Deadline scheduling in the Linux kernel

Juri Lelli, Claudio Scordino, Luca Abeni and Dario Faggioli

Benno Fünfstück

July 8, 2019

Story: A new coffee machine

Components:

- Brewing controller
- User interface controller
- Web interface

Story: A new coffee machine

Components:

- Brewing controller
- User interface controller
- Web interface

⇒ real-time tasks
(guaranteed worst-case response time)

Linux for real-time

First approach: run Linux as a task in real-time hypervisor

- examples: RTAI, RTLinux, Xenomai
- maintenance of HAL and microkernel for real-time
- custom tools and API necessary for real-time part

Linux for real-time

First approach: run Linux as a task in real-time hypervisor

- examples: RTAI, RTLinux, Xenomai
- maintenance of HAL and microkernel for real-time
- custom tools and API necessary for real-time part

Second approach: make the Linux kernel itself suitable for real-time

- PREEMPT_RT patchset

Linux for real-time

First approach: run Linux as a task in real-time hypervisor

- examples: RTAI, RTLinux, Xenomai
- maintenance of HAL and microkernel for real-time
- custom tools and API necessary for real-time part

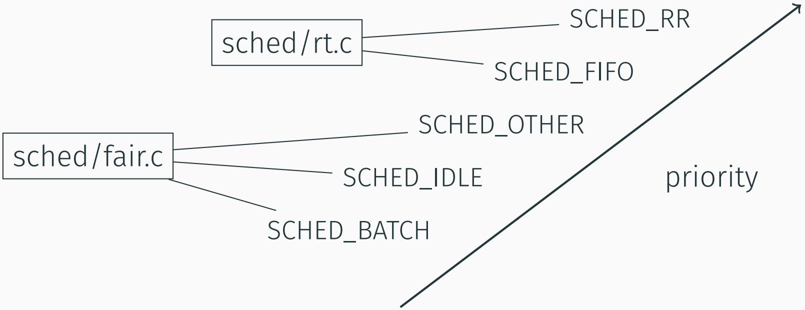
Second approach: make the Linux kernel itself suitable for real-time

- PREEMPT_RT patchset
- real-time scheduler

Design of a realtime scheduler

Linux modular scheduling framework

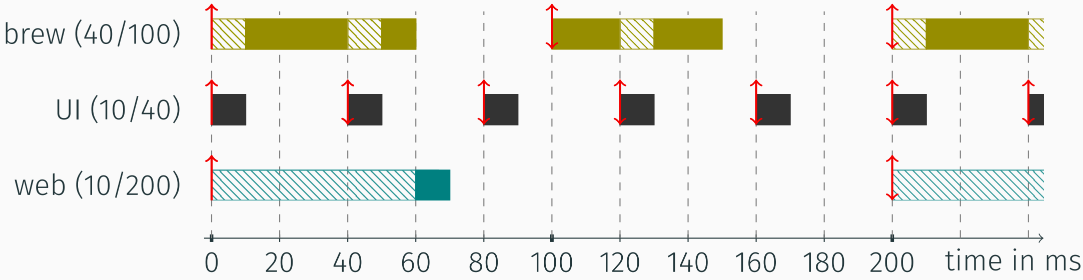
Since Linux 2.6.23



Real-time Tasks

Properties of real-time tasks: *runtime* and *period*

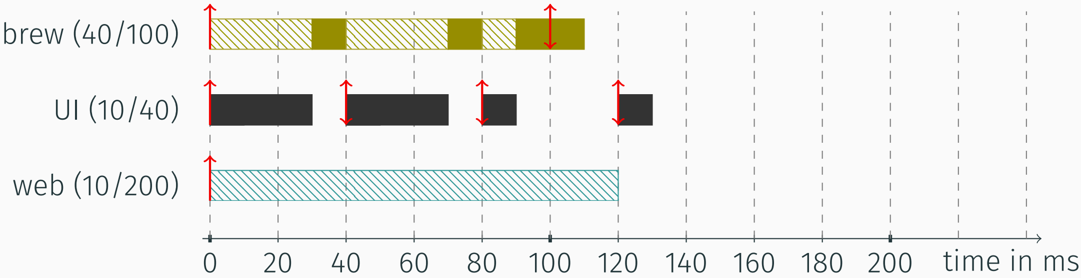
For our coffee machine:



Real-time Tasks

Properties of real-time tasks: *runtime* and *period*

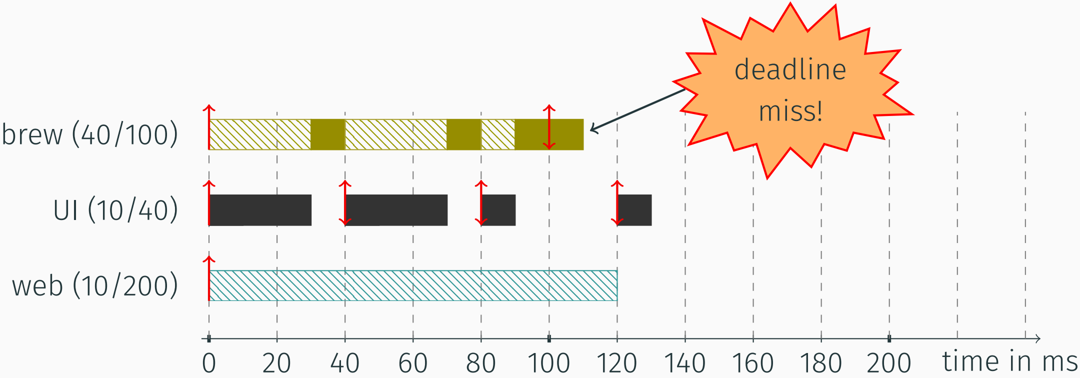
For our coffee machine:



Real-time Tasks

Properties of real-time tasks: *runtime* and *period*

For our coffee machine:

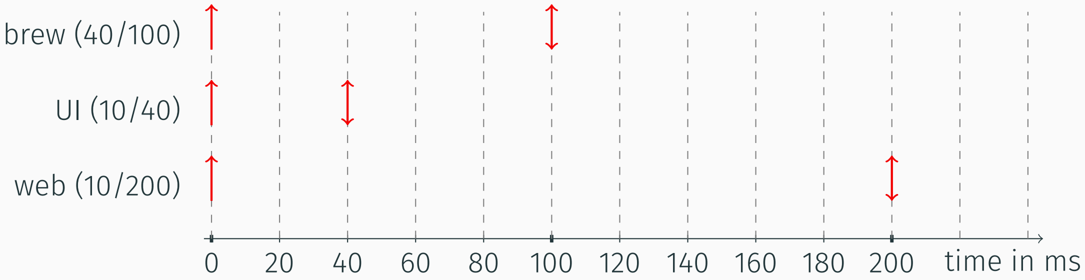


Related Work

- Linux/RK: resource reservation, CPU (fixed priority) and disk (EDF)
- OCERA: resource-reservation based scheduler, as loadable kernel module
- LITMUS^{RT}: real-time scheduling testbed, not aiming to be production quality
- ExShed: kernel extension to allow scheduler implementation in user space
- SCHED_SPORADIC: priority based, aimed for inclusion in mainline but failed

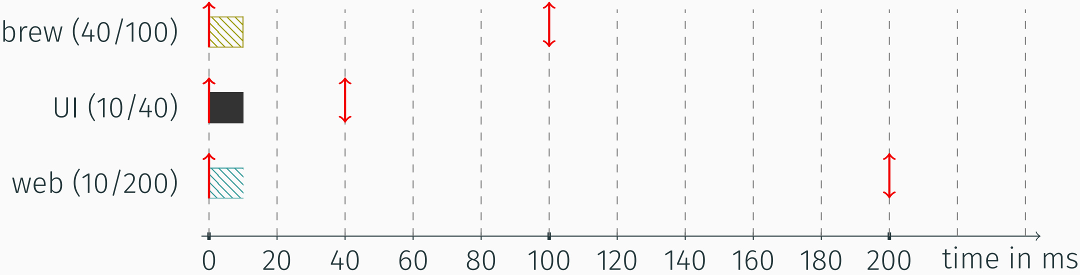
Example

	brew	UI	web
budget	40	10	10
deadline	100	40	200



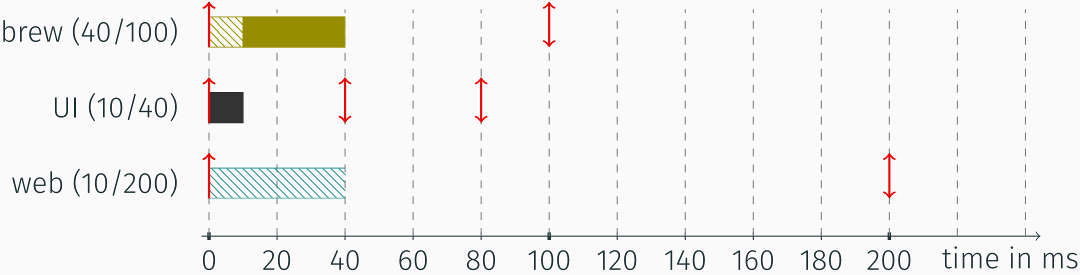
Example

	brew	UI	web
budget	40	0	10
deadline	100	40	200



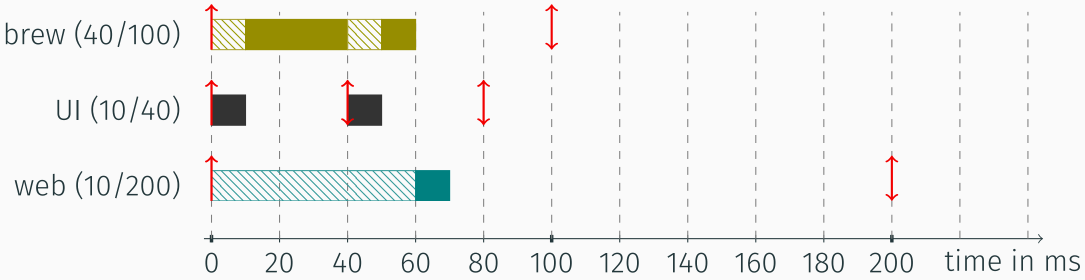
Example

	brew	UI	web
budget	10	10	10
deadline	100	80	200



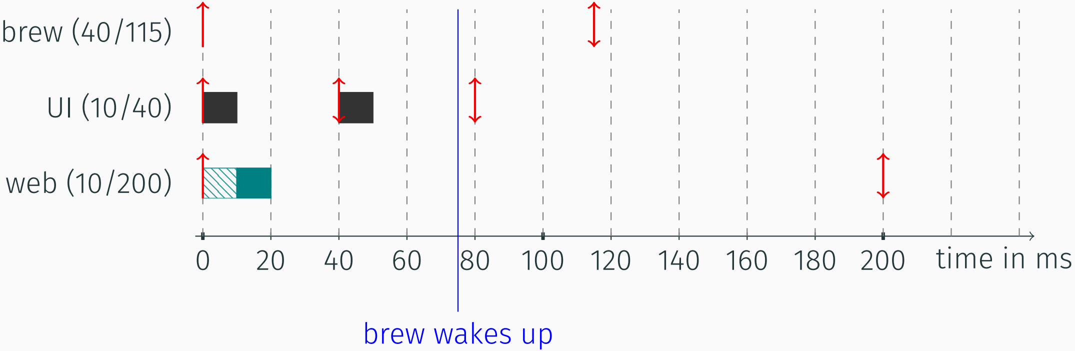
Example

	brew	UI	web
budget	0	0	0
deadline	100	80	200



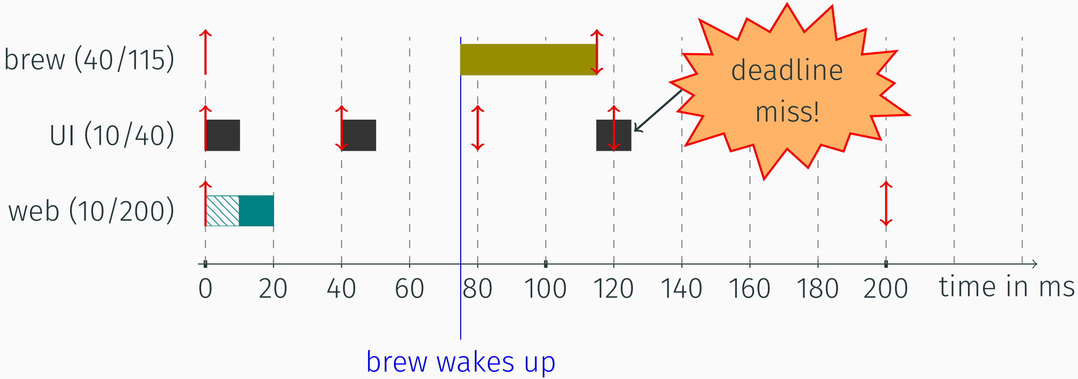
Example 2

	brew	UI	web
budget	40	0	0
deadline	115	80	200



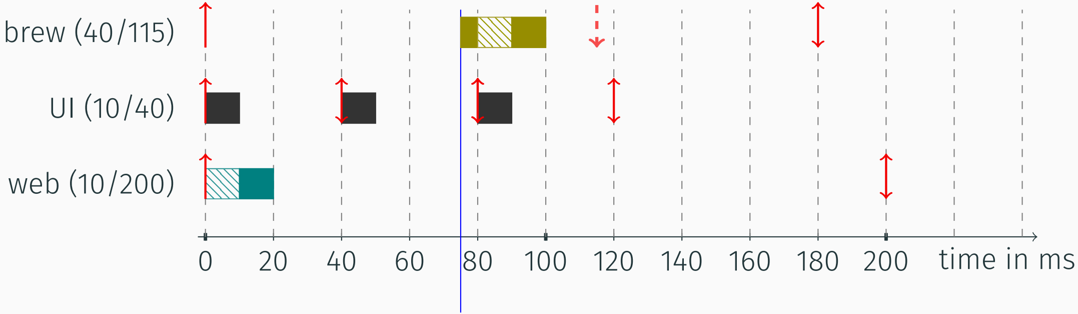
Example 2

	brew	UI	web
budget	0	10	0
deadline	115	120	200



Example 2

	brew	UI	web
budget	40	10	0
deadline	180	120	200



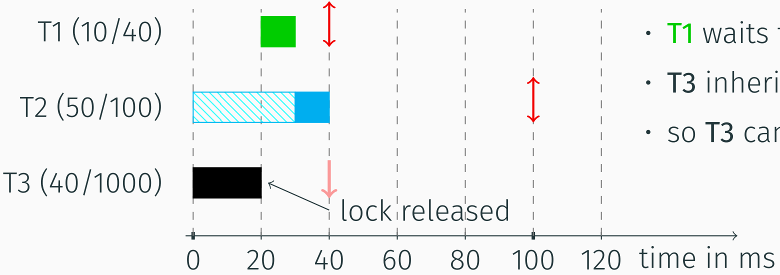
brew wakes up, new deadline generated

because $40 > (115 - 75) * (40/115) = 13.9$

API, shared resources, multicore

User level API: two new syscalls `sched_setattr` and `sched_getattr`

Shared resources: inherit deadline of blocked task



- T1 waits for lock hold by T3
- T3 inherits deadline of T1
- so T3 can preempt T2

Multicore: partitioned (via cpuset) and global supported

Evaluation

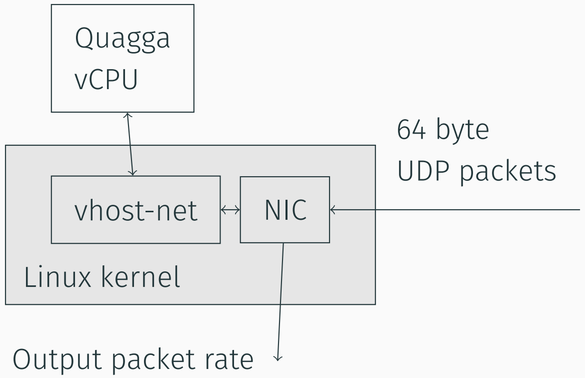
Synthetic workload (partitioned)

U(%)	SCHED_DEADLINE(%)	SCHED_FIFO(%)	SCHED_OTHER(%)
60	0	0	0.58
70	0	0	1.87
80	0	0.003	6.03
90	0	0.38	10.20

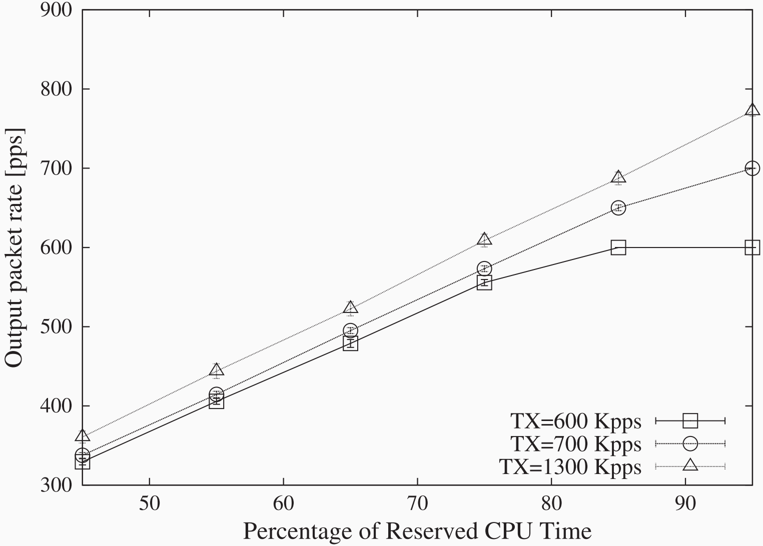
Synthetic workload (global, six cores)

U(%)	SCHED_DEADLINE(%)	SCHED_FIFO(%)	SCHED_OTHER(%)
500	0.027	0.001	3.303
510	0.023	0.002	4.310
520	0.051	0.011	4.992
530	0.099	0.023	6.046
540	0.138	0.230	7.093
550	0.239	0.271	8.097
560	0.289	0.380	9.977
570	0.351	0.640	11.554
580	0.618	1.380	15.384
590	1.295	2.535	19.774

Virtual router



Limit vhost-net thread



Conclusion

Still some work left:

- M-BWI: Multiprocessor Bandwidth Inheritance
- Power aware algorithms (example: GRUB-PA)
- Support cgroups interface

Not rocket science, but solid implementation of proven concepts (EDF, CBS)

… ready for production use

… upstream in the Linux kernel

… with simple to use API

We can run our coffee machine on stock linux! :)

Algorithm

For each Task (Q_i / T_i), keep track of:

- budget (remaining runtime) q_i and
- scheduling deadline d_i

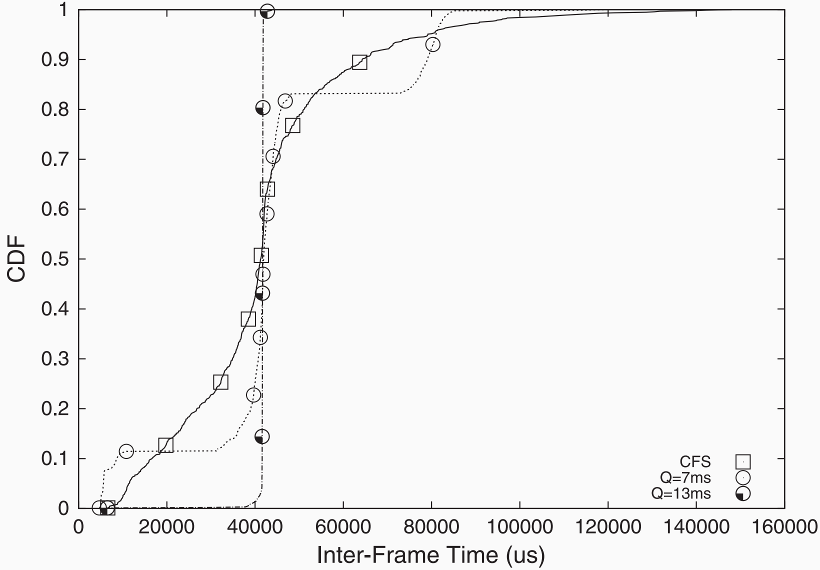
When a task wakes up:

- if $q_i \geq (d_i - t_i)(Q_i/T_i)$, recharge budget and set new deadline (one period)

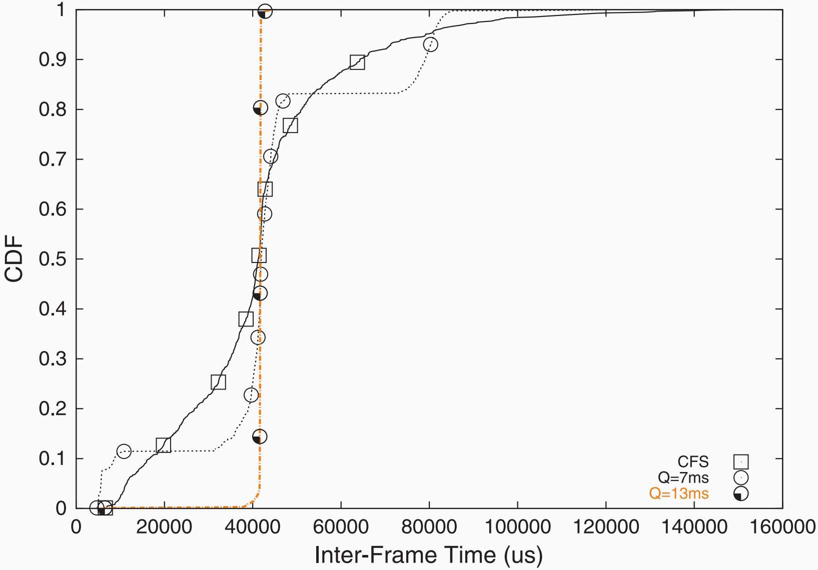
Always run task with earliest deadline, decrease budget accordingly

In single-CPU case, all deadlines are hit if sum of Q_i/T_I is less than 1

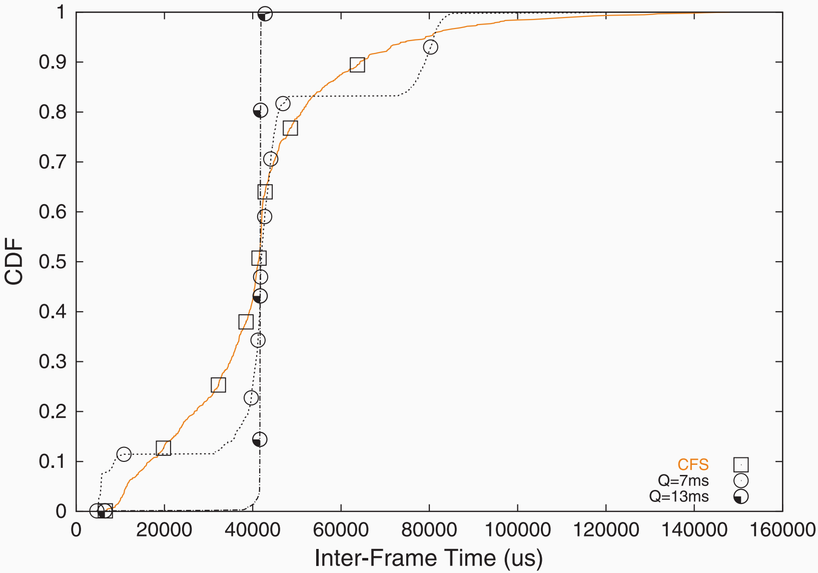
MPlayer on loaded system



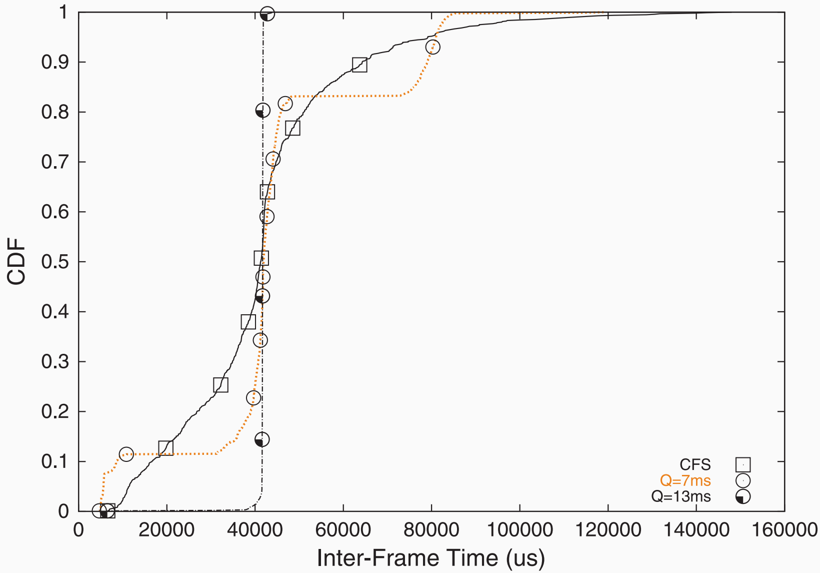
MPlayer on loaded system



MPlayer on loaded system



MPlayer on loaded system


CDF
Inter-Frame Time (us)
CFS
Q=7ms
Q=13ms