SELF-TUNING SCHEDULERS FOR LEGACY REAL-TIME APPLICATIONS

Tommaso Cucinotta, Fabio Checconi, Luca Abeni, Luigi Palopoli

Motivations

General-Purpose Operating Systems

- Very effective for storing & managing multimedia contents
- Designed for
  - **average-case** performance
  - serving applications on a **best-effort** basis
- They are <u>not</u> the best candidate for serving *real-time applications* with **tight timing constraints**
  - nor for **real-time multimedia**

Motivations

Overcoming limitations of a GPOS for multimedia

- **Large buffers** used to compensate *unpredictability*
  - ==> poor real-time interactivity and no low-latency multimedia
- **One-application one-system** paradigm
  - For example, for low-latency real-time audio processing (jack), gaming, CD/DVD burning, etc...
- **POSIX real-time extensions**
  - Priority-based, no temporal isolation
  - Not appropriate for deploying the multitude of (soft) real-time applications populating the systems of tomorrow

Motivations

Recent developments in GPOS CPU scheduling

- EDF-based **real-time scheduling** with *temporal isolation*
  - Better utilization of resources
  - Independent applications are isolated from the temporal perspective
- Various APIs for accessing the enhanced functionality
  - For example, the FRSH API from frescor
  - For example, the API from aQosa
- They require modifications of the applications
  - at the source-code level
- What about **legacy multimedia** applications ?

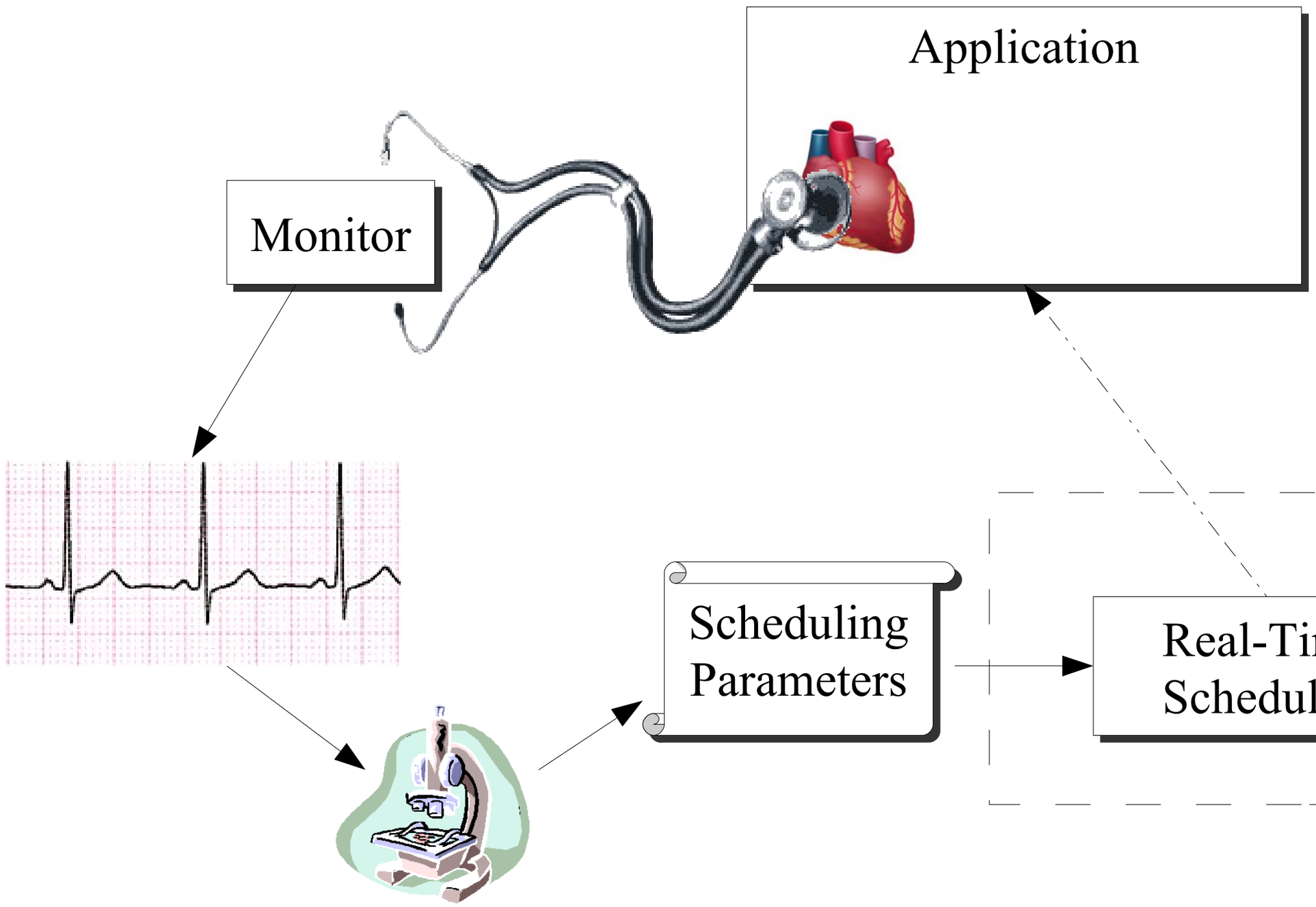
Research objectives

This research aims to

- allow (legacy) real-time periodic applications
- to benefit of real-time scheduling facilities increasingly available on a GPOS
- **without any change** in the application source-code

Proposed approach

Proposed approach – LFS++



Proposed approach

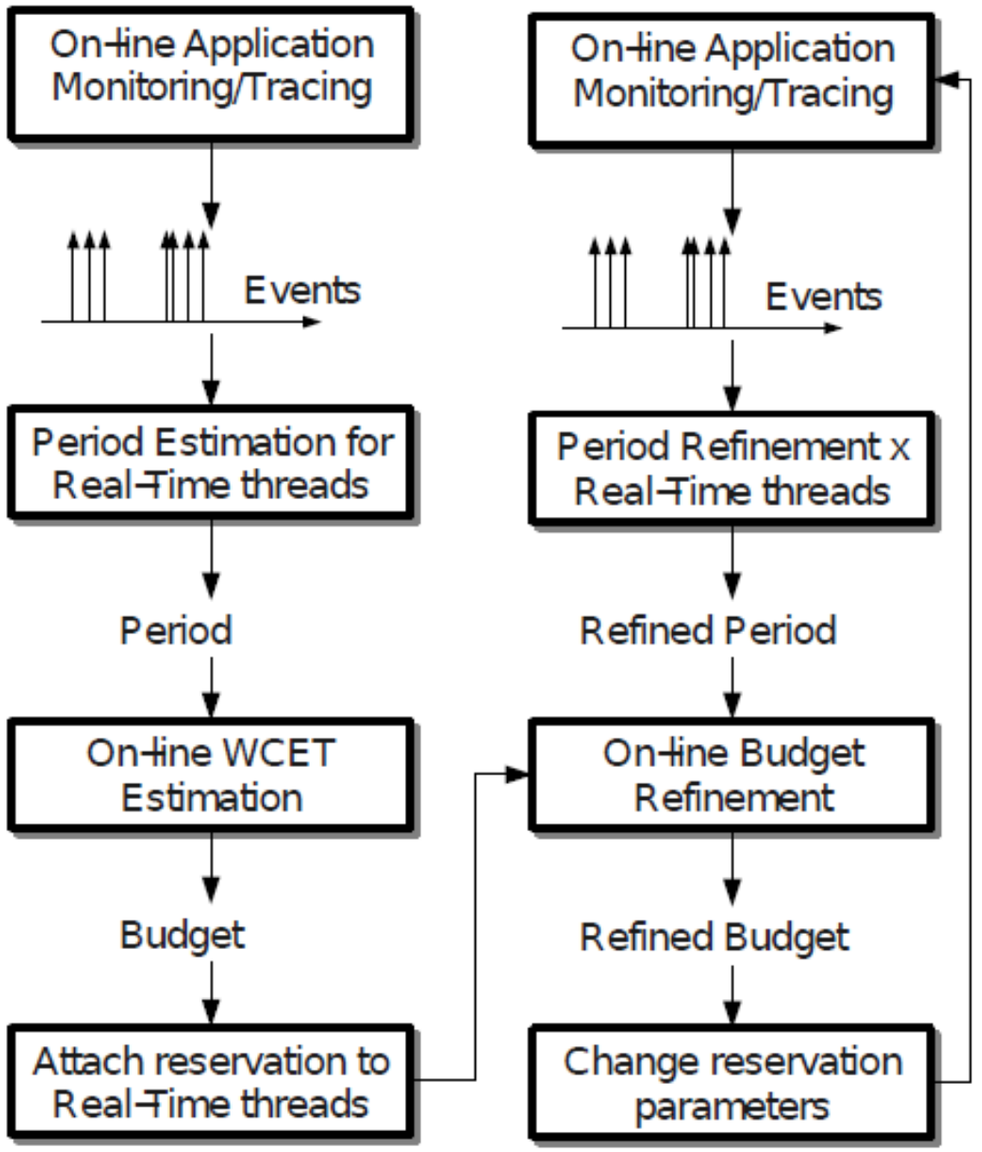
Legacy Feedback Scheduling (LFS++)

- An appropriate **tracing mechanism** observes the application, *inferring main parameters* affecting a (periodic) multimedia application temporal behaviour:
  - *job execution time*
  - **period**
- Scheduling guarantees are **automatically** provisioned by the OS, according to proper **scheduling parameters**
  - Based on sound arguments from real-time theory

Proposed approach

Comprehensive view

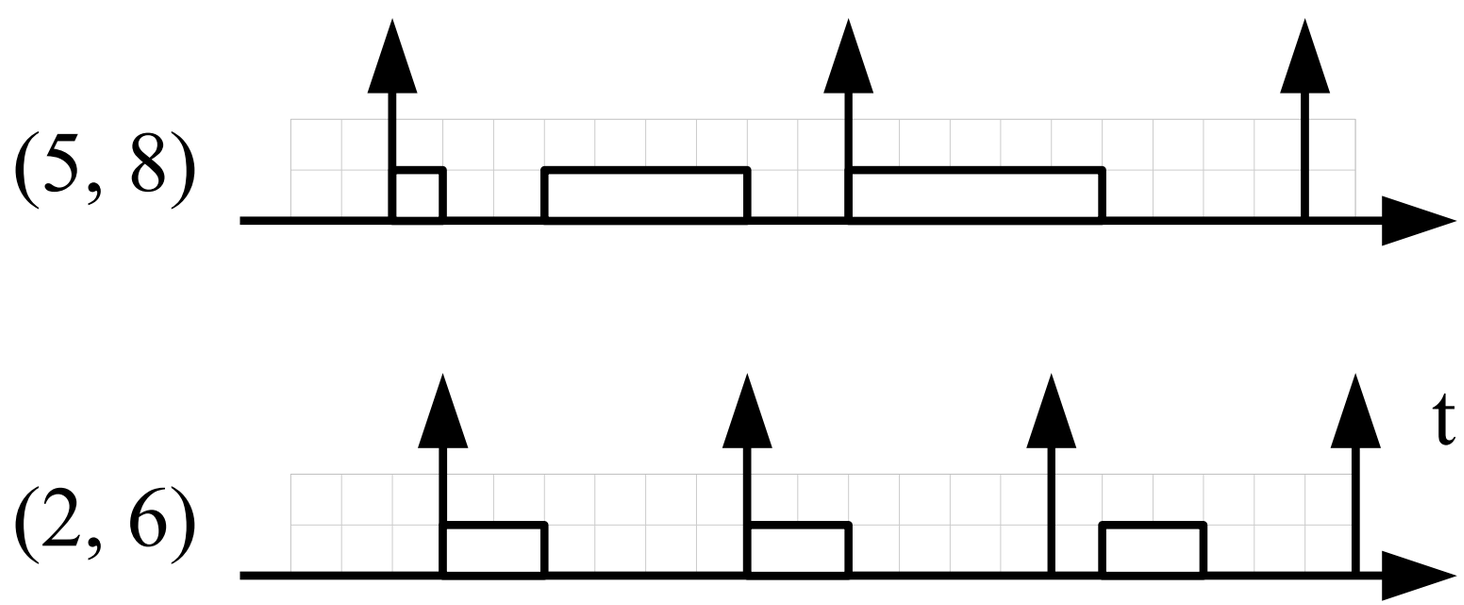
- Application tracing
- Period estimation (events analysis)
- On-line WCET estimation
- Automagic provisioning of scheduling guarantees



Real-time theory

Reservation-based scheduling: (Q_i, P_i)

- "Q_i time units **guaranteed** on CPU every P_i time units"



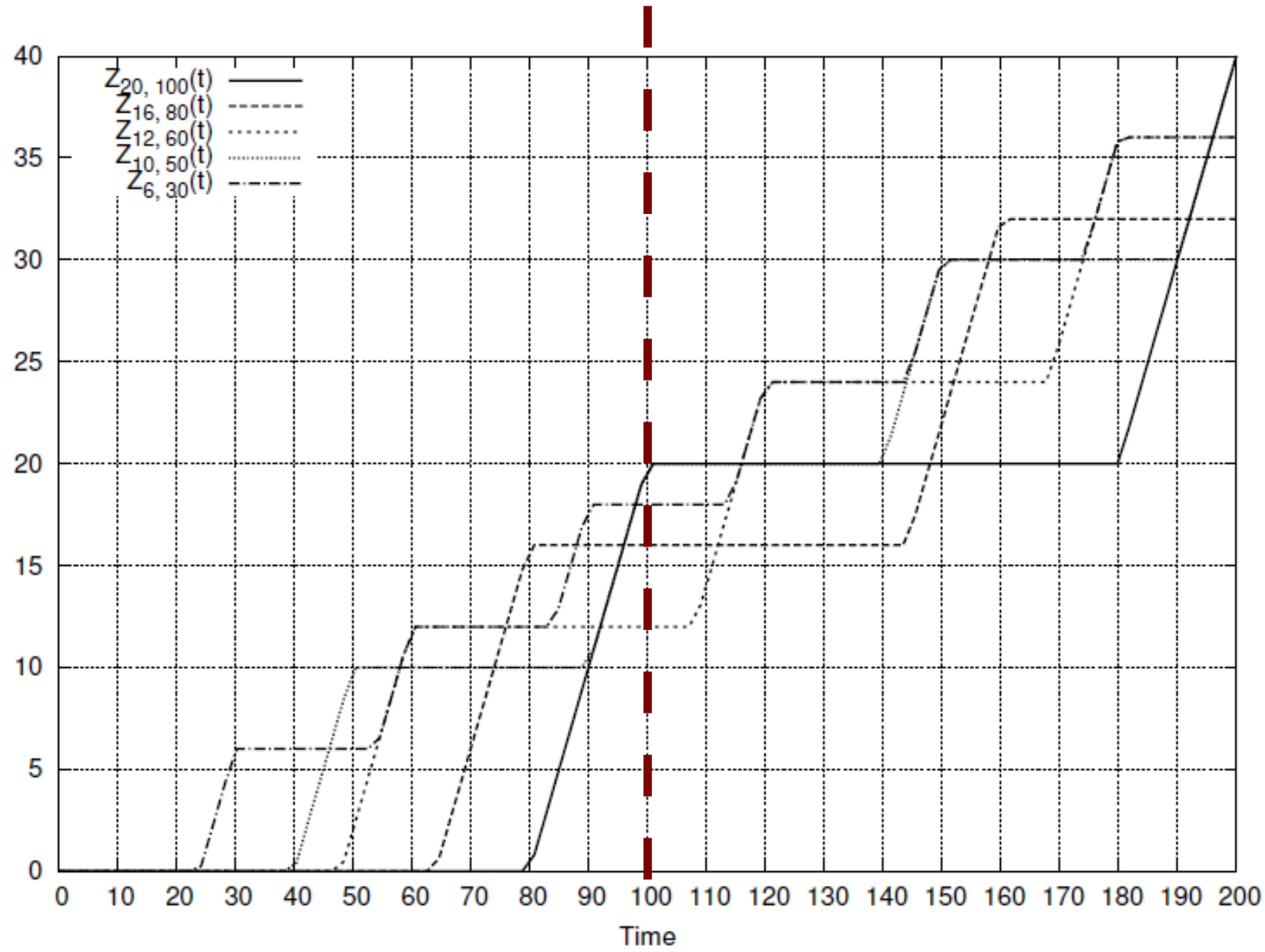
Real-time throttling on Linux is different

- Only constraint to "no more than Q_i every system-wide P"

Real-time theory

Known results from the real-time theory

- Not all (Q, P) with a given Q/P are equal
- Highlighted by the shown supply-bound function

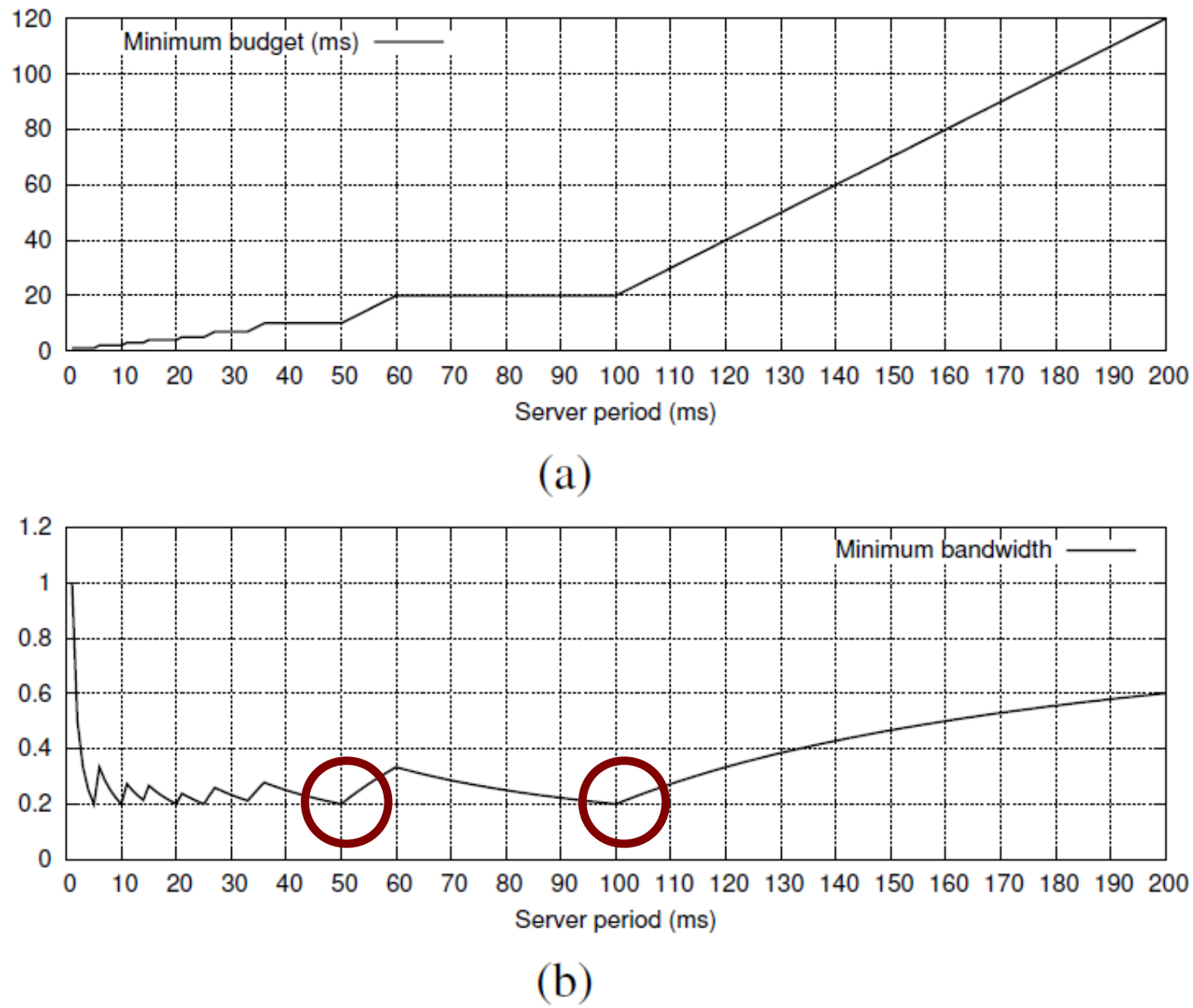


Real-time theory

What are the reservation parameters needed for correctly scheduling a real-time periodic task with assigned WCET and period ?

The minimum reservation utilization is achieved

- when the **reservation period equals the task period**
  - or any integer sub-multiple



Core problem

How to detect the application period ?

- Of a legacy real-time application (no source-code availability, no modifications)

Proposed approach

- Tracing the application behaviour *at run-time*
- For the purpose of identifying "**periodicity patterns**"

Legacy application tracing

Tracing legacy multimedia application

- Tracing **what** the application does
  - What system calls it periodically calls
    - Waiting, reading time information, posting timers, disk operations, network operations, etc...
- Tracing **how** the application behaves temporally
  - When it suspends and resumes

Results obtained with system-call tracing

Legacy application tracing

Available mechanisms

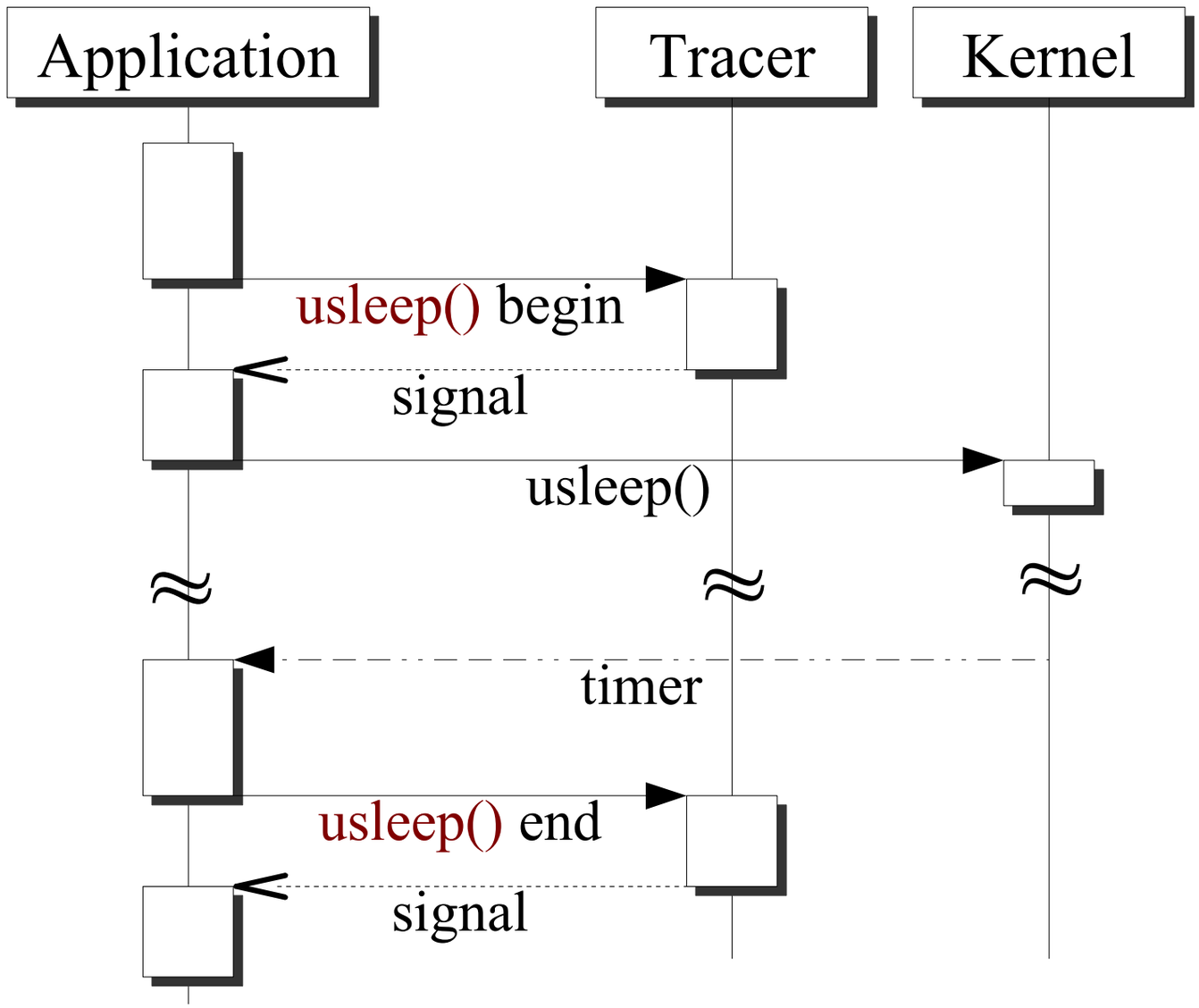
- strace Linux utility
  - Designed as debugging helper, too much overhead due to the generation of unneeded data
- ptrace() system call
  - Allows an external process to trace the execution of a target process, forcing it to stop at each entry and exit of a system call

We developed qtrace

- A low-overhead kernel-level system-call tracer

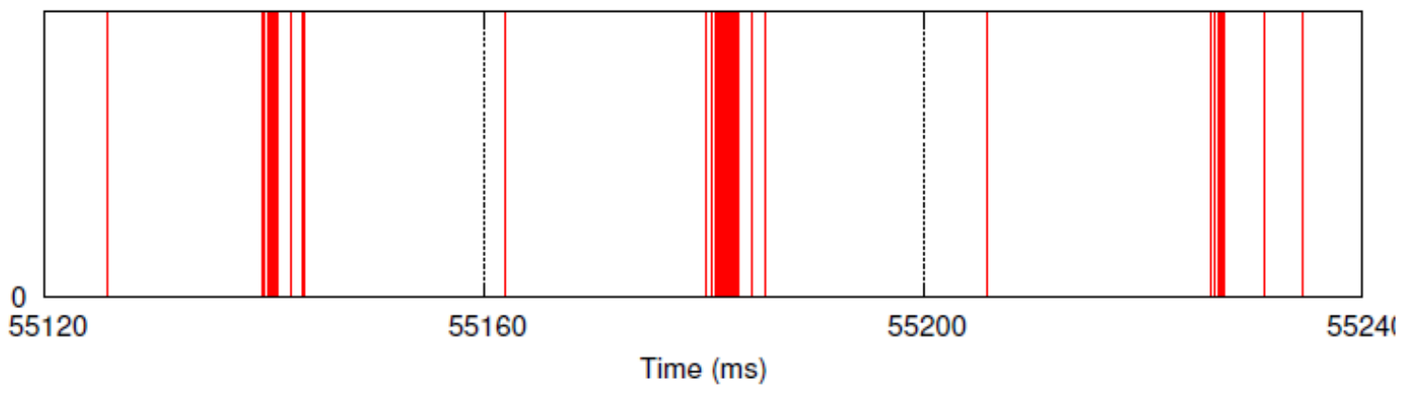
Why a custom tracer

The ptrace() ping-pong

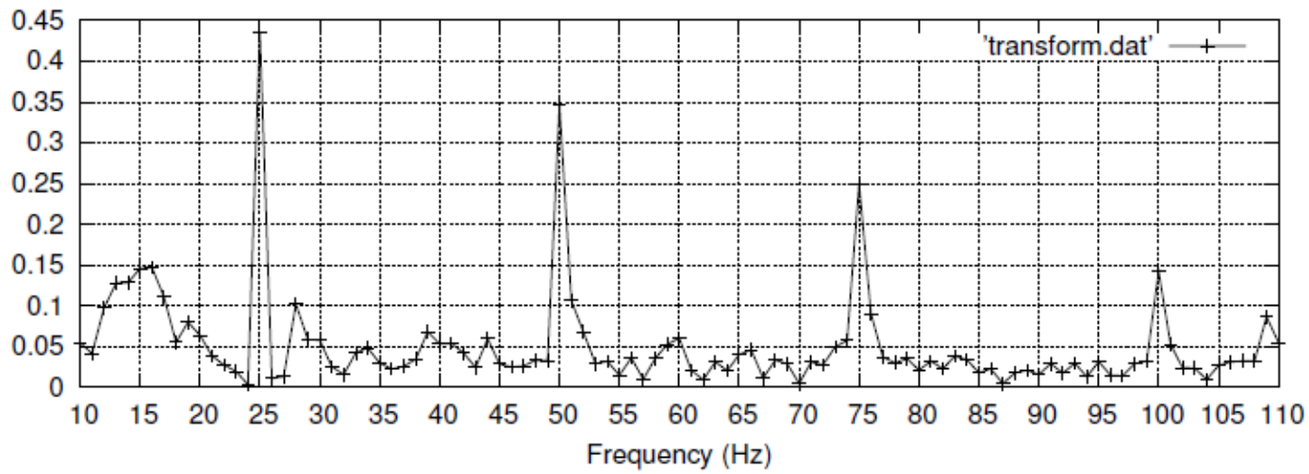


Period detection

The tracer produces a sequence of time-stamps



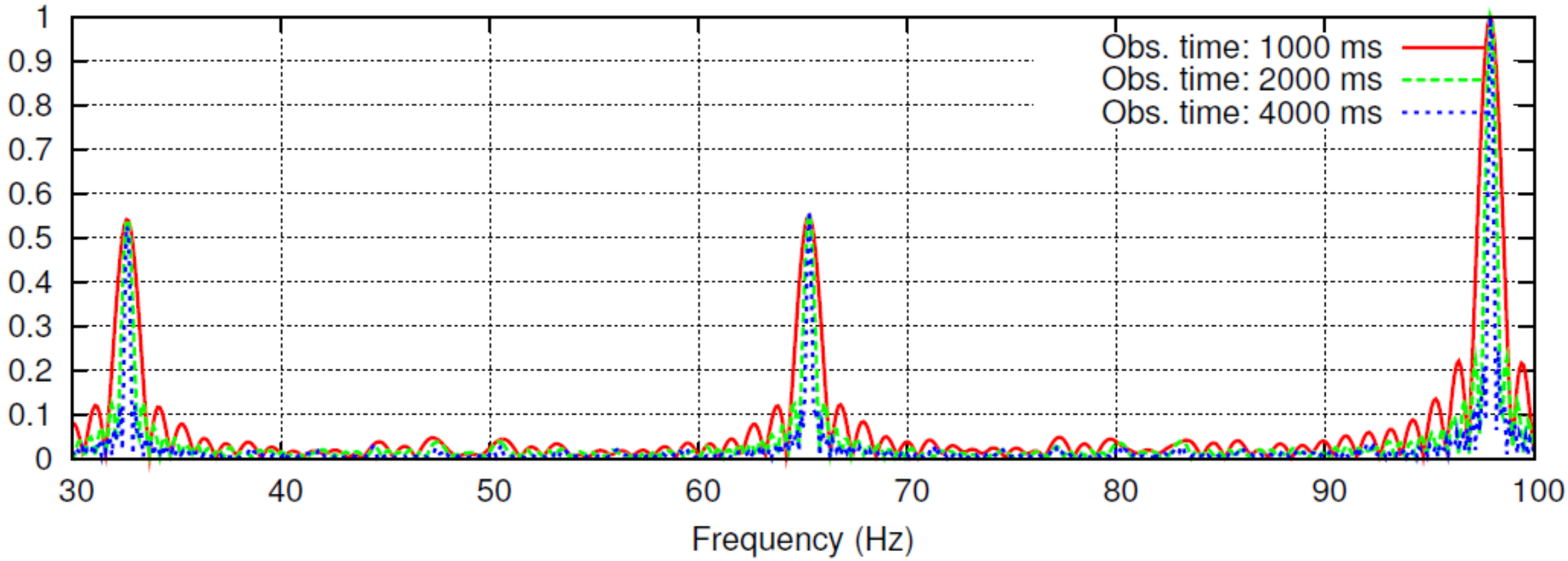
Time-stamps used to compute a Fourier-transform



A heuristic catches the first harmonic

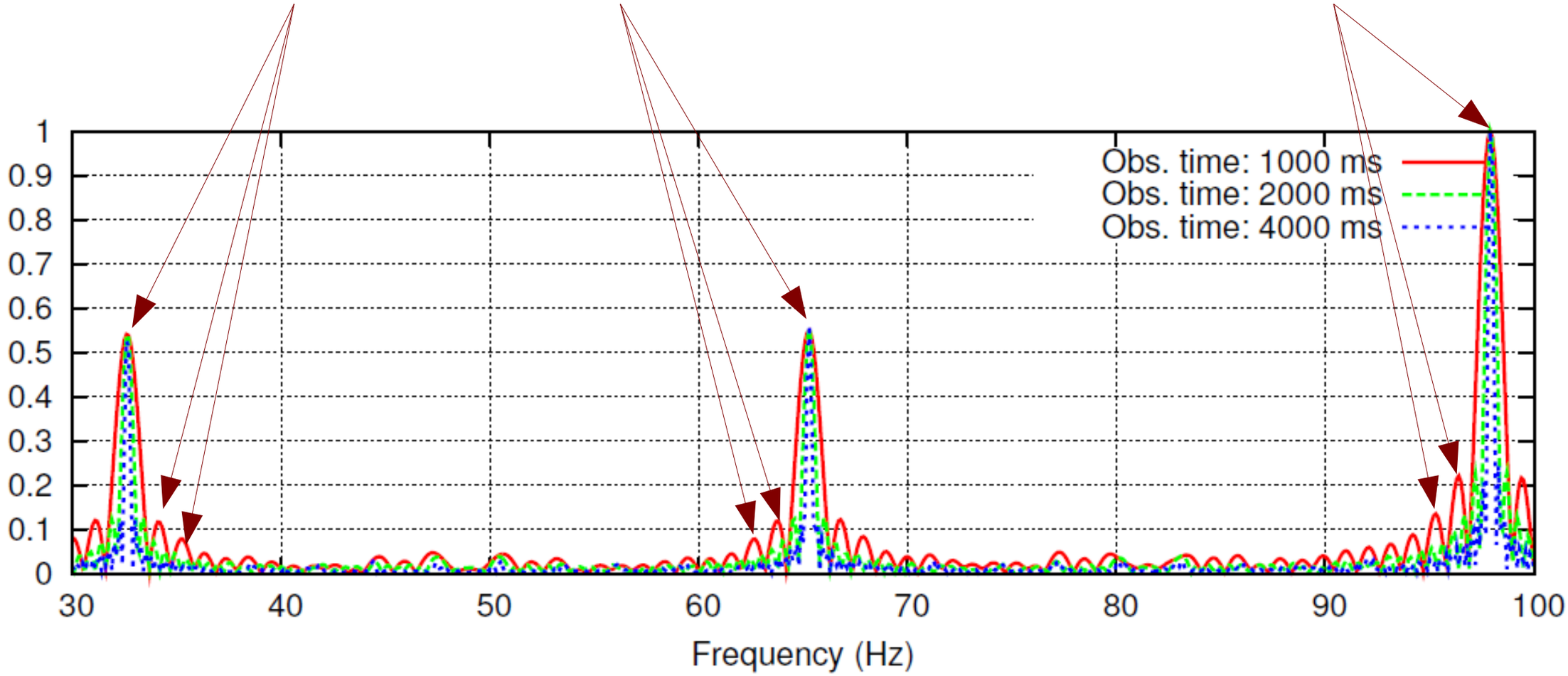
Heuristic

Start from the Fourier Transform sampled at δf



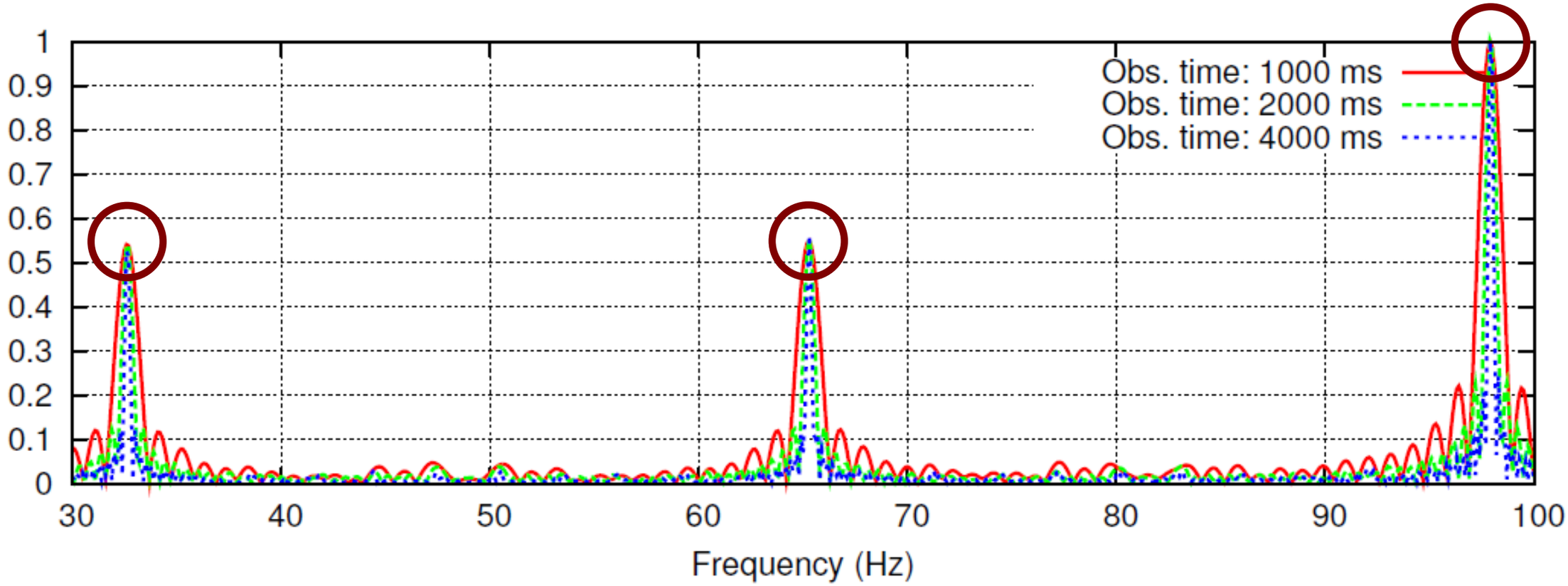
Heuristic

Identify local maxima of the Fourier Transform



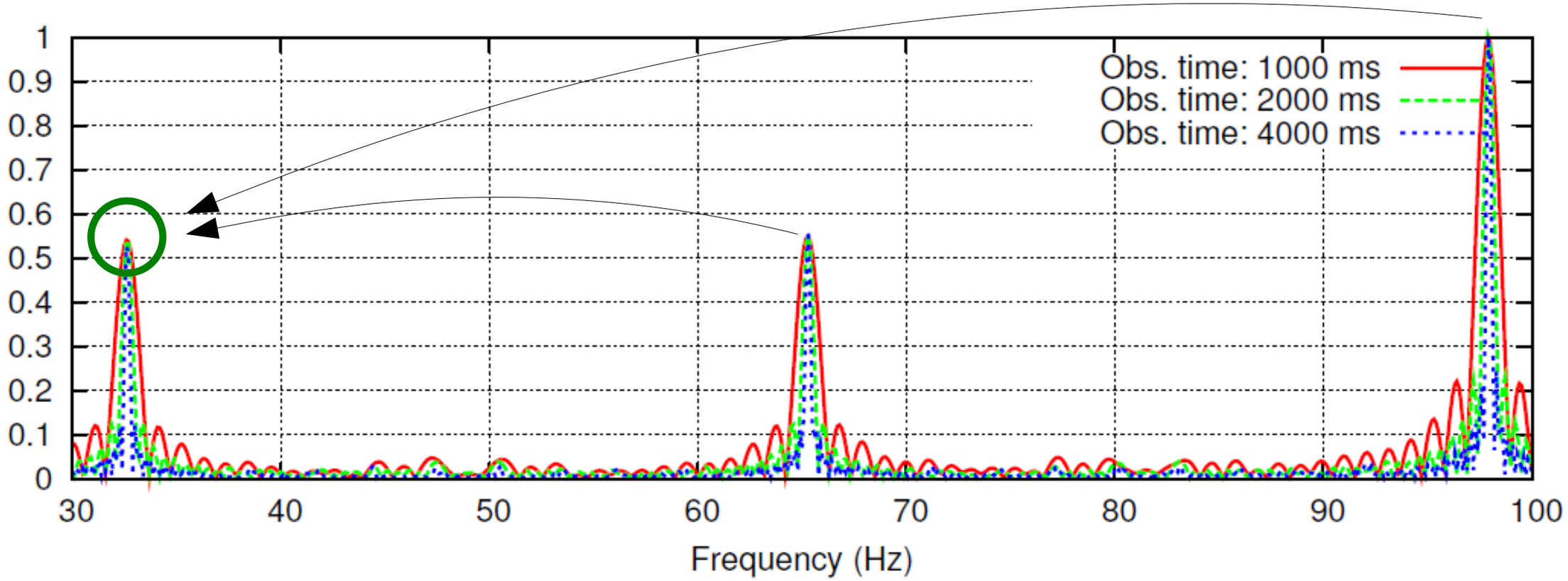
Heuristic

Select peaks higher than α times average value



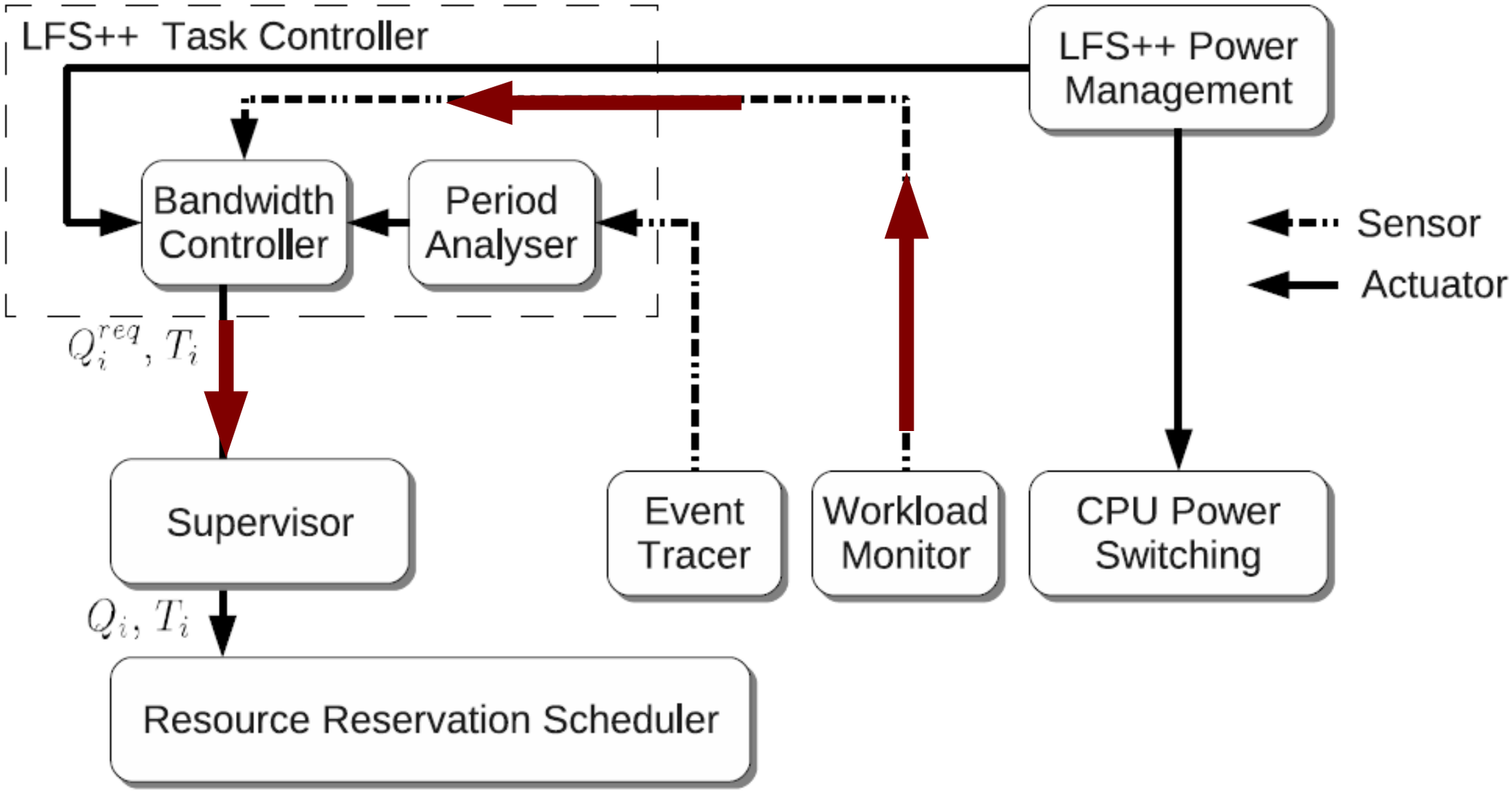
Heuristic

Accumulate FT of up to k^{max} multiples of each peak



Budget identification

“Feedback-based scheduling” budget control loop



Experimental results

Experimental results

Set-up

- Linux 2.6.29, with an implementation of the CBS scheduler
- Feedback-scheduling by means of AQuoSA
- **mplayer**
  - modified to monitor the **Inter-Frame Time (IFT)** and
  - Application tracing by using **qtrace** (kernel-level)

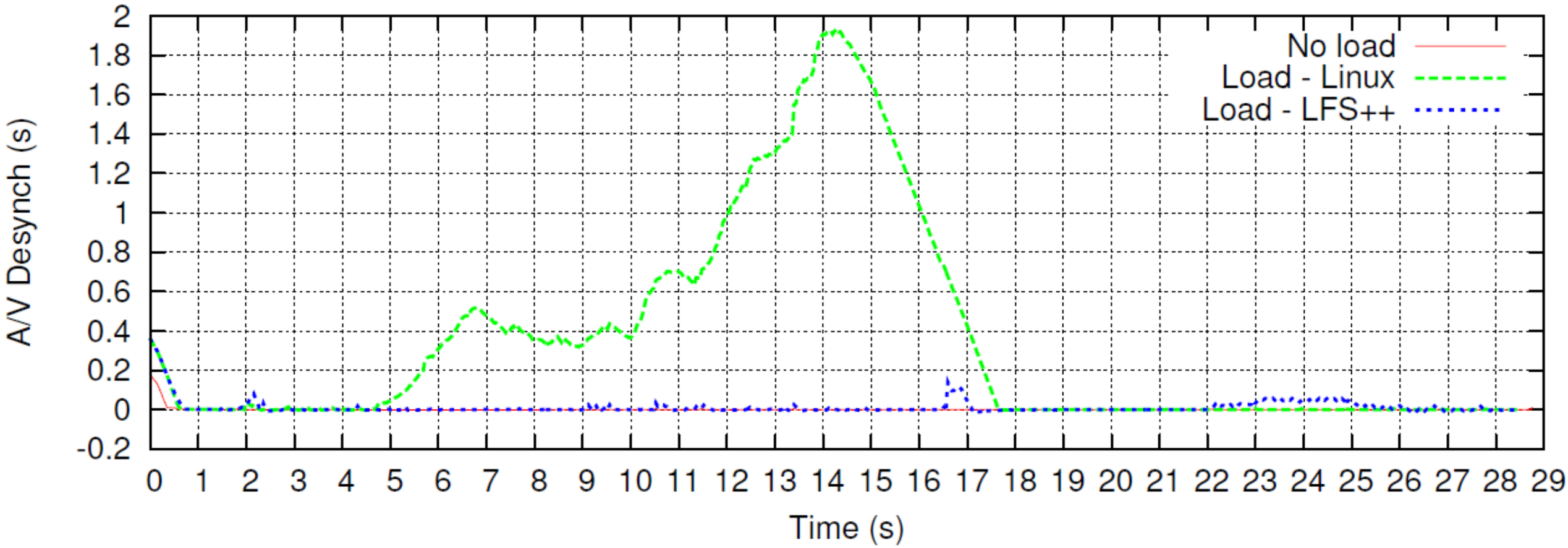
Validation metrics

- **Inter-Frame Time** (for mplayer)
- **A/V desynchronisation** (for mplayer)
- **Response-time** (for synthetic application)
- **Allocated bandwidth** on the Real-Time scheduler

Benefits for the application
(LFS++ improves over Linux)

A/V desynchronisation in mplayer while starting the Eclipse IDE

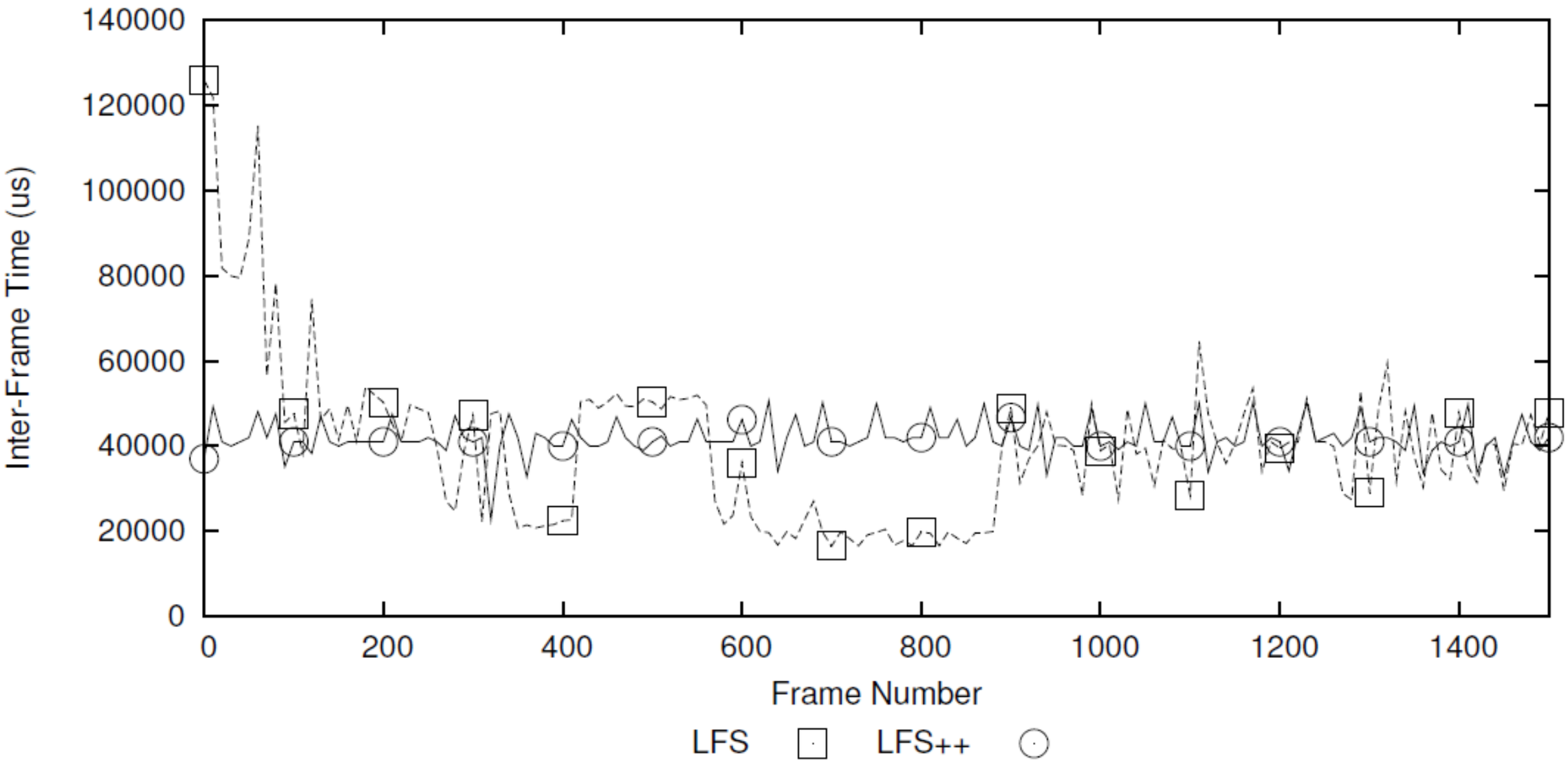
- **90% reduction** of the peak A/V desynchronisation



LFS++ improves over LFS

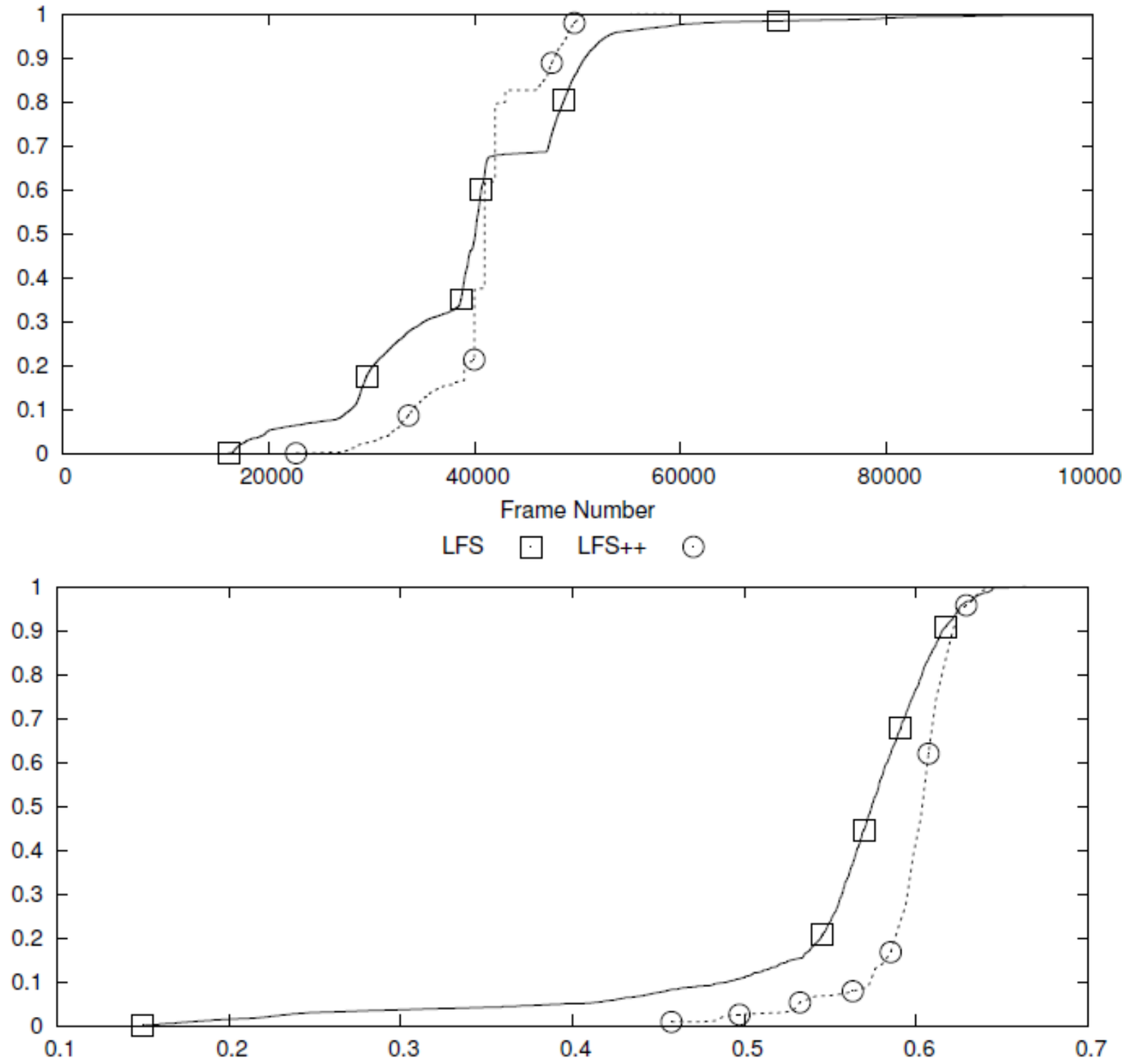
(our prior work)

Inter-frame times of mplayer are much more stable



LFS++ improves over LFS
(our prior work)

Inter-frame times CDF is tighter with LFS++



Allocated bandwidth CDF is tighter with LFS++

Multiple LFS++ instances

Frequency misdetection degree with other LFS++ controllers already active

- New application runs without guarantees while traced

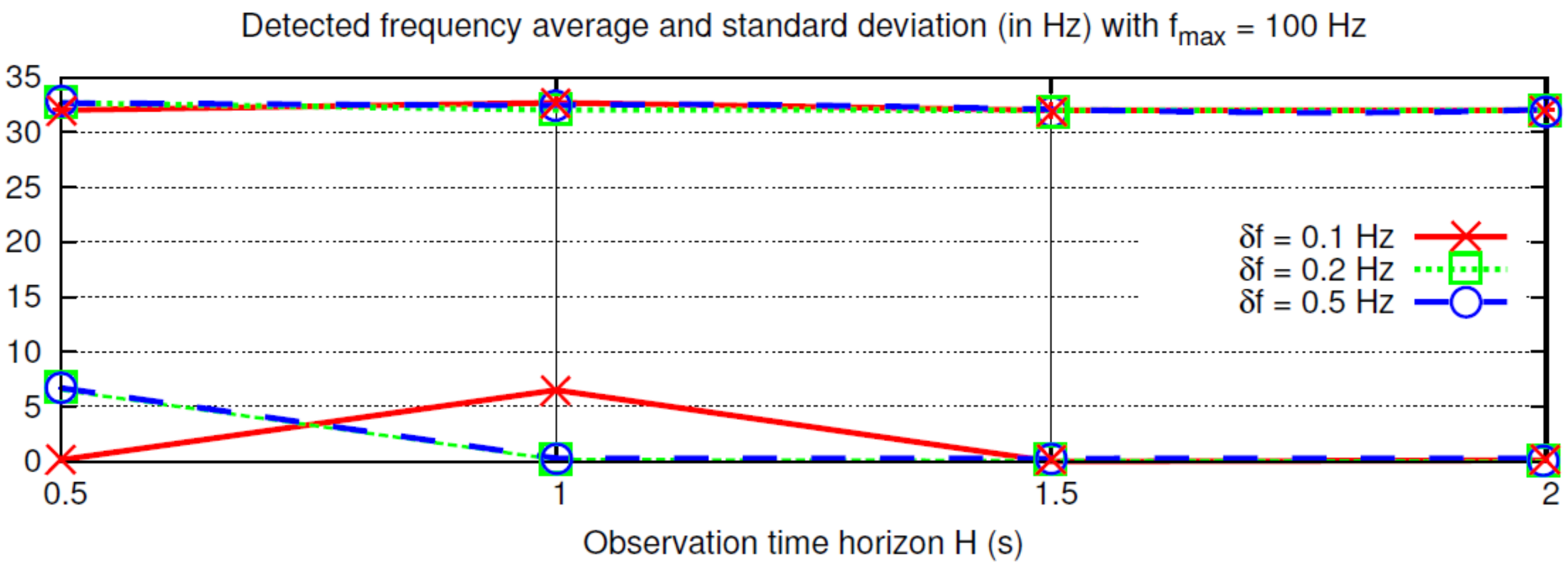
Overall load	New reservation	Average freq (Hz)	Std Dev (Hz)	Max (Hz)
0%	-	32.69	6.60	98
15%	(645,4300)	41.67	22.97	97
30%	(1200,8000)	57.98	30.79	95
45%	(1650,11000)	75.03	26.35	92
60%	(2250,15000)	68.47	25.51	93

Table 2. Precision of the period detector with respect to the real-time load in the system. Reservation budgets and periods are in μs, average, standard deviation and maximum values of the detected frequency are in Hz.

Experimental results

Precision of the frequency detection heuristic

- acceptable even with 0.5s of observation-time
- detected frequency rock-stable from 1.5s observation-time



Cost of tracing

Tracing overhead

- When using strace: +5,51%
- When using qostrace: +2,69% (previous paper)
- When usinig qtrace: +0.63% (kernel-level tracer)

88.57% overhead reduction with the custom tracer

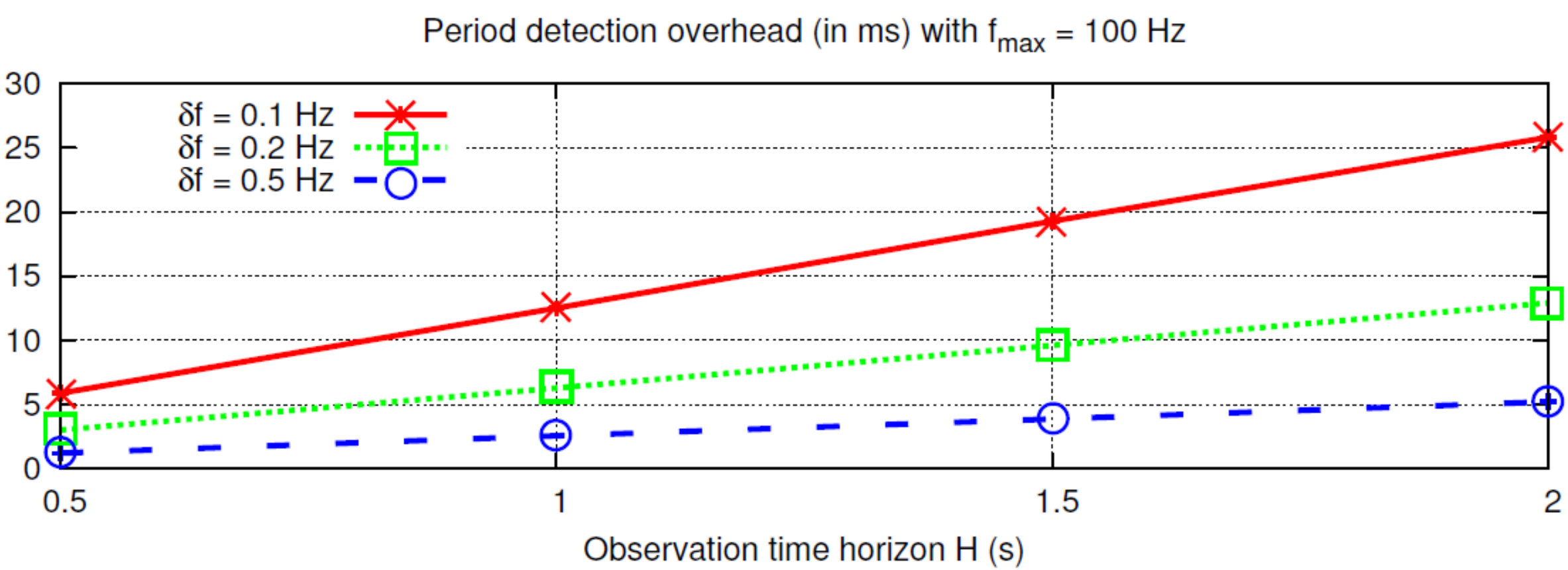
Tracer	Average (sec)	Relative average	Standard deviation (sec)
NOTRACE	21.0916	-	0.094951
QTRACE	21.2253	0.63%	0.143581
QOSTRACE	21.658	2.69%	0.221327
STRACE	22.2536	5.51%	0.140593

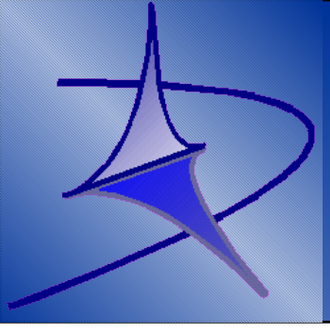
Table 1. Overhead introduced by various tracers, compared to when no tracer is used (first row).

Cost of Fourier Transform

Fourier Transform Overhead

- 1ms to 25ms, depending on parameters (and accuracy)
- Linear in the observation-time and δf parameter (as expected)





Conclusions

Future work

Better engineering of the tracer

- **Tracing of wake-up times** instead of system calls
  - **done** in submitted journal extension – improved heuristic precision at reduced observation times
- Security issues: tracing data queried via ioctl() on special device

Integration with power-management

- Supporting **dynamic CPU frequency switching**
  - **done** in submitted journal extension – need cooperation between LFS++ schedulers and OS daemon for power management

Future work

Guaranteeing multi-threaded applications

- e.g., VideoLAN Coder (vlc)
- Only marginally tested
- Main problem with inter-dependencies
  - Plan to use bandwidth inheritance techniques

Multi-core systems

- All experiments on a single-CPU machine

Feedback-based budget control

- Closer cooperation with the scheduler for **quicker response** to **workload increases**

Thanks for your attention

Questions ?

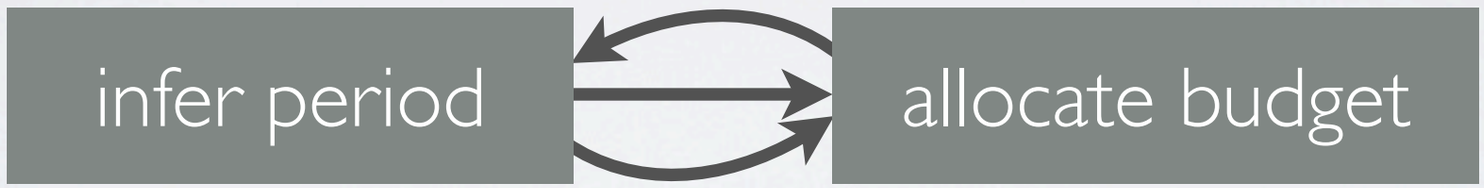
http://retis.sssup.it/people/tommaso

PLUS

- I would always prefer automatic inference of real-time parameters of manual provisioning à la Redline
- nice explanation of the impact of ill-selected server period
- generalizes to proportional share / round-robin-style schedulers

MINUS

- the proposed tracing solution is not new (see dtrace)
- detection of real-time parameters:



- unclear: how does the predictor bootstrap?
- are we constantly over-allocating?
- variable frame rate video, game engines with irregular periods