Reservation-Based Scheduling for IRQ Threads

Luca Abeni, Nicola Manica, Luigi Palopoli

luca.abeni@unitn.it, nicola.manica@gmail.com, palopoli@dit.unitn.it

University of Trento, Trento - Italy

Overview of the Talk

- Introduction: problem definition
- Effects of interrupt handling in vanilla Linux
- Effects of interrupt handling on Preempt-RT
  - Some problems are solved...
  - ...But some problems are still there!
- We've got to look beyond fixed priorities...
  - Reservation-based scheduling
  - How do CPU reservations apply to IRQ threads?
  - Do they allow to control the impact of interrupt handlers
  - Do they allow to control the hw devices throughput?

Introduction

- Real-Time theory traditionally addressed problems related to CPU allocation. . .
- . . .But some real-time applications also need other resources to execute
- Example: some time-sensitive applications need to access some hardware device respecting some temporal constraints
  - Correct CPU scheduling is useless if the hardware device is not properly served
  - Giving CPU time to an application is not enough if device drivers cannot execute
- Sometimes, device drivers can steal CPU time to applications

Interrupt Handling

- Traditional kernels: ISRs and *Bottom Halves*
- Have always priority over real-time applications
  - Can preempt real-time tasks
  - Can steal time to real-time tasks
- RT kernels: interrupts served in dedicated threads
  - Linux $\rightarrow$ Preempt-RT patch: transforms ISRs and bottom halves in threads
  - Interrupt threads can have lower priorities than real-time tasks
  - If real-time tasks do not need to interact with hardware devices (they do not depend on the interrupt threads), the problem is solved!
  - Problem: how to schedule the IRQ threads?

Example - What to test

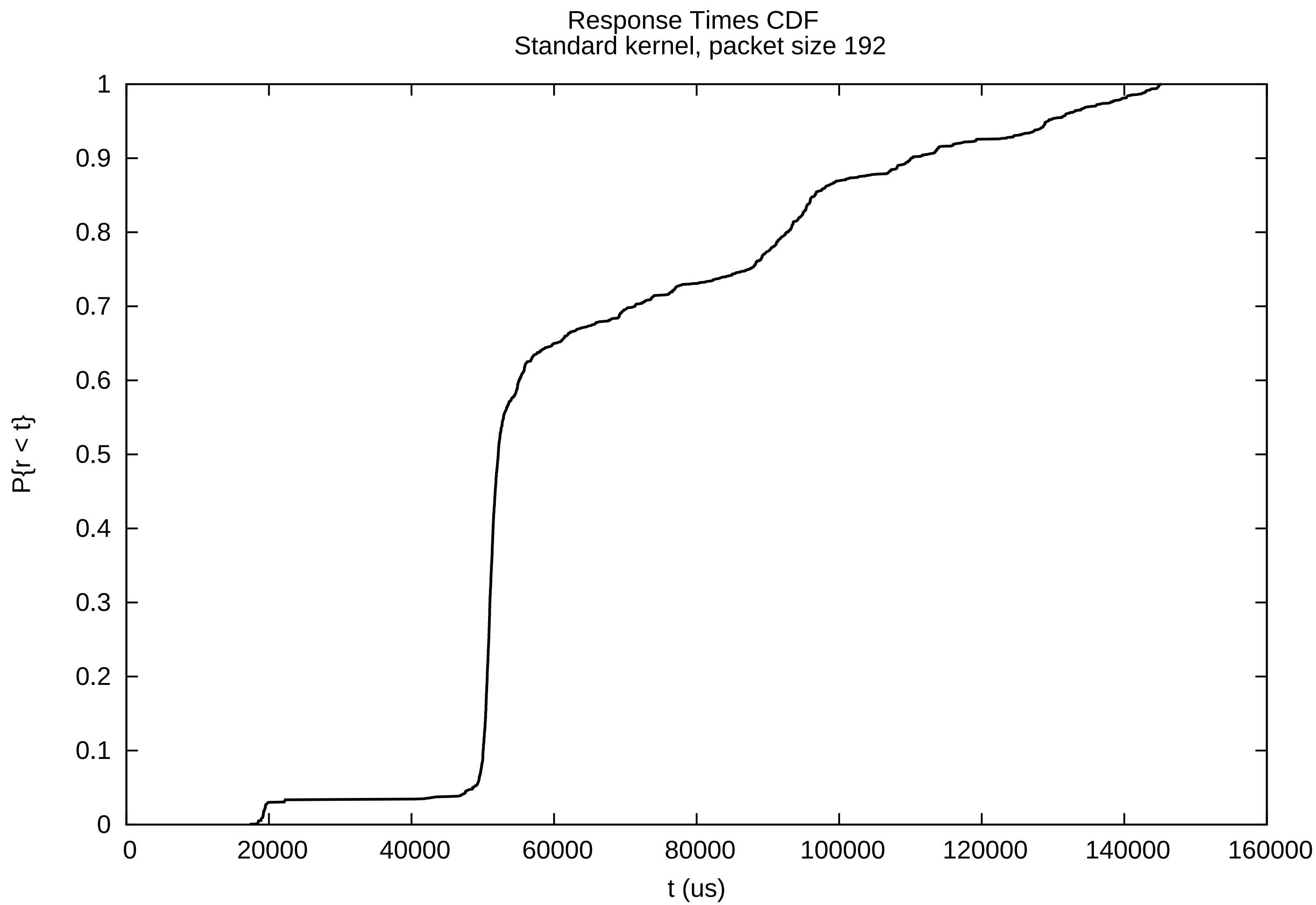
- Effects of device handling on real-time tasks
  - Real-time performance: response time (affected by the *kernel latency*)
    - Highest priority task: worst case response time = WCET + latency
  - Hardware device: network card
    - high throughput device
    - controlling the workload is easy
- Someone already mentioned problems with high network load and small packets...
  - Interesting things happen when the system is overloaded

Example - Experimental Setup

- Periodic real-time task, scheduled with high priority
  - A task with period $50ms$ and execution time around $20ms$ is used
  - The task is scheduled with the highest real-time priority $\rightarrow$ expected response time: around $20ms$
- A non real-time task receiving a lot of traffic from the network can increase the response time of the real-time task!!!
  - The `netperf` program is used
- The netperf server is run as non real-time $\rightarrow$ it should not affect the real-time performance

Example - Results

- When using 192-bytes long UDP packets, the response time of the periodic task goes to more than $100ms$!!!

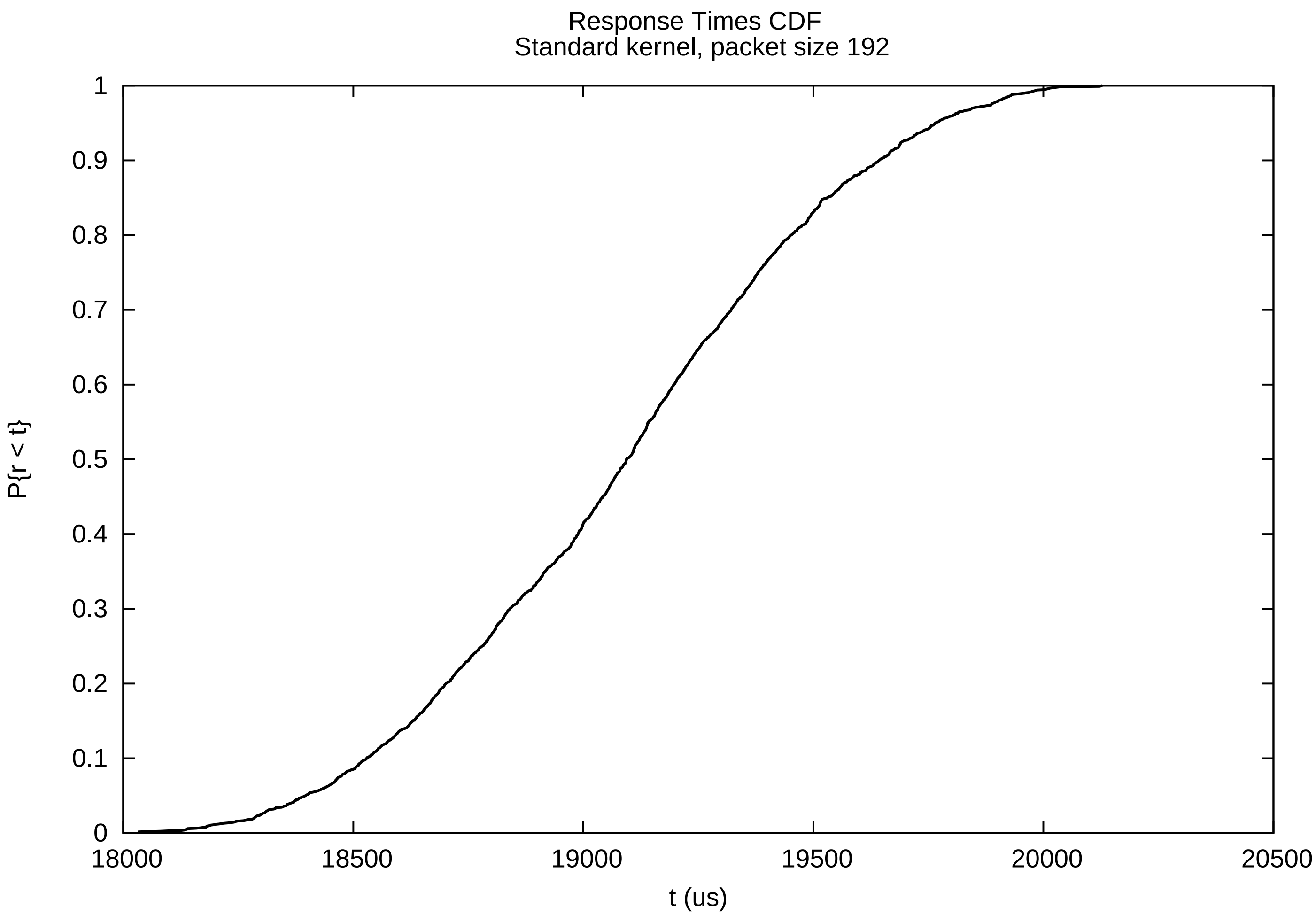


Solution: Preempt-RT

- The Preempt-RT patch transforms Linux in a real-time kernel. It addresses the mentioned problem by transforming ISRs and bottom halves in threads
  - If an IRQ thread is scheduled with a lower priority than a real-time task, then the real-time task's response time is not affected
- Fixes the problem, but...
  - Fixed priority scheduling is not flexible enough!
- Let's see!

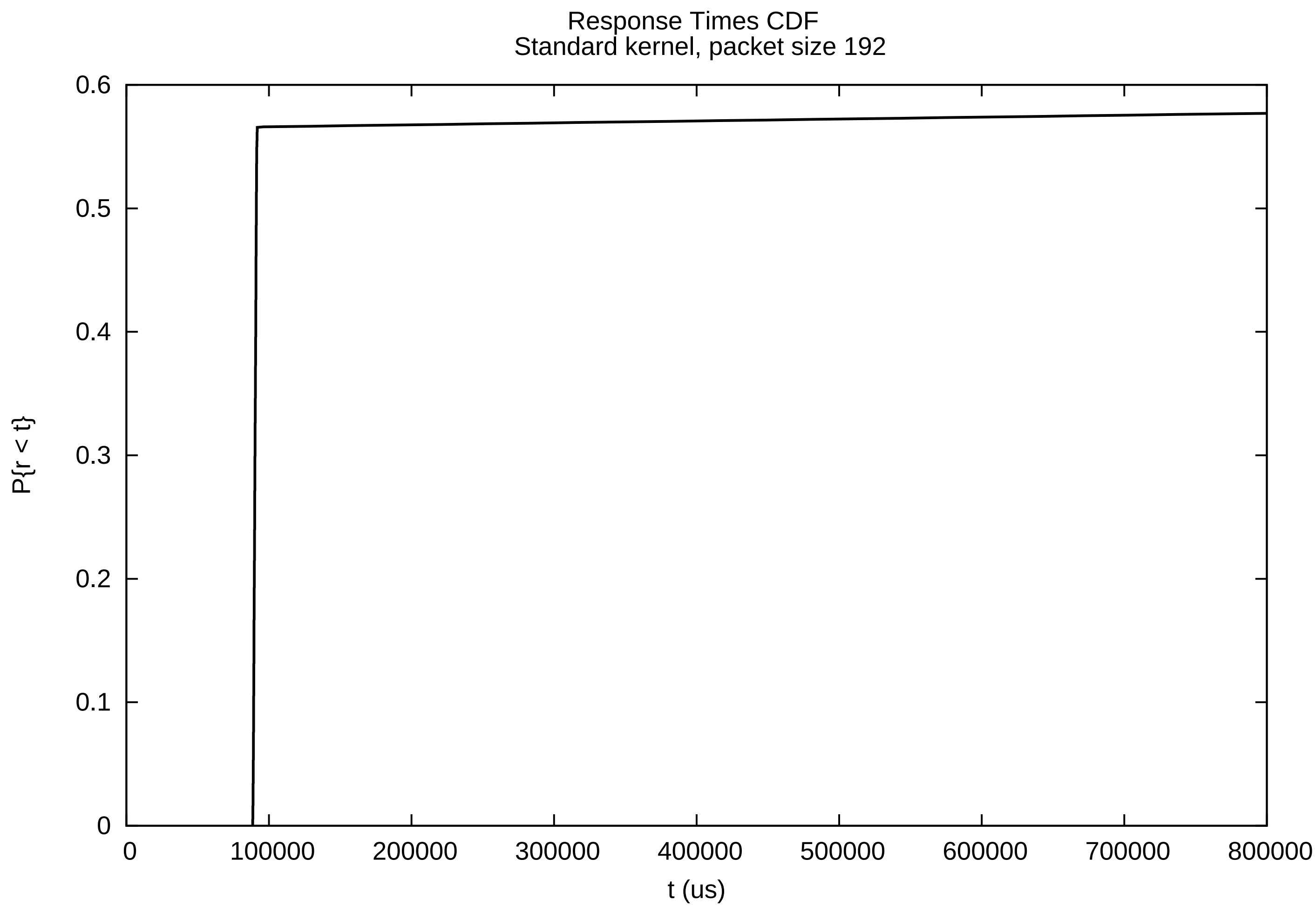
Priority to the Real-Time Task

- Low response times, low throughput ($48Mbps$)



Priority to the IRQ Thread

- High throughput ($74Mbps$), high response times

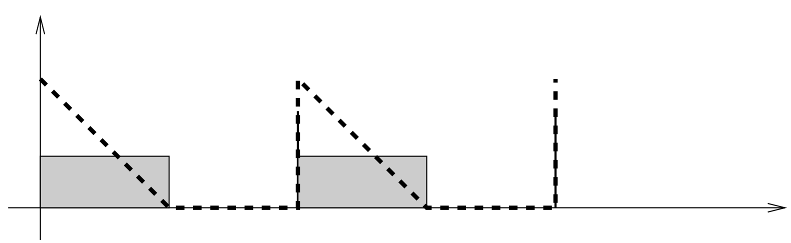


Throughput/Latency Trade-Offs

- Problem: fixed priority scheduling is not flexible enough
  - It only allows to say things like “the real-time task is more important than the device driver” or “the device driver is more important than the real-time task”
  - How to schedule the IRQ handlers?
- We might want to say things like “give $x\%$ of the CPU time to the device driver”, or similar
- Resource Reservations!

Resource Reservations

- Resource Reservations $\rightarrow$ temporal protection
  - Every task is allowed to use a resource for an amount of time Q^s every period T^s
  - Accounting and Enforcement
- CPU scheduling $\rightarrow$ CPU Reservations (implemented in Resource Kernels)
  - Traditional implementations $\rightarrow$ aperiodic servers
  - Deferrable Server...



- Here, the Constant Bandwidth Server (CBS) is used

The Constant Bandwidth Server

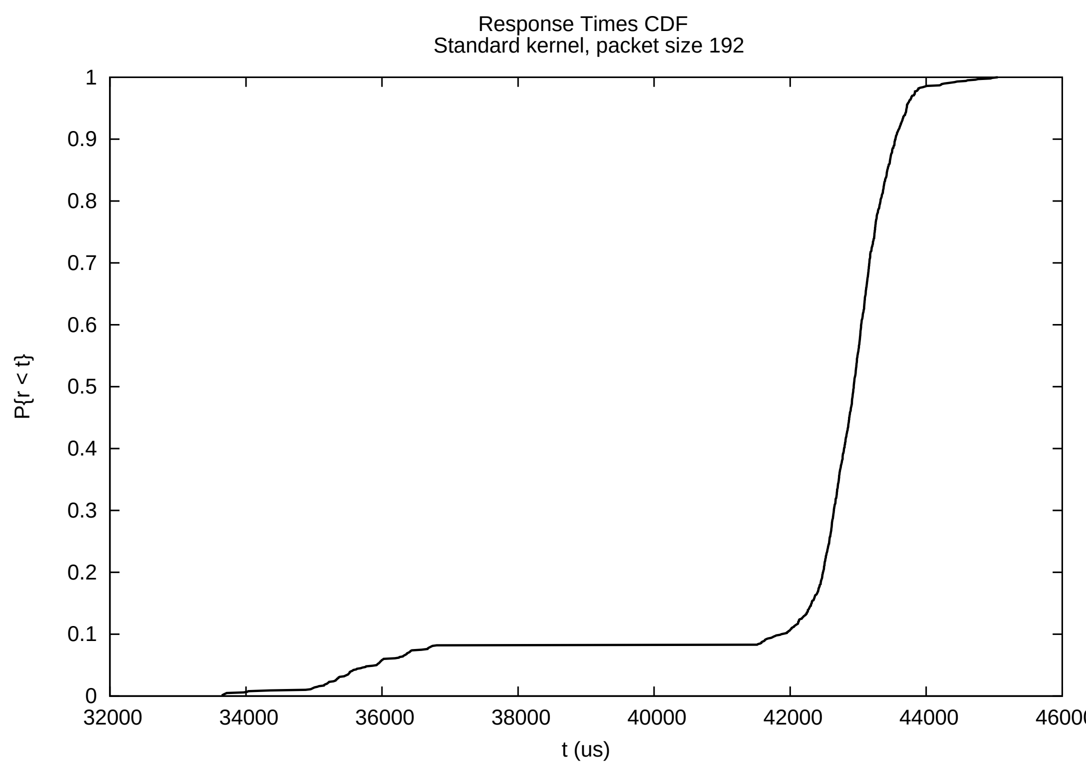
- The CBS is used, but every reservation-based scheduler can be used
  - Reservations based on RM, EDF, whatever...
- Basic Ideas:
  - *budget* $\rightarrow$ decreases when the served task executes
  - *server deadline* $\rightarrow$ assigned to served task
  - job arrival (wakeup) $\rightarrow$ check if the last server deadline can be used
  - budget exhausted $\rightarrow$ deadline postponed
- Server parameters:
  - Q_i: maximum server budget
  - T_i^s: server period (soft relative deadline)

Reservation-Based Scheduling

- Two scheduling parameters (Q^s, T^s)
- Q^s/T^s is the fraction of CPU time reserved to a task
- T^s is the "granularity" of the allocation
- Serving an IRQ thread with a (Q^s, T^s) reservation:
  - Reducing Q^s/T^s, the impact of interrupt handling on real-time tasks can be reduced...
  - T^s allows to control the "device's responsiveness"
  - We have some theoretical analysis

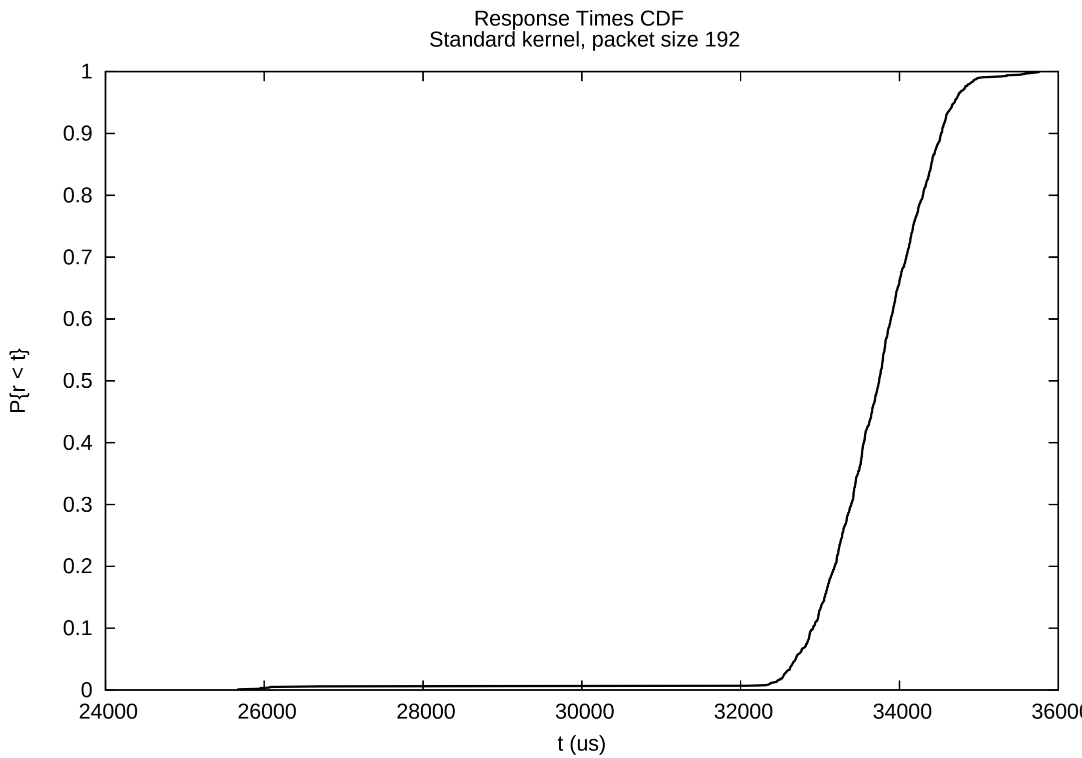
Reservations and IRQ threads

- Example: $RSV_1 = (4, 10)$ for the periodic task, $RSV_2 = (4, 10)$ for the hard IRQ, $RSV_3 = (1.5, 10)$ for the netperf server
  - Throughput: $74 Mbps$
  - Worst-Case Response Time: $46 ms$



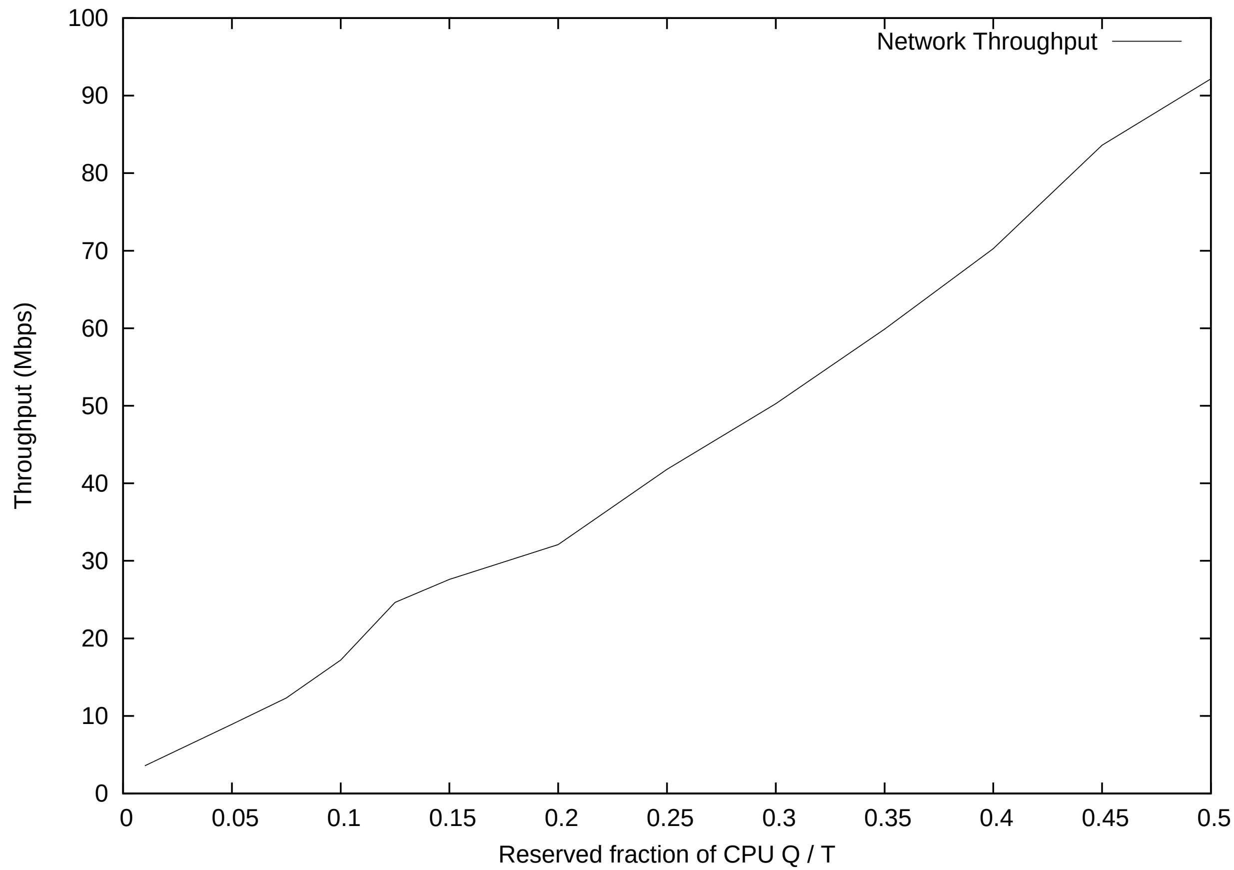
Latency / Throughput Trade-Offs

- Example: The response time can be reduced by using $RSV_1 = (5, 10)$, $RSV_2 = (2, 10)$, $RSV_3 = (1, 10)$
  - Throughput: $65Mbps$; Worst-Case Response Time: $36ms$



Controlling the Throughput

- Example: The CBS parameters (Q^s, T^s) can be used to control the network throughput
- Non-overloaded system (larger UDP packets):



Controlling the Network Latency - 1

- Up to now we considered:
  - Latency / Response Time for the real-time task
  - Network throughput
- What about network latency?
  - The server period T^s can be used to control the response time for network packets
  - Tested by looking at the `ping` RTT
  - RTT as a function of the CBS parameters

Controlling the Network Latency - 2

Q^s	T^s	min RTT	avg RTT	max RTT	mdev RTT
$1ms$	$3ms$	0.062	0.109	16.498	0.289
$2ms$	$6ms$	0.057	0.105	36.504	0.368
$3ms$	$9ms$	0.058	0.103	38.684	0.379
$4ms$	$12ms$	0.058	0.101	50.991	0.428
$5ms$	$15ms$	0.059	0.102	50.928	0.453
$6ms$	$18ms$	0.058	0.103	52.814	0.507
$7ms$	$21ms$	0.059	0.104	79.782	0.566

- Average and minimum RTT values do not depend on T^s...
- But worst case values do!!!

Conclusions

- Device drivers (interrupt handlers) can affect the schedulability of real-time tasks
  - Real-time systems allow to schedule interrupt handlers
- Problem: how to schedule the IRQ threads?
  - Fixed priorities are not flexible enough
  - Low latencies $\rightarrow$ low device throughput
  - High device throughput $\rightarrow$ high latencies
- Reservation-based scheduling allows to find trade-offs between latencies and throughput!!!
  - Also allows to control the device throughput / response times