

Department of Engineering *Enzo Ferrari*
University of Modena and Reggio Emilia

Larnaca, Cyprus, October 8, 2014
Emergency Networks for PPDR Workshop
Part of IEEE WiMob 2014

Talk overview

Problem

what

to easily evaluate packet scheduling solutions

- execution time
- QoS guarantees
- throughput

why

emerging fields need packet scheduling

- 3G/4G, LTE, Emergency Networks
- Technologies QoS driven

how

TEMPEST

State of the Art

typical solution

simulated environment vs emulated environments

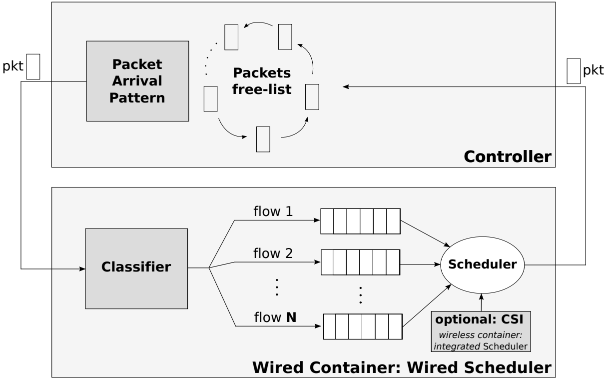
weaknesses

- simulators
  - **easy** to set up a suitable test environment
  - **hard** to import/export real code
  - time is simulated!
- emulators
  - **hard** to set up a suitable test environment
  - **easy** to import/export real code
  - packet generation, reception, device drivers and other costs dominate the measurements;

TEMPEST

Test EnvironMent for Performance Evaluation of the Scheduling of packeTs

UNIX-based open tool able to measure the *actual performance* of a packet scheduler under the desired operating conditions

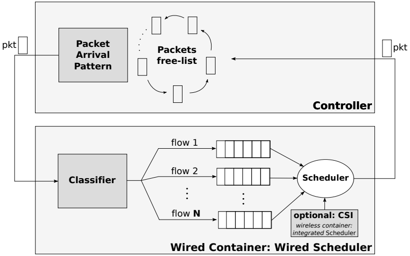


TEMPEST modules 1/3: The Controller

TEMPEST Controller

simulates the desired realistic packet arrival pattern

- manages a free list of fake packets
- uses fake packets to reduce the system overhead (40Mpps)
- controls the number of pending packets
- total low-level control of queues state

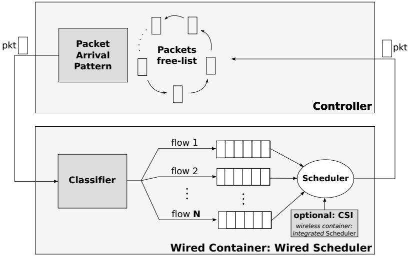


TEMPEST modules 2/3: The Container

TEMPEST Container

featherweight model which incapsulates a packet scheduler

- runs kernel code in user space
- easy code porting kernel ↔ tempest with trivial interface changes
- tracks information for QoS and throughput measurements

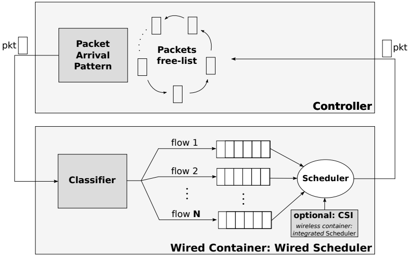


TEMPEST modules 3/3: Channel State Information (CSI)

TEMPEST CSI

the simulation engine behind the wireless scenario representation

- optional block
- used for wireless simulations
- gives channel feedback: SnR or P_{loss}
- cross-layering solution



TEMPEST details

Core

TEMPEST gives a fine-grained level of configuration parameters in a hybrid simulation/emulation tool

Keypoints

From the **emulation** side:

- time is real
- code is real too

From the **simulation** side:

- QoS metrics are simulated
- Throughput and CSI are simulated
- simulated measures are exact!

TEMPEST default schedulers

- **DRR**: $\mathcal{O}(1)$ time complexity, $\mathcal{O}(n)$ deviation from optimal service
- **WF^{2}Q+**: a *timestamp-based* algorithm with optimal service guarantees in $\mathcal{O}(logn)$ time
- **KPS**: approximated timestamp-based schedulers with near-optimal guarantees and $\mathcal{O}(1)$ time complexity (slower than DRR)
- **QFQ**: approximated timestamp-based schedulers with near-optimal guarantees and $\mathcal{O}(1)$ time complexity (as fast as DRR)
- **QFQ+**: improvement of QFQ sometimes faster than DRR
- **W^{2}F^{2}Q**: integrated packet scheduler for wireless link based on WF^{2}Q+ algorithm
- **HFS**: a modular packet scheduler for wireless link, based on QFQ+, with $\mathcal{O}(1)$ time complexity, quasi-optimal service guarantees, high throughput and low energy consumption

TEMPEST input parameters

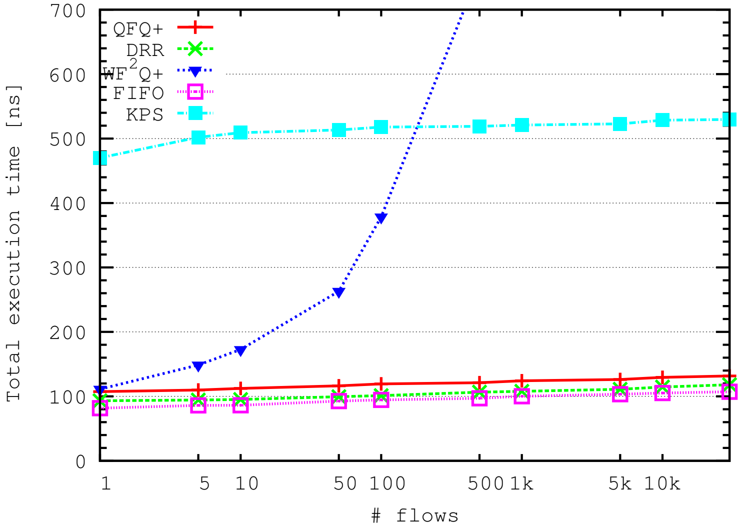
	parameter	short description	default value
	`n`	amount of events for the run	null
controller	`qmin`	min controller's pending packets	0
controller	`qmax`	max controller's pending packets	0
controller	`len`	length of packets in byte	1700
controller	`burst`	set predefined packet arrival patters	0
controller	`qsing`	minimum packets number per flow	0
container	`mode`	choose the container	null
container	`alg`	scheduler algorithm	FIFO
container	`qos`	QoS scheduler algorithm	null
container	`mac`	MAC scheduler algorithm	null
container	`flowsets`	define the QoS flows characteristics	null
container	`flows`	define the number of QoS flows	0
container	`flowsetsmac`	define the MAC flows characteristics	null
container	`flowsmac`	define the number of MAC flows	0
container	`qmac`	define the Q shared-buffer size in packets	0
CSI	`wdistr`	define MAC weight distribution	0
CSI	`ploss`	assign a packet loss for each MAC flow	null
	`intgr_th`	define the good/bad threshold	50%

Choosing the right scheduler

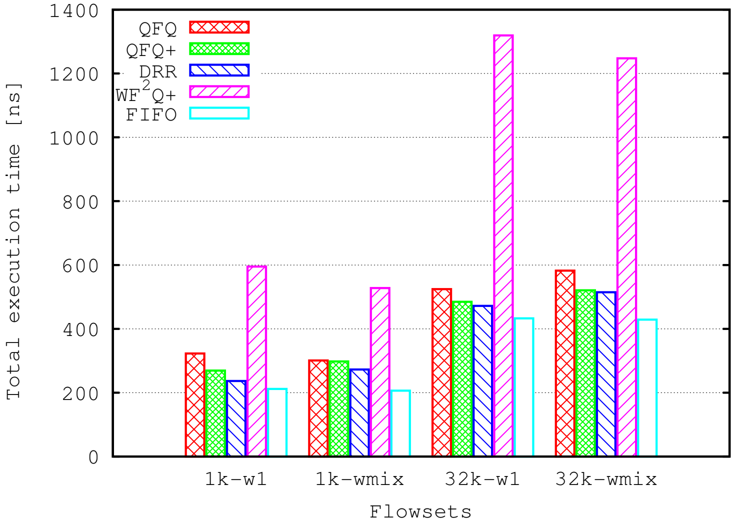
We have many algorithms with different features. How do we choose?

- it depends on the operating conditions:
  - for large N, asymptotic complexity is important.
  - for small N, or certain weight distributions, guarantees or *actual* run times are more important
- theory can tell us about worst-case service guarantees and asymptotic complexity
- we need measurements to determine the run-time constants

Execution time behavior for different schedulers (lower is better)



Execution time with *bursty* packets arrival pattern (lower is better)

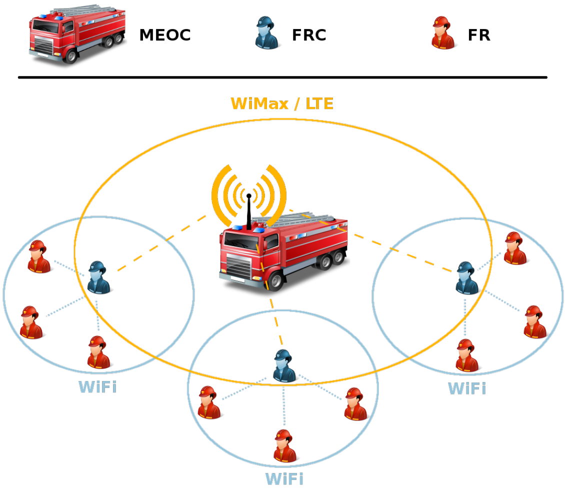


Implementing novel solutions in challenging scenarios

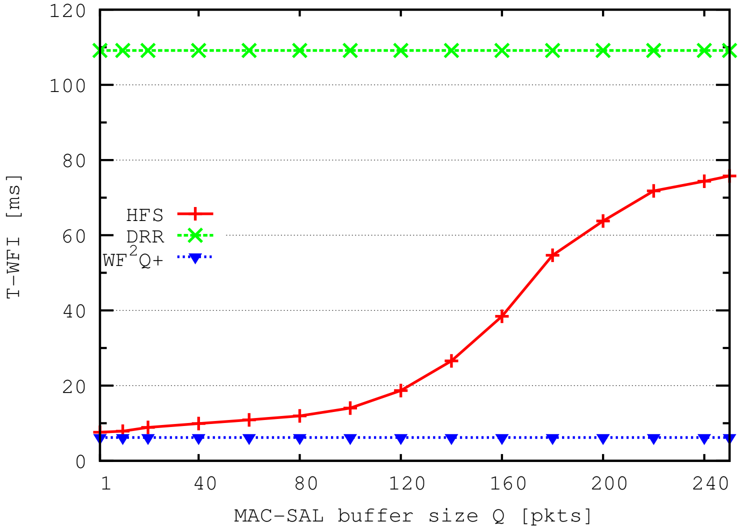
We have several novel technologies QoS driven:

- quickly evaluate novel scheduling solution
- real code is basically the TEMPEST code
- PPDR systems are a perfect challenging example:
  - technologies/architectures used are still evolving
  - thin QoS guarantees on a tough environment

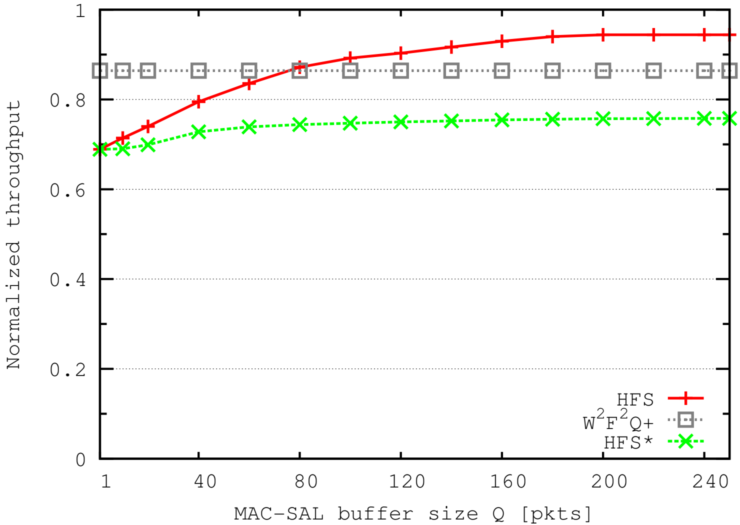
PPDR case study



PPDR case study: T-WFI for different scheduler (lower is better)



PPDR case study: throughput achieved (higher is better)



Conclusions

TEMPEST

a novel test environment for packet scheduling evaluation/implementation

Characteristics

- open and UNIX based
- hybrid simulator-emulator
- measures all the main figures of merit
- flexible and suitable for PPDR systems and challenging environments
- tests proof its accuracy
- support novel technologies/standards
- help research!

thank you
for your attention

carloaugusto.grazia@unimore.it

extra slides

Reference Scenario

- 20 first responders (FR)
- link rate 54 Mb/s
- one MAC-SAL flow per FR
- MAC-SAL flow packet loss probability
  - ranging linearly from 10^0 to 10^{-1}
  - outsider values as 10^{-2}, 10^{-3} and 10^{-4}
  - static
- MAC-SAL flow weight distribution
  - analogical: $\phi_k = (1 - \mathsf{P}_{loss_k}) \cdot 1000$
- 100 QoS flows with different weights