Control of Computer Systems

Karl-Erik Årzen & Anton Cervin

Outline

1. Overview
2. Nice Control of Linux
3. Feedback Scheduling of Control Tasks
   - Infinite horizon
   - Finite horizon
   - MPC
4. Control of Web servers

Control of Computer Systems

Apply control as a techniques to manage uncertainty and achieve performance and robustness in computer and communication systems.

One of the strongest increasing areas in real-time computing (adaptive/flexible scheduling) and networking.

Applications in

- Internet protocols, e.g., TCP and extensions
- Internet servers (HTTP, Email)
- Cellular phone systems (power control, ...)
- CPU scheduling

Control used to manage finite resources (Resource allocation as a control problem = feedback scheduling)

Control of Computer Systems

New area

- however, feedback has been applied in ad hoc ways for long without always understanding that it is control

Textbook has recently been published

- “Feedback Control of Computer Systems”, Hellerstein, Diao, Parekh, Tilbury

Control of Computer Systems

Control of computing systems can benefit from a lot of the classical control results

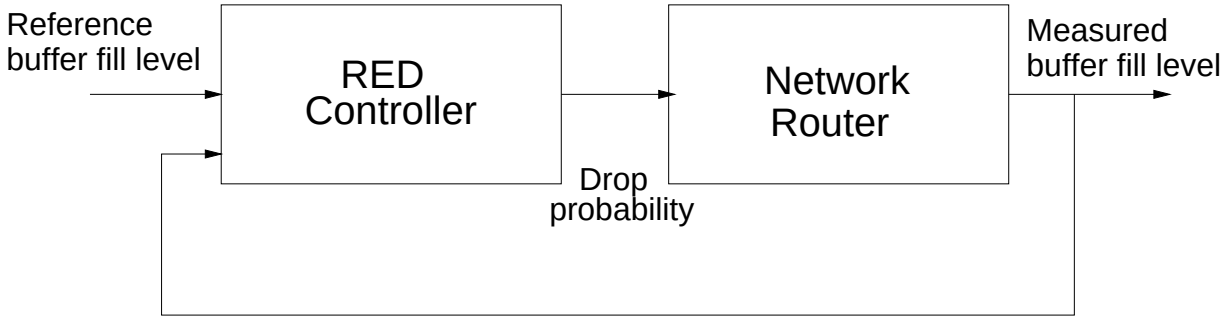
- However, several new challenges
- First principle modeling not so natural
- Complex dynamics no longer the problem

Example: Internet Protocol

The congestion control in TCP is one of the major reasons why Internet has been able to expand at the current high rate and still work properly.

- Congestion window (cw) decides how many un-ack'ed packets a host can have
- When cw below threshold it grows exponentially
- When cw above threshold it grows linearly
- Whenever there is a timeout the threshold is set to half the cw and cw is set to 1.
- Nonlinear behavior

Example: Internet Protocols

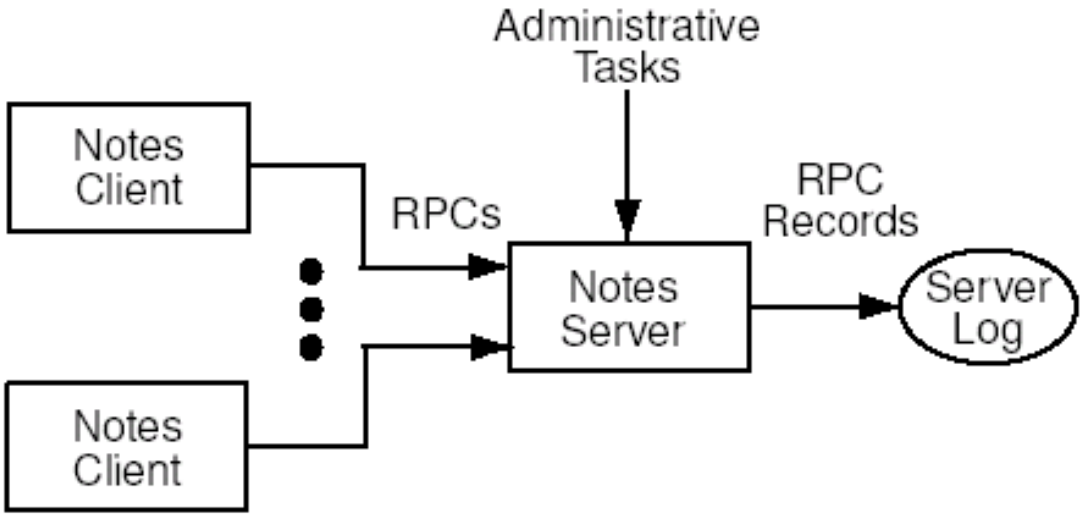


Random Early Detection (RED) of Router Overloads

- Prevent router buffers from overflowing
- Random drops of packets before the buffer is full

A lot of ongoing work on improvements of IP based on models and theory rather than on ad hoc fixes

Example: Lotus Notes E-Mail Server

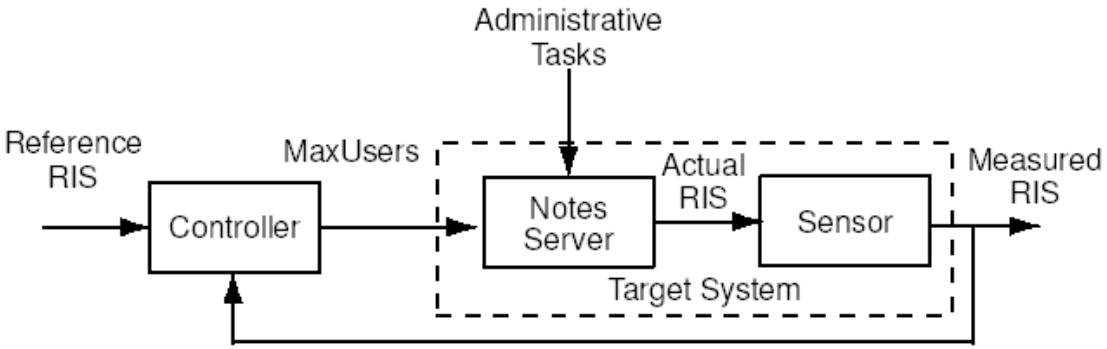


Client-server application

Interaction using Remote Procedure Calls (RPC)

Server log of RPC statistics

Example: Lotus Notes E-Mail Server



Control the number of RPCs in the server (RIS) by dynamically adjusting the maximum allowed users (MaxUsers)

First order model derived from data:

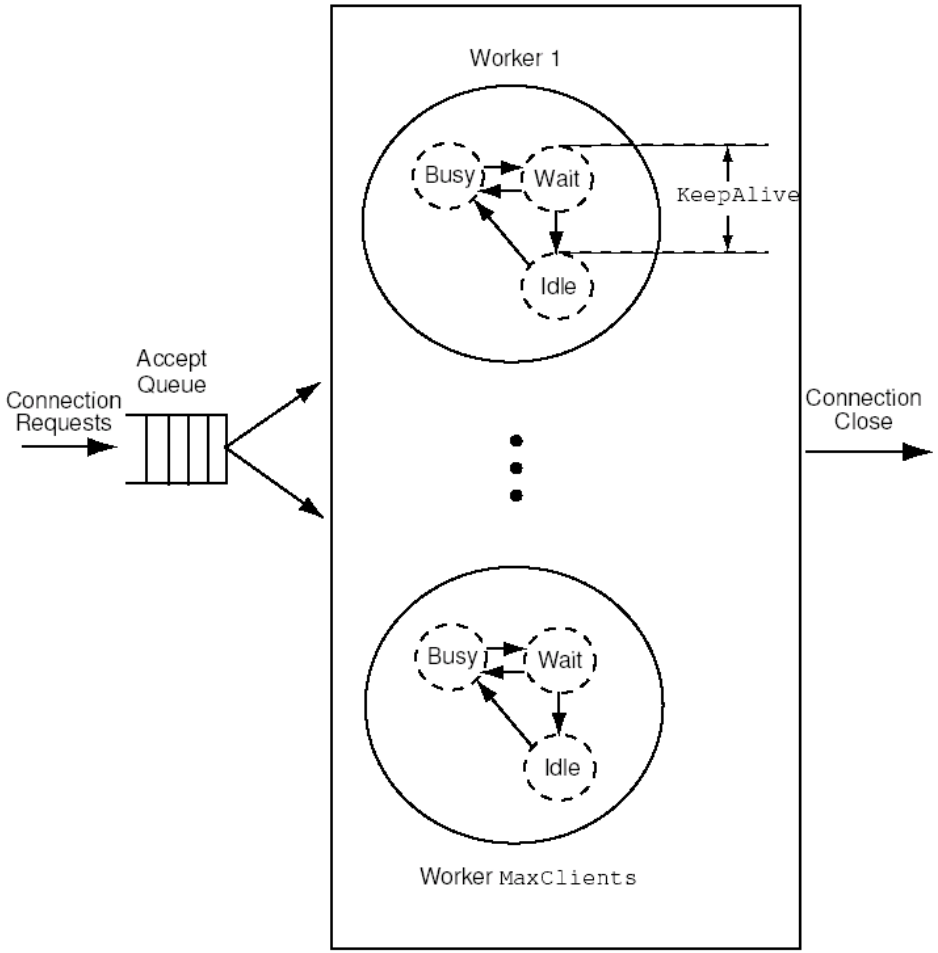
$y(k+1) = 0.43y(k) + 0.47u(k)$ where $y(k) = RIS(k) - RIS_0$ and $u(k) = MaxUsers(k) - MaxUsers_0$

First order LP-filter added to remove outliers

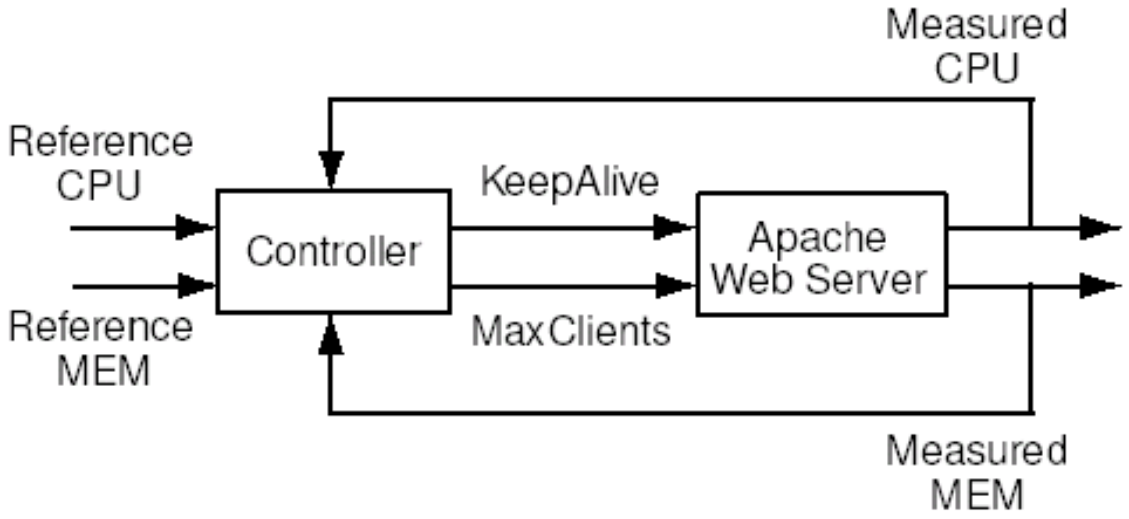
Resulting, second order system, controlled by PI-controller designed using pole-placement

Example: Apache HTTP Server

- HTTP requests from clients to server
- Pool of workers being either Idle, Busy or Waiting (Persistent Connections)
- MaxClients limits the size of the worker pool
- KeepAlive determines how long a worker is waiting before it becomes idle



Example: Apache HTTP Server



Control of CPU utilization and memory utilization

Too large MaxClients → large consumption of CPU and memory

Too large KeepAlive → underutilization

Too small KeepAlive increases CPU consumption since connections must be re-established for requests from the same user

Example: Apache HTTP Server

Two first-order transfer functions derived from input-output data

$$CPU(z) = G_{MC}(z)MC(z) + G_{KA}KA(z)$$

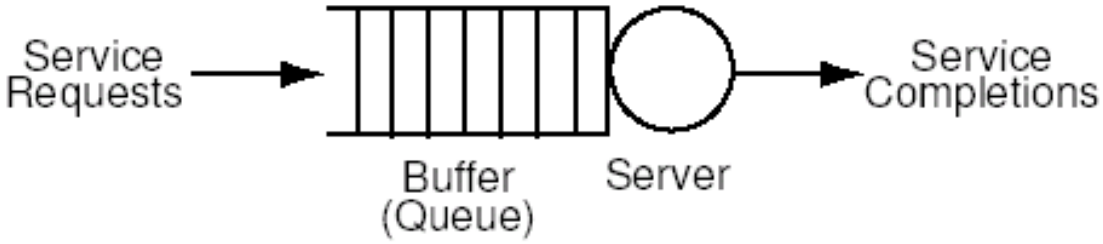
around a certain operating point

PI-controller using KeepAlive control signal

Design using pole placement

In principle, the same type of design that you have been doing in the projects

Example: Queuing Systems



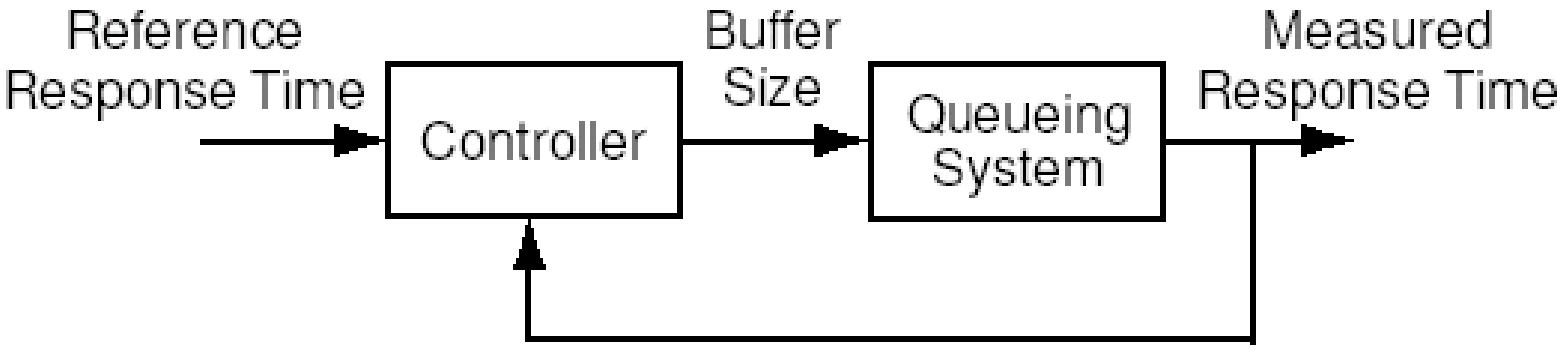
Work requests (customers) arrive and are buffered

Service level objectives (response time for request belonging to class X should be less than Y time units)

Reduce the delay caused by other requests, i.e., adjust the buffer size and redirect or block other requests

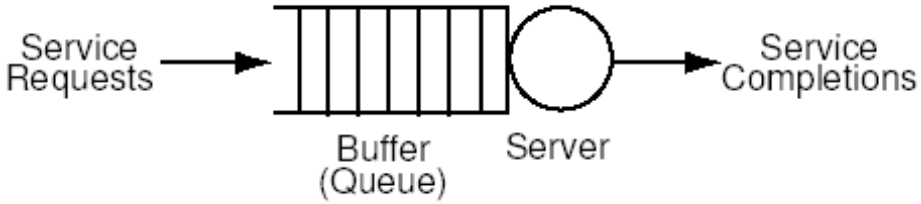
Admission control

Example: Queuing Systems



Example: Queue Length Control

Assume an M/M/1 - queuing system:

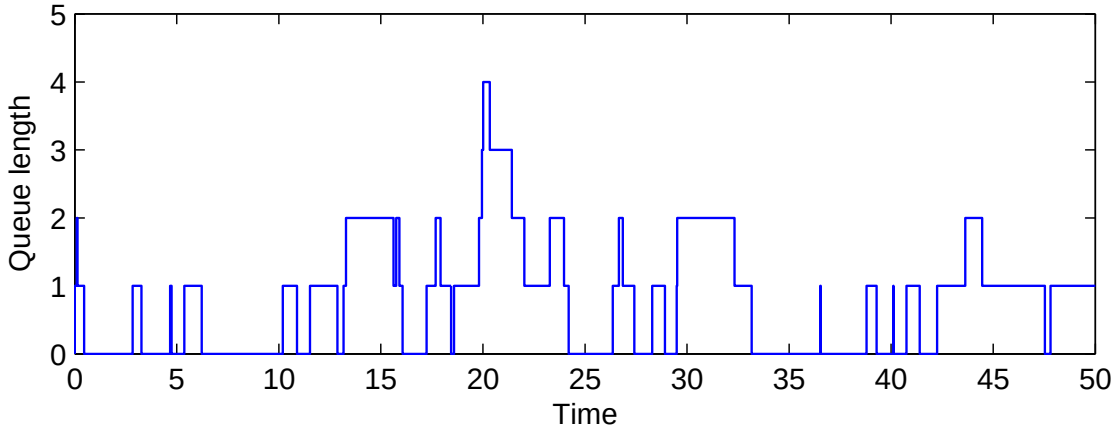


- Random arrivals (requests), average λ per second
- Random service times, average $1/\mu$ and exponentially distributed
- queue containing x requests

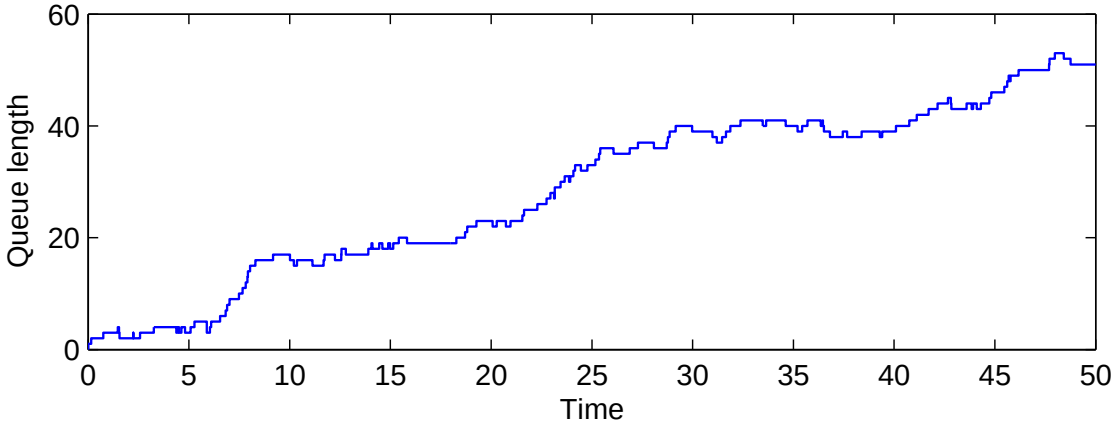
Intuition: $x \to \infty$ if $\lambda > \mu$

Queue Length Control: Simulation

$\lambda = 0.5, \mu = 1$



$\lambda = 2.0, \mu = 1$



Queue Length Control: Model

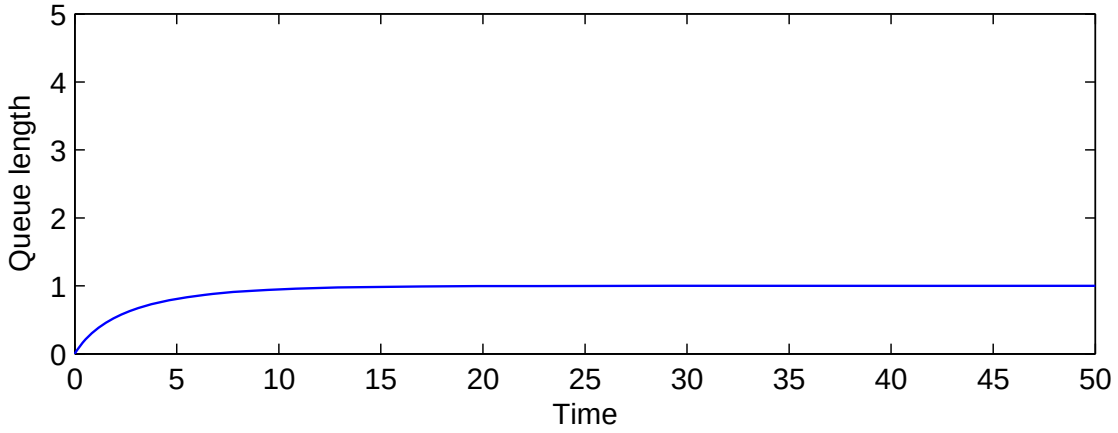
Approximate the system with a nonlinear flow model (Tipper's model from queuing theory)

The expectation of the queue length x is

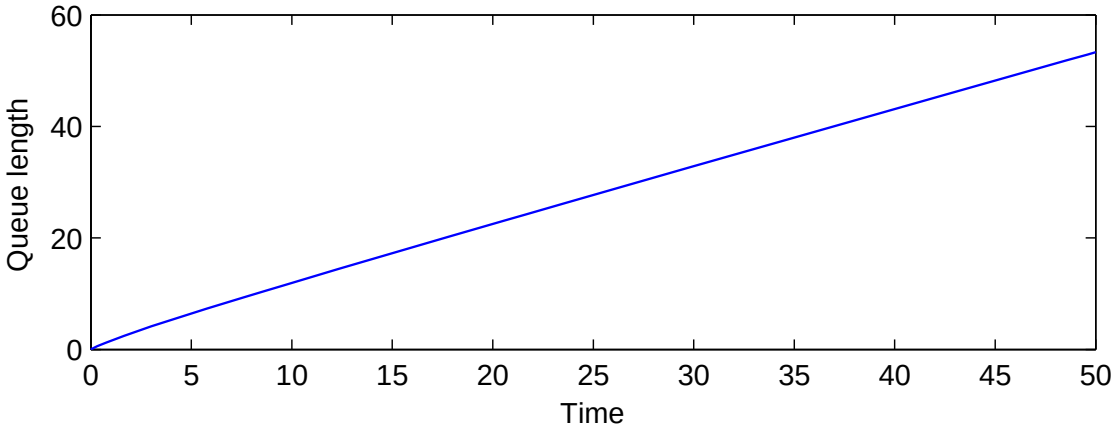
$$\dot{x} = \lambda - \mu \frac{x}{x+1}$$

Queue Length Control: Model

$\lambda = 0.5, \mu = 1$



$\lambda = 2.0, \mu = 1$



Queue Length Control: Model

Control the queue length by only admitting a fraction u (between 0 and 1) of the requests

$$\dot{x} = \lambda u - \mu \frac{x}{x+1}$$

Admission control

Queue Length Control: Linearization

Linearize around $x = x^\circ$

Let $y = x - x^\circ$

$$\dot{y} = \lambda y - \mu \frac{1}{(x^\circ + 1)^2} y = \lambda u - \mu a y$$

Queue Length Control: P-Control

$$u = K(r - y)$$

$$\dot{y} = \lambda K(r - y) - \mu a y$$

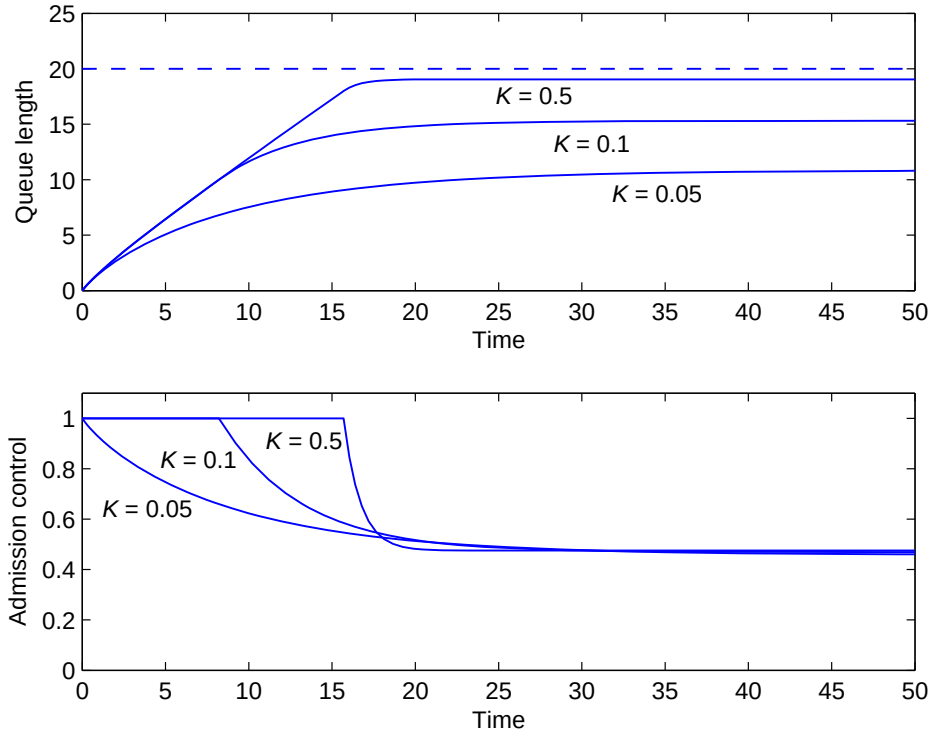
$$(s + \lambda K + \mu a)Y(s) = \lambda K R(s)$$

$$G_{cl}(s) = \frac{\lambda K}{s + \lambda K + \mu a}$$

With K the closed loop poles can be placed arbitrarily

Queue Length Control: P-Control

Simulations for $\lambda = 2, \mu = 1, x^\circ = 20$ and different values of K



Stationary error

Nonlinear (control signal limitations)

Queue Length Control: PI-Control

$$G_P(s) = \frac{\lambda}{s + \mu a}$$

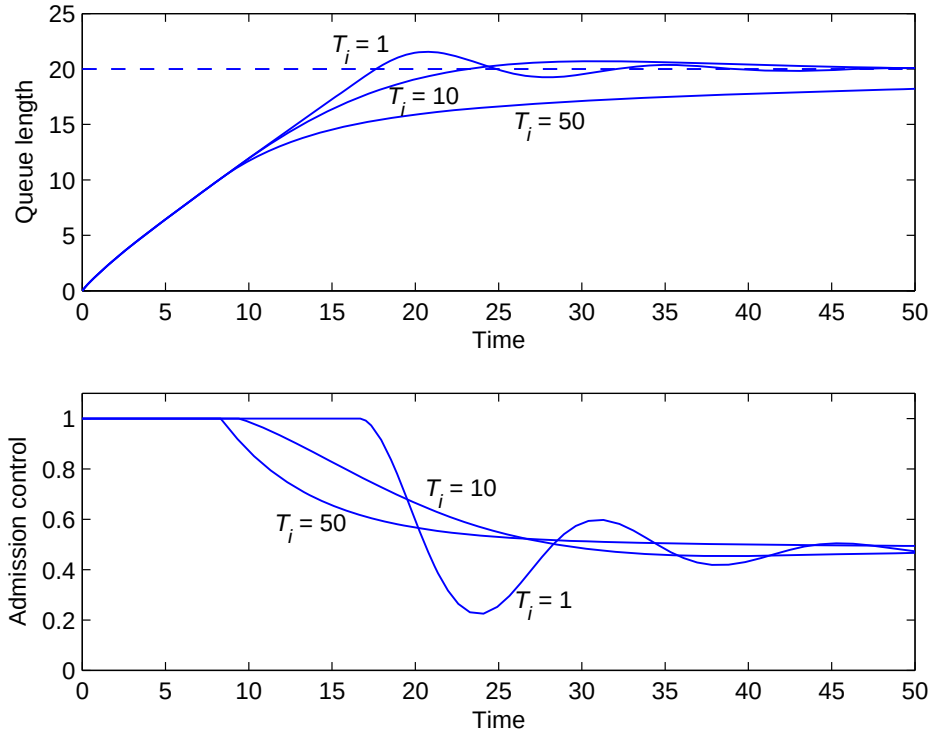
$$G_R(s) = K(1 + \frac{1}{sT_i})$$

$$G_{cl}(s) = \frac{G_P G_R}{1 + G_P G_R} = \frac{\lambda K(s + \frac{1}{T_i})}{s(s + \mu a) + \lambda K(s + \frac{1}{T_i})}$$

With K and T_i the closed loop poles can be placed arbitrarily

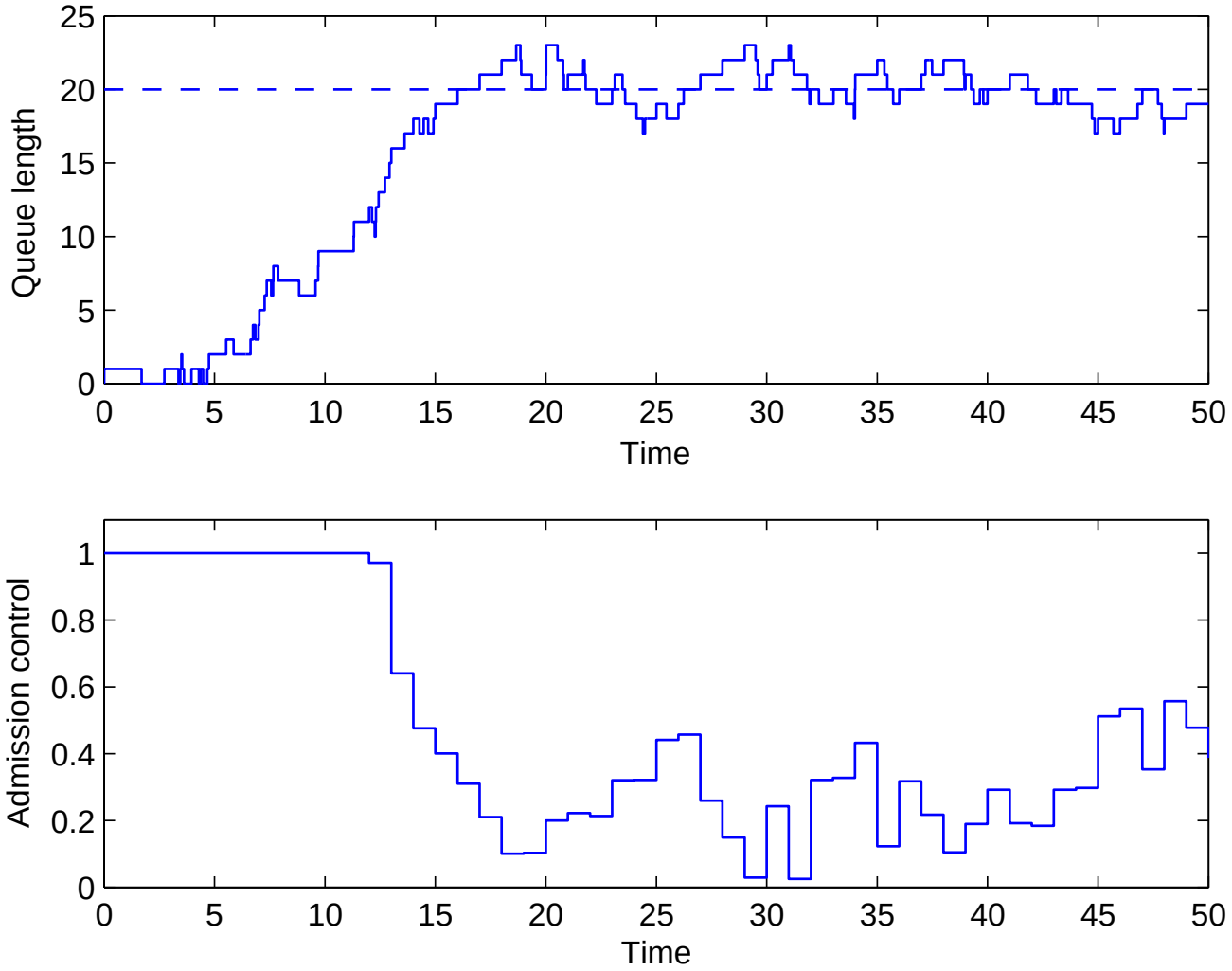
Queue Length Control: PI-Control

Simulations for $\lambda = 2, \mu = 1, x^\circ = 20, K = 0.1$ and different values of T_i

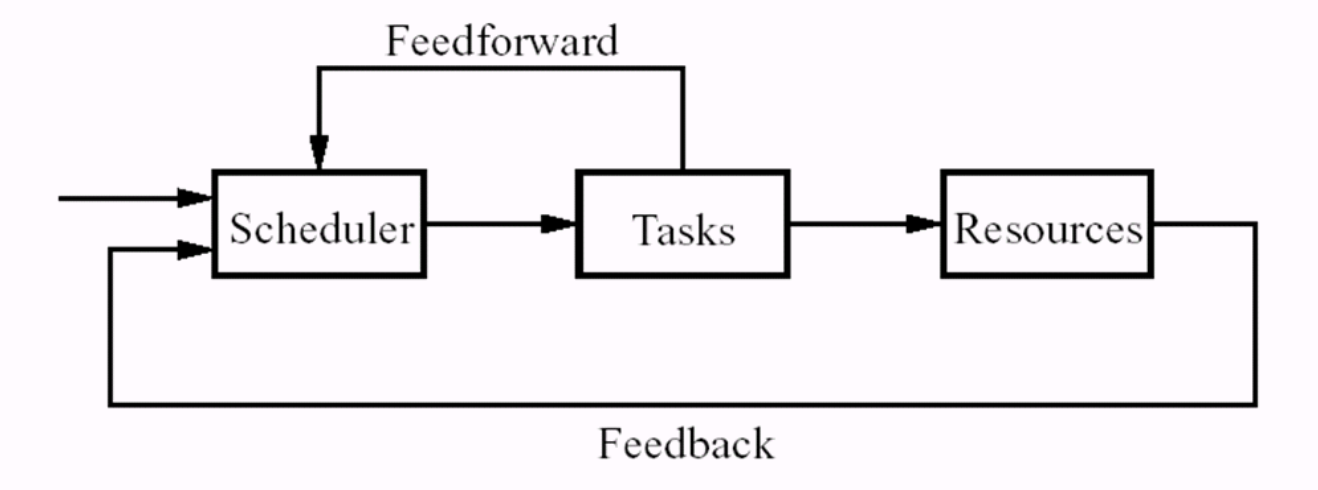


Stationary error removed

Queue Length Control: PI-Control on Process



Example: Task Scheduling



Control CPU utilization by adjusting

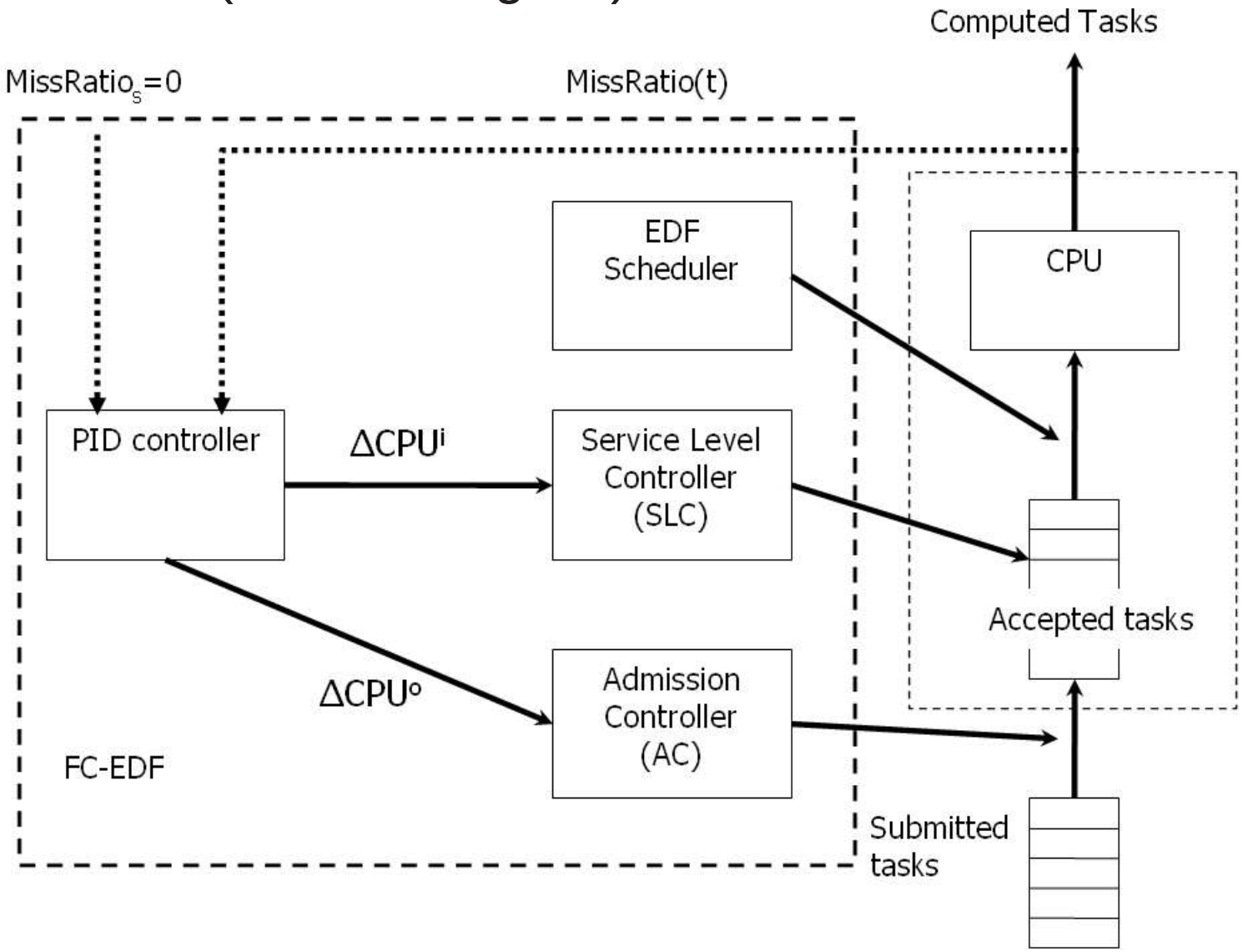
- task periods
- task execution demands
- priorities

Setpoint = schedulability bound

Feedforward to handle mode changes

Feedback Control Real-Time Scheduling

Stankovic et al (Univ of Virginia)



Example: Numerical Integration

The automatic step-size adjustment in numerical integration routines for ODEs (ordinary differential equations) can be cast as a control problem

Ordinary PI/PID control works well

PhD thesis by Kjell Gustafsson, Dept of Automatic Control, Lund 1992

General Observations

The plant under control rarely have any real dynamics or only very simple dynamics

- static nonlinearities + time delays (possibly time-varying)
- first or (maybe) second-order dynamics

Dynamics introduced through the sensors

- Time averages

Event-based control seems a more natural approach than time-based (though very few try to apply it)

General Observations

Seldom any measurement noise

- high gain feedback a possibility

Decentralized control in communication with local and global contstraints

- control the resource allocation of tasks or jobs
- local minimum constraints
- global maximum contstraints
  - schedulability conditions
  - total available amount of resource limited (e.g. power)

General Observations

Much to learn from control engineering

- Control principles
- Model-based design
- Optimization-based design

Simple controllers often enough

- P, I, PI + feedforward, PD
- anti-windup to achieve good performance

Lack of first principles knowledge that can be used to derive models

- queuing systems an exception
  - however, the models here are averages over long time horizons
  - how use these for control?
- models often derived from input-output data (be cautious)

General Observations

So far, primarily applications of classical linear and non-linear time-driven control

It could be expected that there is room for special control theory developed to better fit these types of application

The interest for this area is currently higher in the computer community than in the control community (unfortunately)

Outline

- Overview
- **Nice Control of Linux**
- Feedback Scheduling of Control Tasks
  - Infinite Horizon
  - Finite Horizon
  - MPC
- Control of Web servers

Nice Control of Linux

Idea:

- Reservation-based control of the CPU resources in Linux

Measurements:

- The CPU bandwidth of a Linux non-realtime task

Control signal:

- The nice value of the task (related to the task priority)

Controller:

- PI-controller

Applications:

- Web server control

CPU Bandwidth – Definition

A task's CPU bandwidth is defined as

$$\text{CPU bandwith} = \frac{\text{a task's execution time during a time interval}}{\text{the length of the same time interval}}$$

CPU Bandwidth – Sensors and Actuators

What is already available?

- A task's total execution time is available in kernel space, i.e., inside the operating system.
- The CPU bandwidth is affected by the UNIX nice value.

nice Value – What is it?

The nice value can be set by the user on a per task level. It affects:

- The task's priority.
- The size of the time slice given to the task.

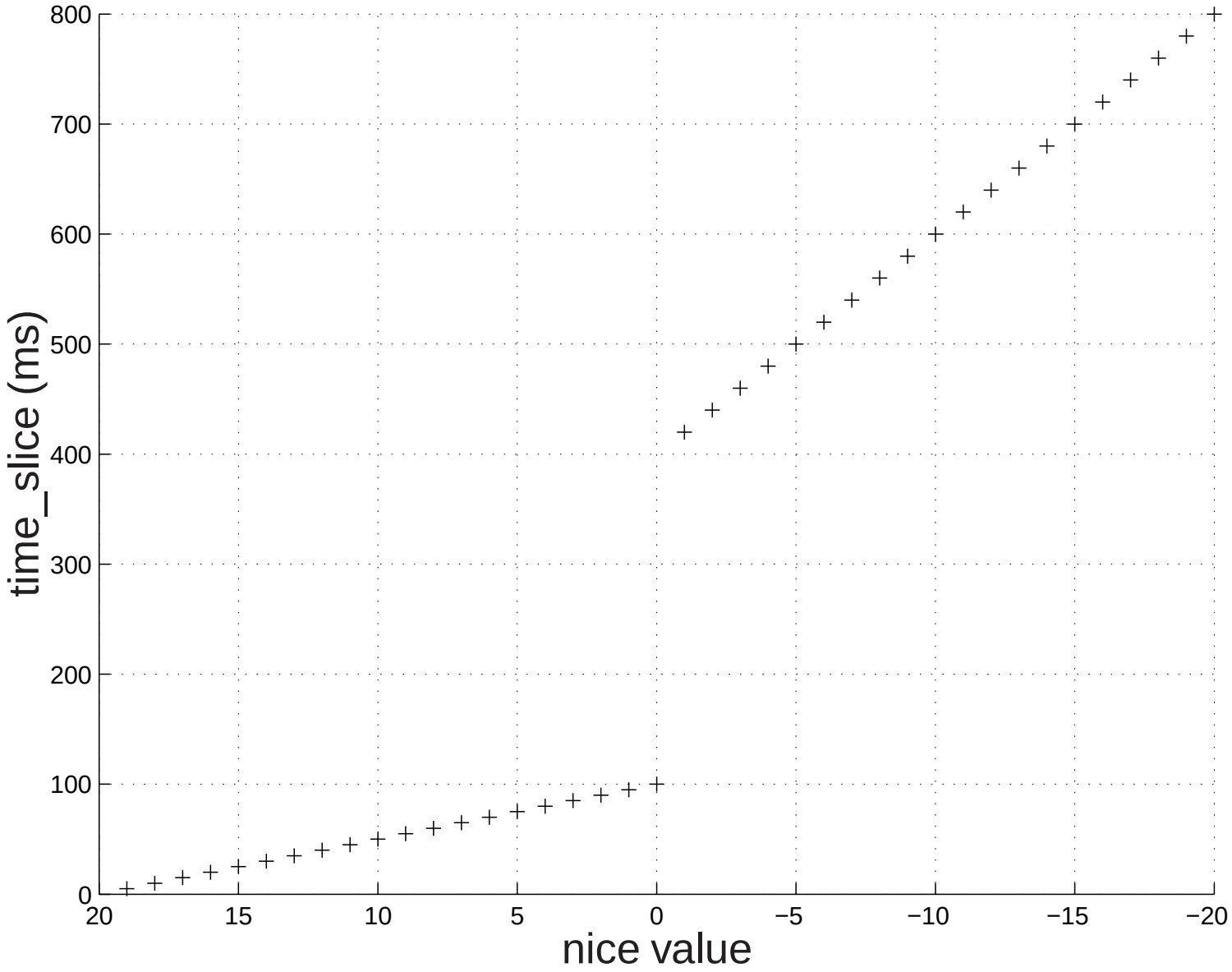
Normally used to run things in the background.

Simple Model

In simple words:

- Every task gets a time slice whose size depends on the nice value given to that task.
- When a task has executed for its whole timeslice, every other task gets to execute for their whole timeslices before the first task is executed again.

Size of the Time Slize



Example

If, for example, tasks 1, 2 and 3 have nice value 0 and task 4 has nice value -1. Then task 4 will get:

$$fraction(i) = \frac{time\_slice(i)}{\sum_{\forall j} time\_slice(j)} = \frac{420}{3 * 100 + 420} \approx 58\%$$

Under the assumption that all tasks are willing to run.

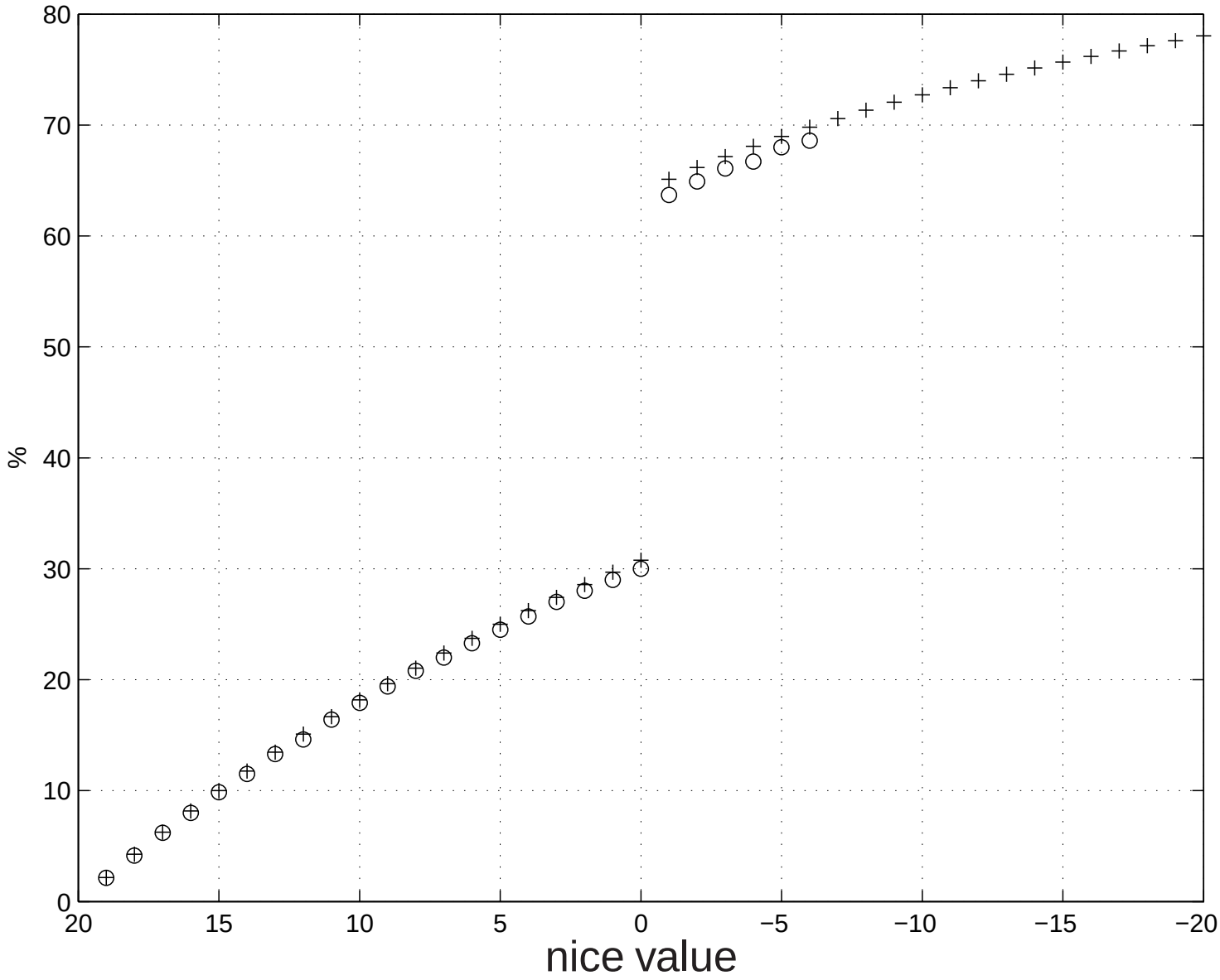
Evaluation of the Model

Setup:

- Four tasks running in endless while loops.
- For one of the tasks, the nice value is changed from 19 to -7 and the CPU bandwidth time is measured.

$+$ is according to the model.
$\mathbf{o}$ is measured values.

Evaluation



Controller

- A kernel module has been implemented that periodically samples task data and measures the execution time of a task between two samples.
- A PI-Controller has been implemented in the same kernel module. $K = 0.01$ and $T_i = 52$.
- The sampling time is 20 ms.

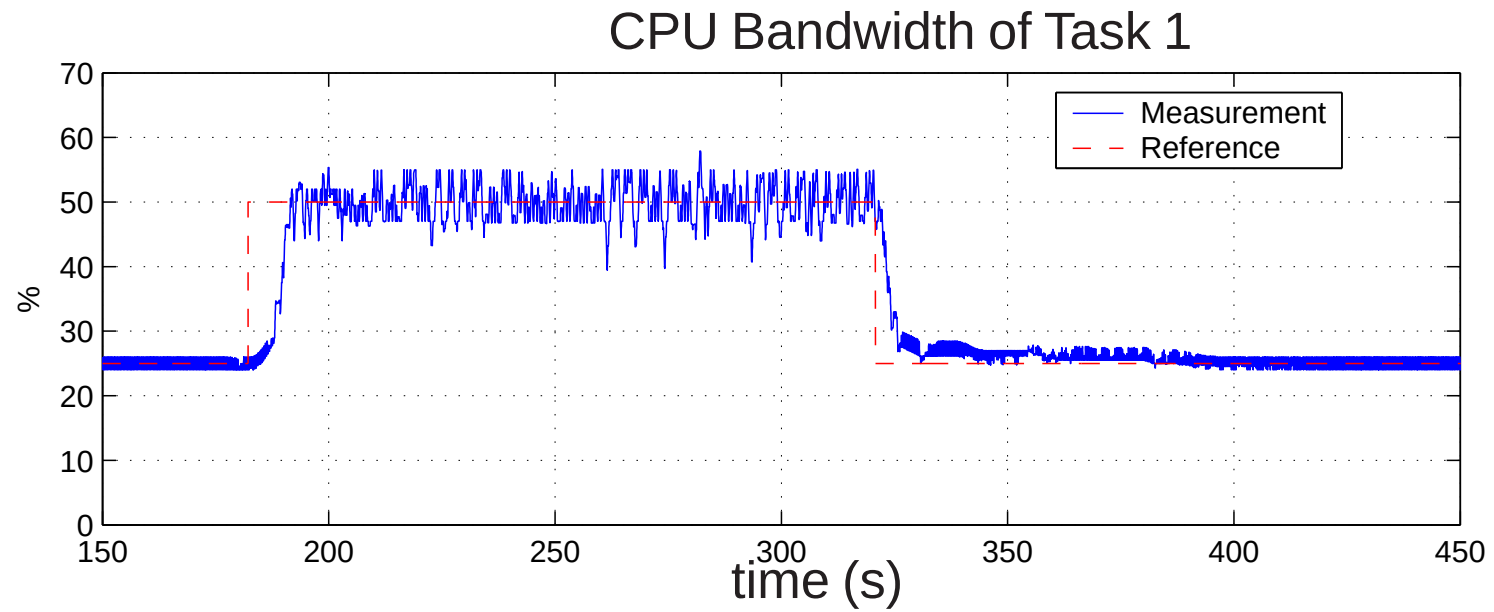
Experiments

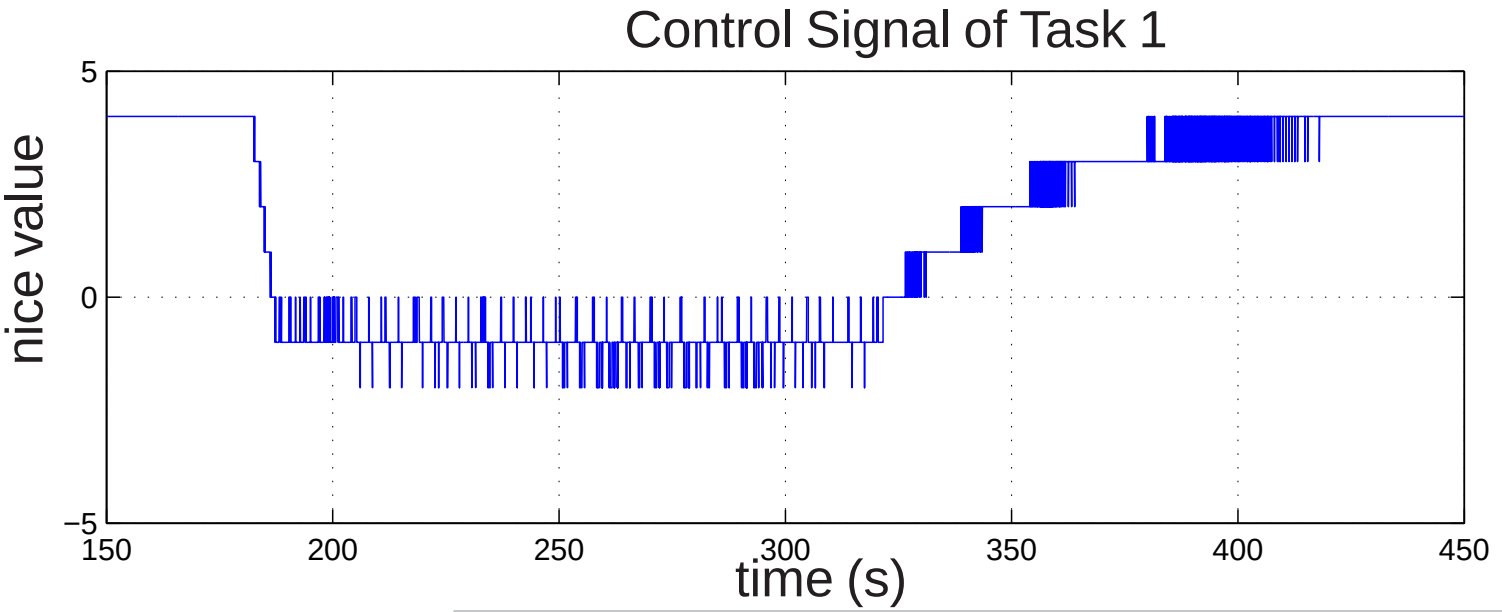
- All experiments have been performed on the author's desktop computer.
- Bandwidth measurements have been filtered using a moving average window of 4 s.
- No special steps were taken before the measurements, i.e., the computer is at the same time running normal programs such as Firefox, Thunderbird . . .

Experiment: Step Response

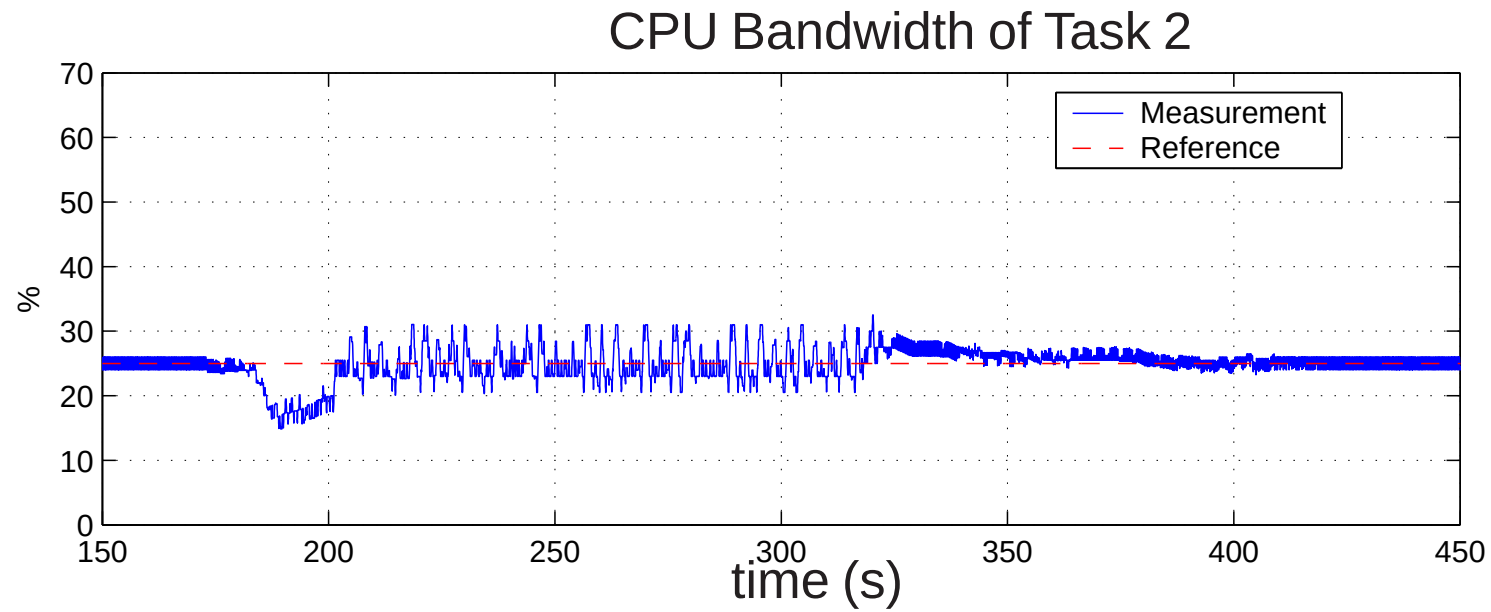
- Four tasks running in endless while-loops.
- Two of them are controlled.
- The other two have a static nice value of 5.
- Reference 1 is changed as a square-wave while reference 2 is kept at 25%.

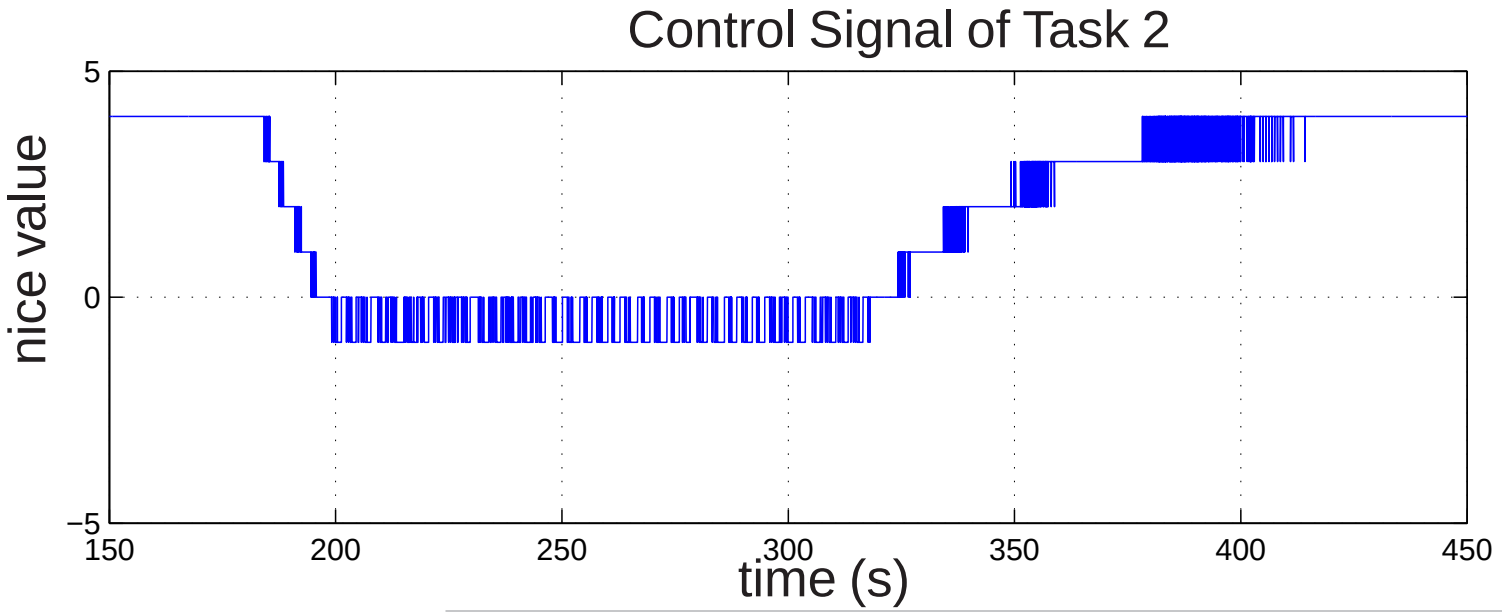
Step Response





Step Response Cont'd





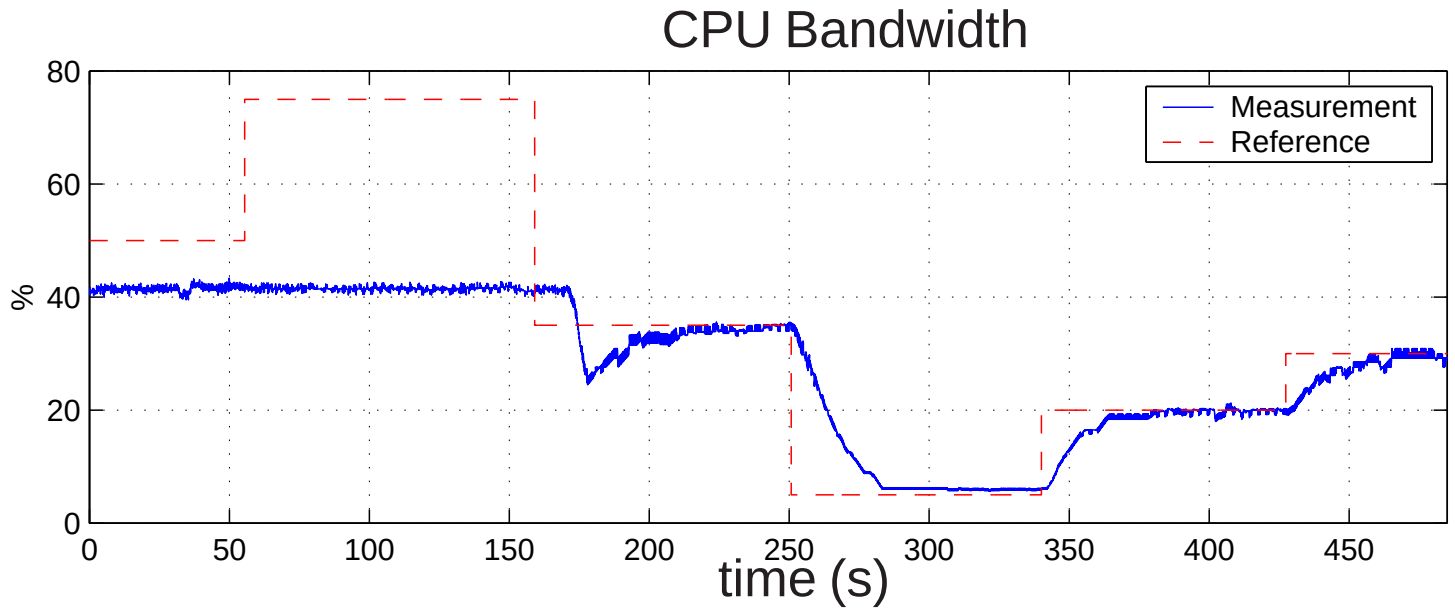
Taking the State Into Account

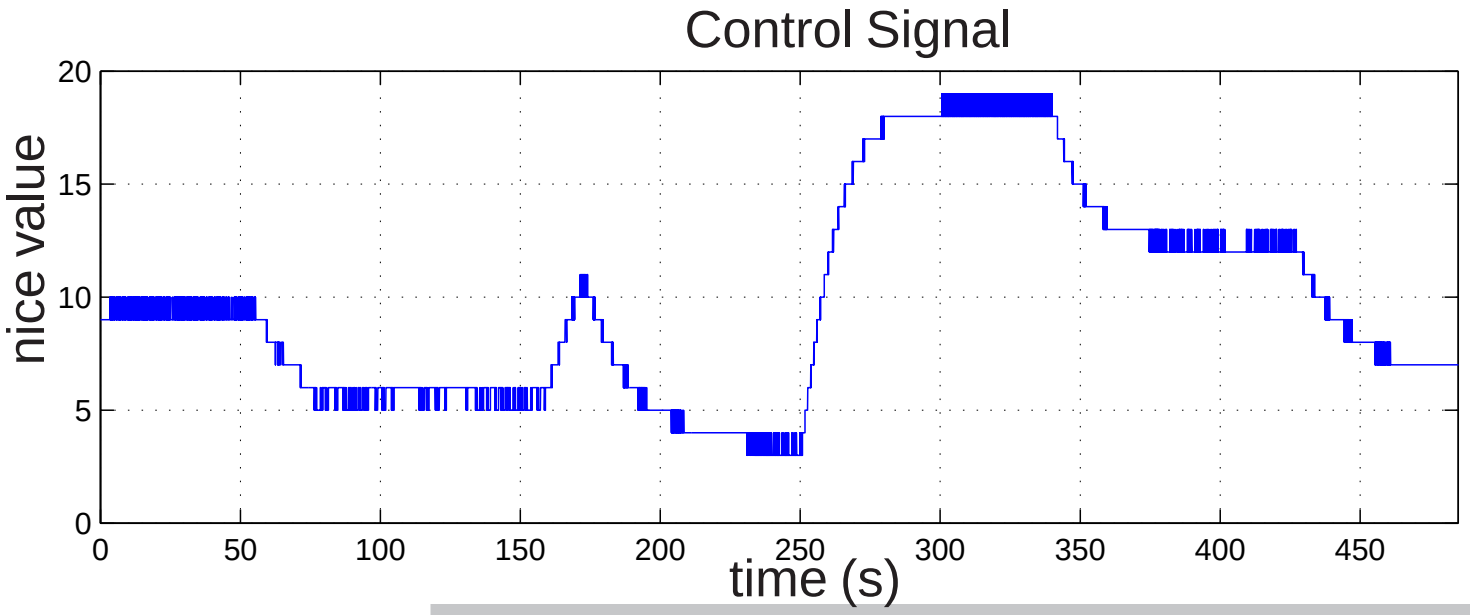
It is also possible to control the CPU bandwidth of tasks that are not always willing to run. This makes the presented concept much more useful.

Experiment 4 – Taking the State Into Account

- One controlled task that refuses to use more than 40% of the CPU bandwidth.
- Two tasks running in endless while-loops with nice value 5.

Taking the State Into Account

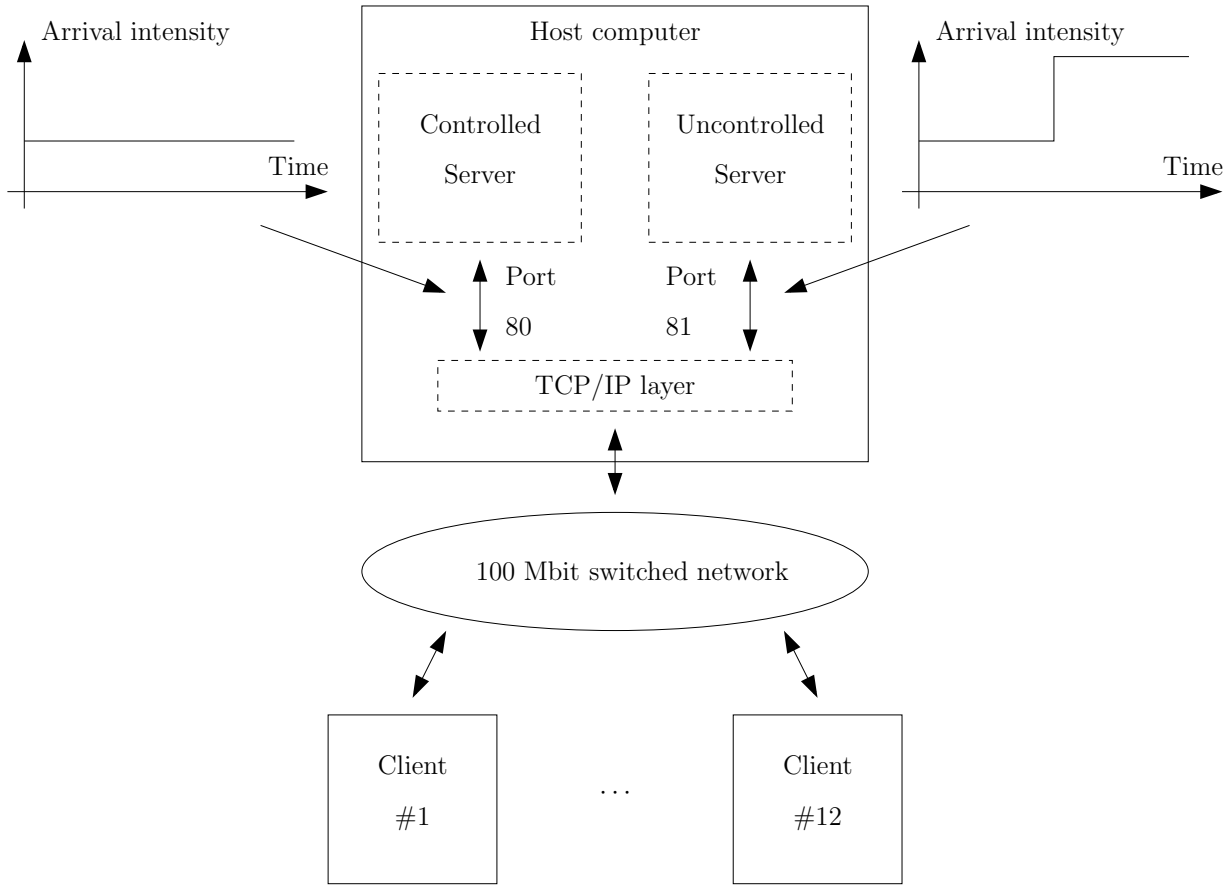




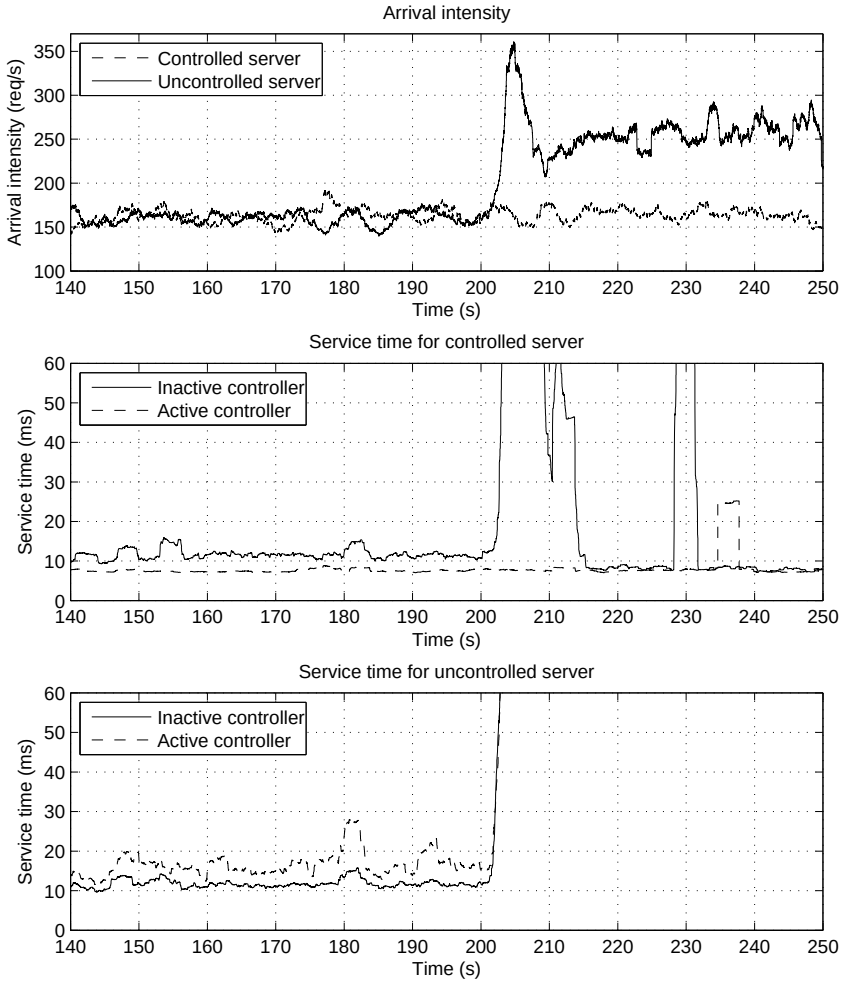
Apache Web Server Control

Two web servers hosted on the same CPU (Pentium 4)

Traffic generated by 12 client computers



Apache Web Server Control: Results



Top: Average arrival rates.

Middle: Response time for the controlled server with and without feedback control.

Bottom: Response time for the uncontrolled server.

Outline

- Overview
- Nice Control of Linux
- **Feedback Scheduling of Control Tasks**
  - Infinite Horizon
  - Finite Horizon
  - MPC
- Control of Web servers

Resource Allocation as a Control problem

In applications with multiple tasks or jobs the dynamic allocation of resources to the tasks can be viewed as a control problem in itself

Use feedback as a technique for mastering uncertainty and guaranteeing performance

Feedback Scheduling

Dynamic on-line allocation of computing resources

Feedback from actual resource utilization

In principle, any computing resource

Here,

- Scheduling of the execution of real-time tasks
- In particular, real-time controller tasks

Feedback scheduling

The Problem: Ordinary priority-based scheduling may fail:

- Design based on worst-case execution times:
  - Slow sampling, all deadlines are met
  - Low utilization, low control performance
- Design based on average-case execution times:
  - Faster sampling, many deadlines are missed
  - Higher utilization, low control performance

Solution: dynamic scheduling based on feedback

Optimal Feedback Control

Mostly ad-hoc approaches

Adjust sampling periods and/or execution time demands so that the task set is schedulable

In control it is important to take the application performance into account

- E.g. adjust scheduling parameters in such a way that the global performance is optimized

Requires performance metrics that are properly parameterized

- JitterBug

Outline

- Overview
- General Observations
- Nice Control of Linux
- Feedback Scheduling of Control Tasks
  - **Infinite Horizon**
  - Finite Horizon
  - MPC
- Control of Web servers

Linear Rescaling

A linear proportion rescaling of the nominal sampling periods to meet the utilization set-point is optimal w.r.t. global control performance for:

- quadratic cost functions $J_i(h_i) = \alpha_i + \beta_i h_i^2$
- linear cost functions $J_i(h_i) = \alpha_i + \gamma_i h_i^2$

where the objective is to minimize the sum or the weighted sum of the control cost functions

$$h_{\text{new}} = \frac{h_{\text{nom}} U}{U_{\text{sp}}}$$

Linear Rescaling

Great!

- Simple and fast
- Preserves rate-monotonic ordering
- Good approximations of true cost functions

Linear Rescaling

Additional constraints can be added:

- Use nominal sampling periods as minimum sampling periods and use these whenever the utilization is less that the utilization set-point

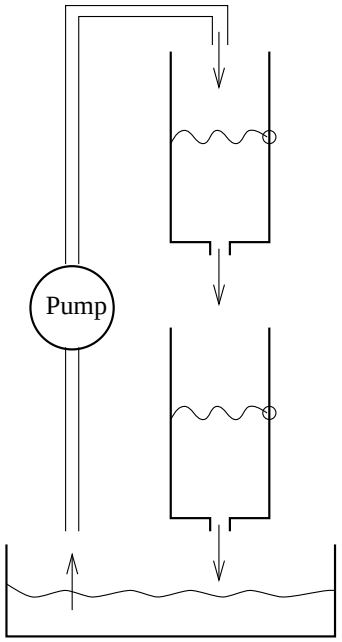
Linear rescaling property does not always hold:

- Mixed task set
- Tasks with maximum sampling periods $\Rightarrow$ iterative LP computation

Case Study: Set of Hybrid Controllers

The double-tank process:

Use pump, $u(t)$, to control level of lower tank, $y(t)$



Hybrid control strategy:

- PID control in steady state
- Optimal control for setpoint changes

PID Controller

$$
\begin{aligned}
P(t) &= K(y_{sp}(t) - y(t)) \\
I(t) &= I(t-h) + a_i(y_{sp}(t) - y(t)) \\
D(t) &= a_d D(t-h) + b_d(y(t-h) - y(t)) \\
u(t) &= P(t) + I(t) + D(t)
\end{aligned}
$$

Average execution time: $C = 2.0$ ms

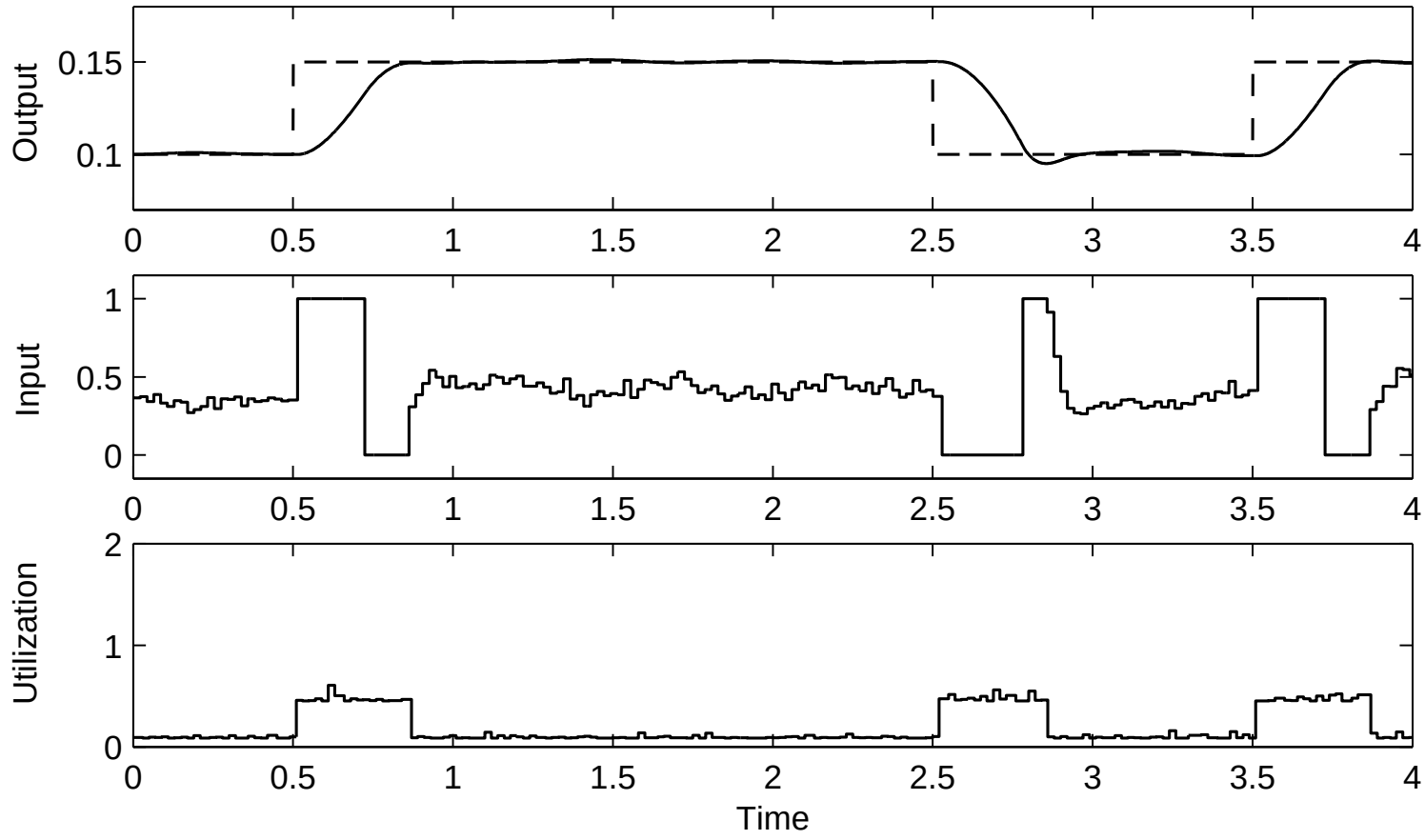
Optimal Controller

$$x_2(x_1) = \frac{1}{a}((ax_1 - b\overline{u})(1 + \ln(\frac{ax_1^R - b\overline{u}}{ax_1 - b\overline{u}})) + b\overline{u})$$

$$V_{close} = \begin{bmatrix} x_1^R - x_1 \\ x_2^R - x_2 \end{bmatrix}^T P(\theta, \gamma) \begin{bmatrix} x_1^R - x_1 \\ x_2^R - x_2 \end{bmatrix} + \text{more } \ldots$$

Average execution time: $C = 10.0$ ms

Nominal Behavior, $h = 21\ \mathrm{ms}$



Scheduling Experiments

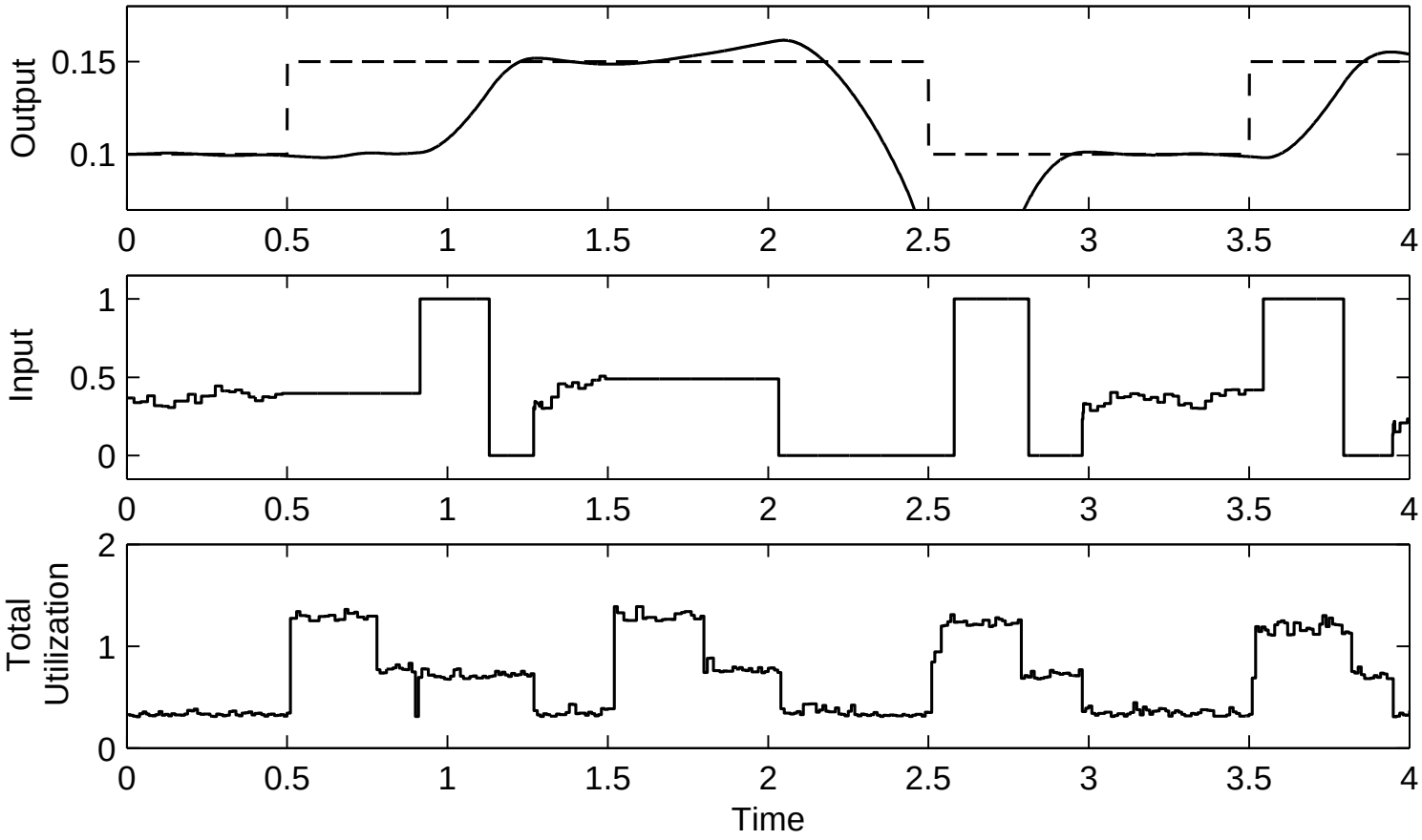
- Three hybrid controllers execute on one CPU
- Nominal sampling periods: $(h_1, h_2, h_3) = (21, 18, 15)$ ms
- Potential problem: All controllers in Optimal mode $\Rightarrow$ $U = \sum \frac{C}{h} = 170\%$

Compare strategies:

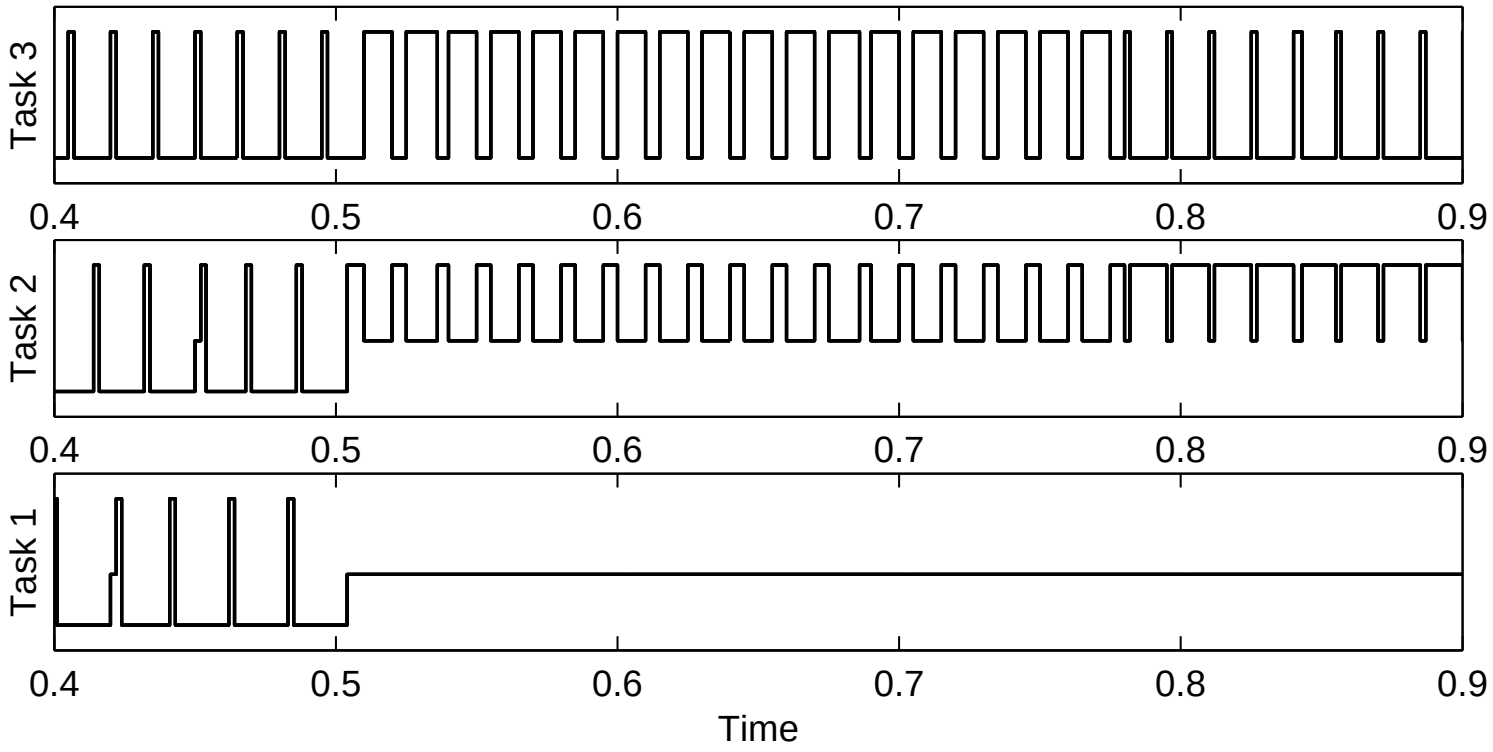
1. Open-loop scheduling
2. Feedback scheduling
3. Feedback + feedforward scheduling

- Co-simulation of scheduler, controllers, and double tanks
- Focus on the lowest-priority controller

Open-Loop Scheduling

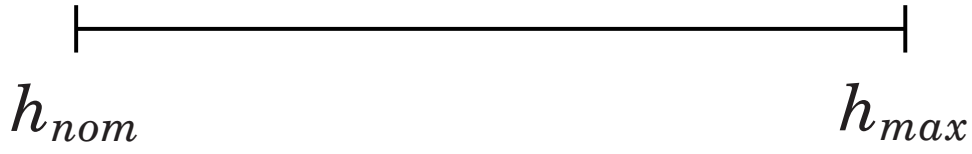


Open-Loop Scheduling

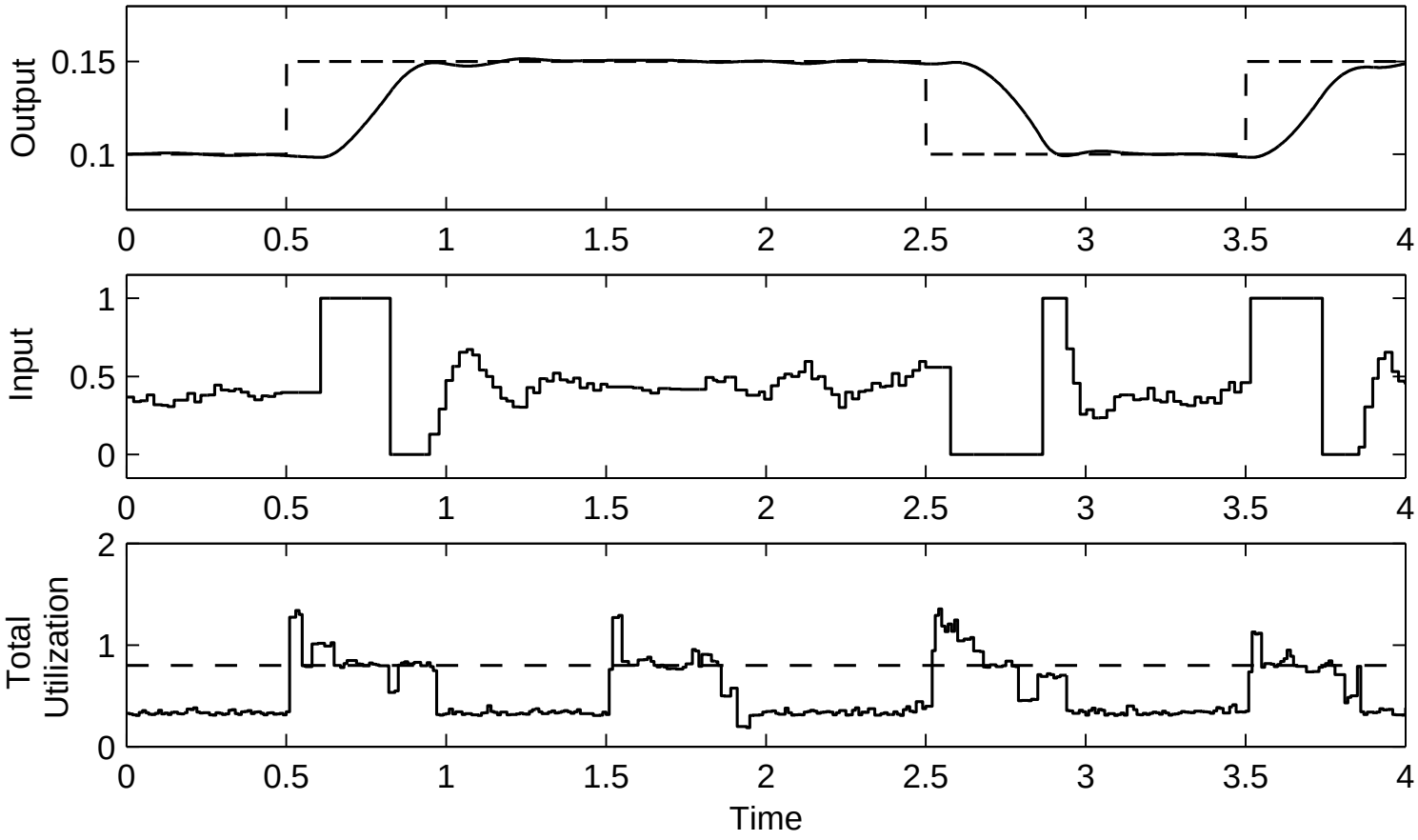


Feedback Scheduler

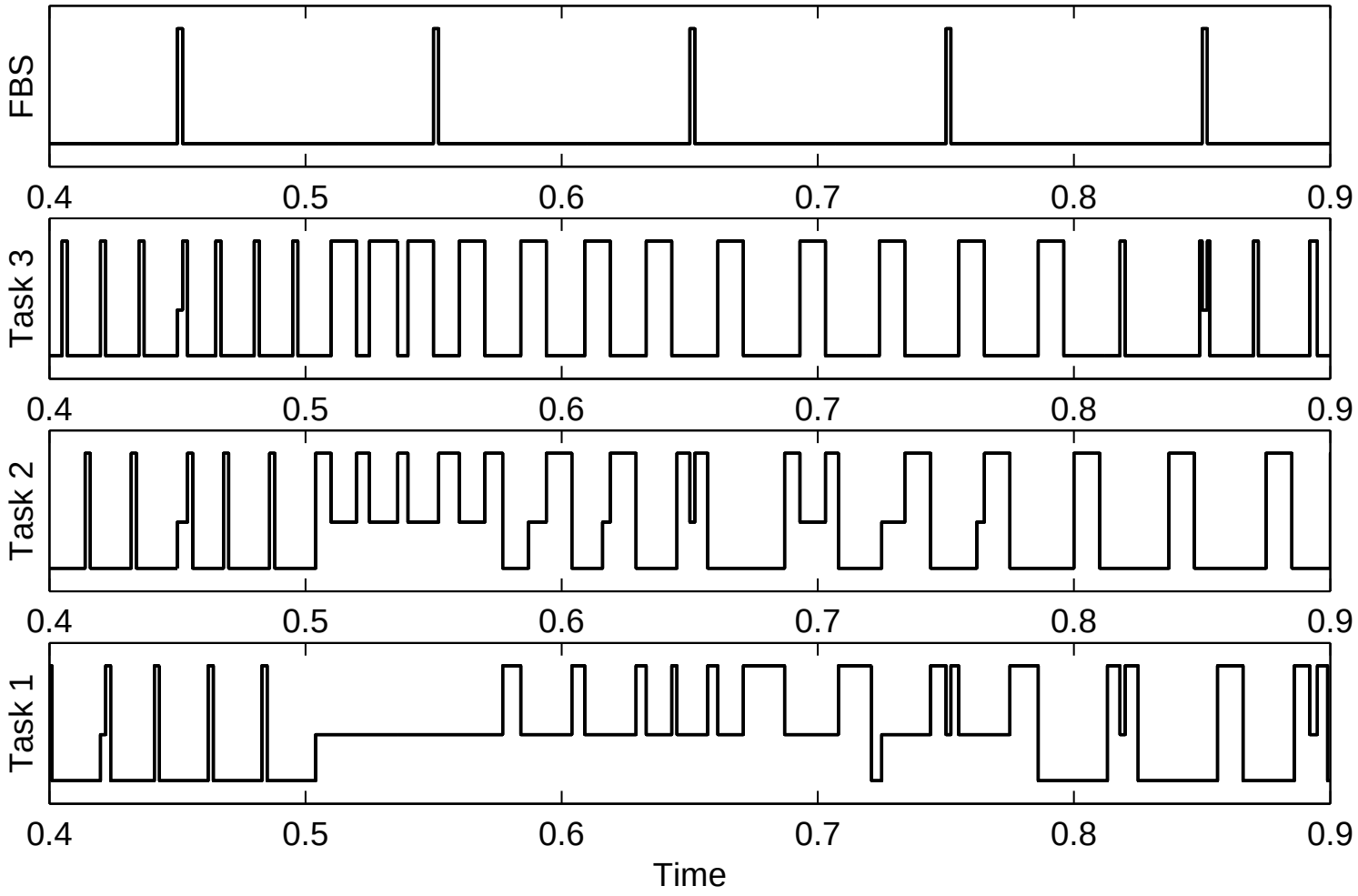
- A high-priority task, $T_{FBS} = 100$ ms, $C_{FBS} = 2$ ms
- Setpoint: $U_{sp} = 80\%$
- Estimate execution times using first-order filters
- Control U by adjusting the sampling periods
  - Simple linear rescaling
  - No regard for control performance
  - Range of acceptable sampling intervals



Feedback Scheduling



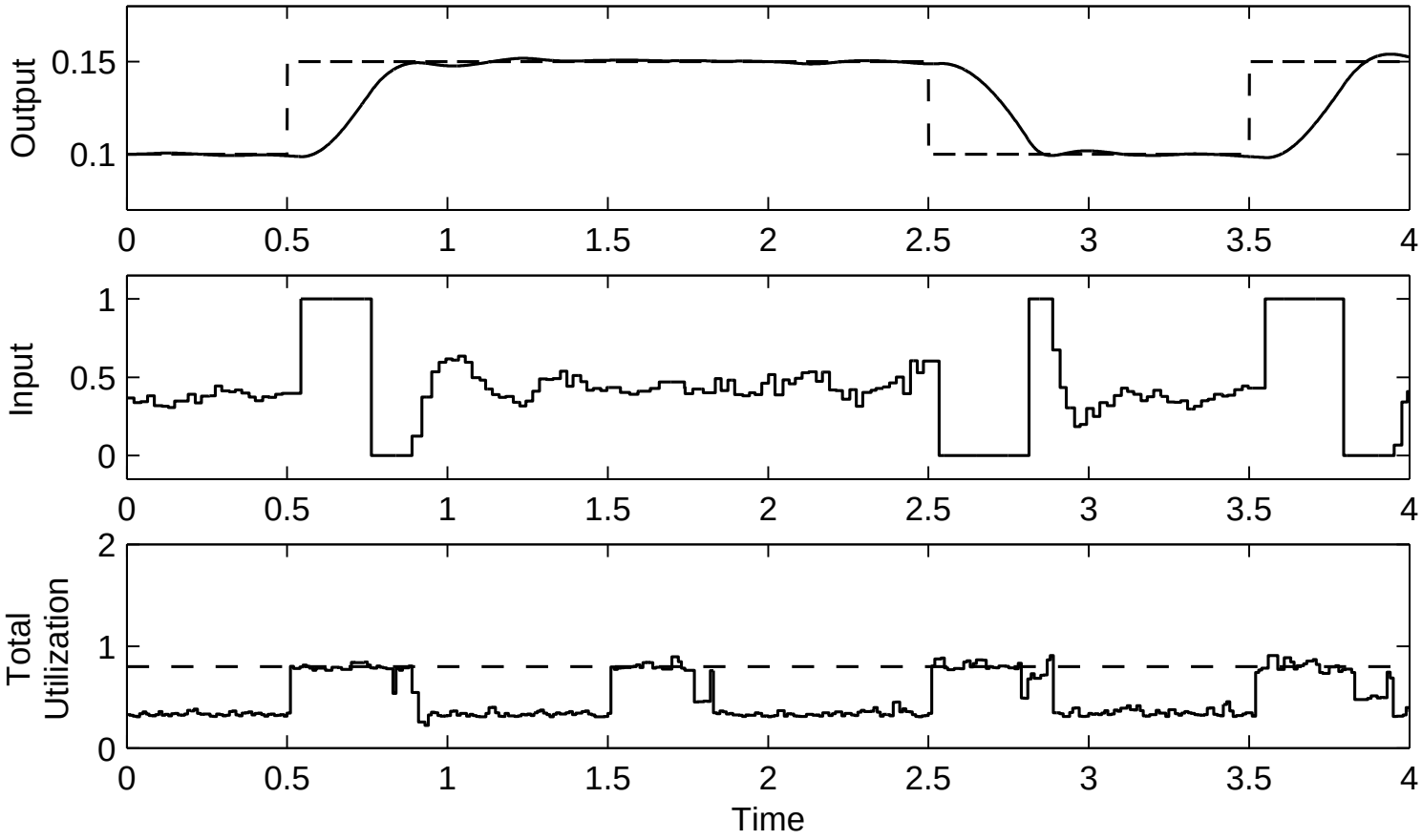
Feedback Scheduling



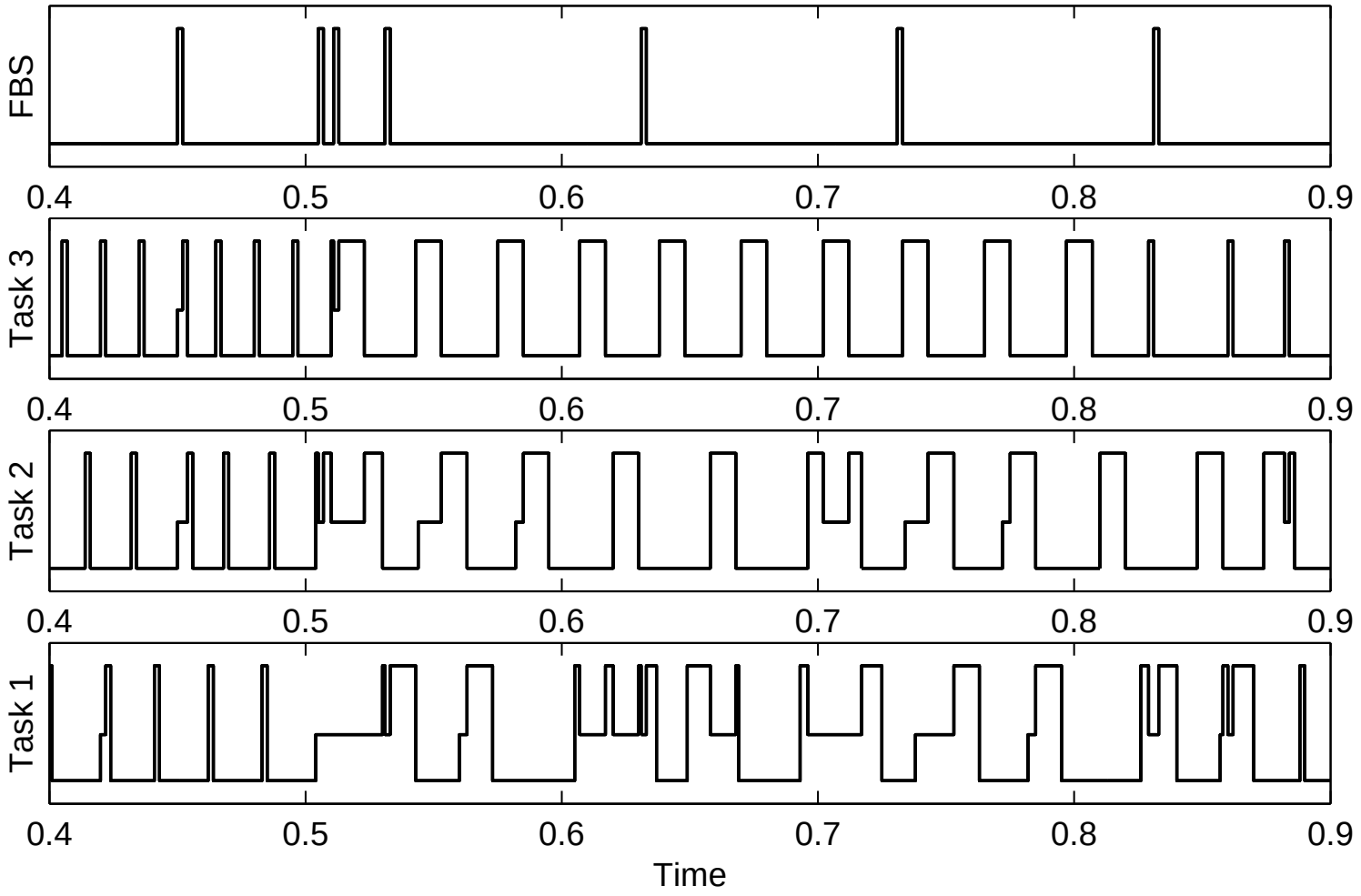
Feedforward

- Controller notifies feedback scheduler when switching from PID to Optimal mode
- Scheduler is released immediately
- Separate execution-time estimators in different modes

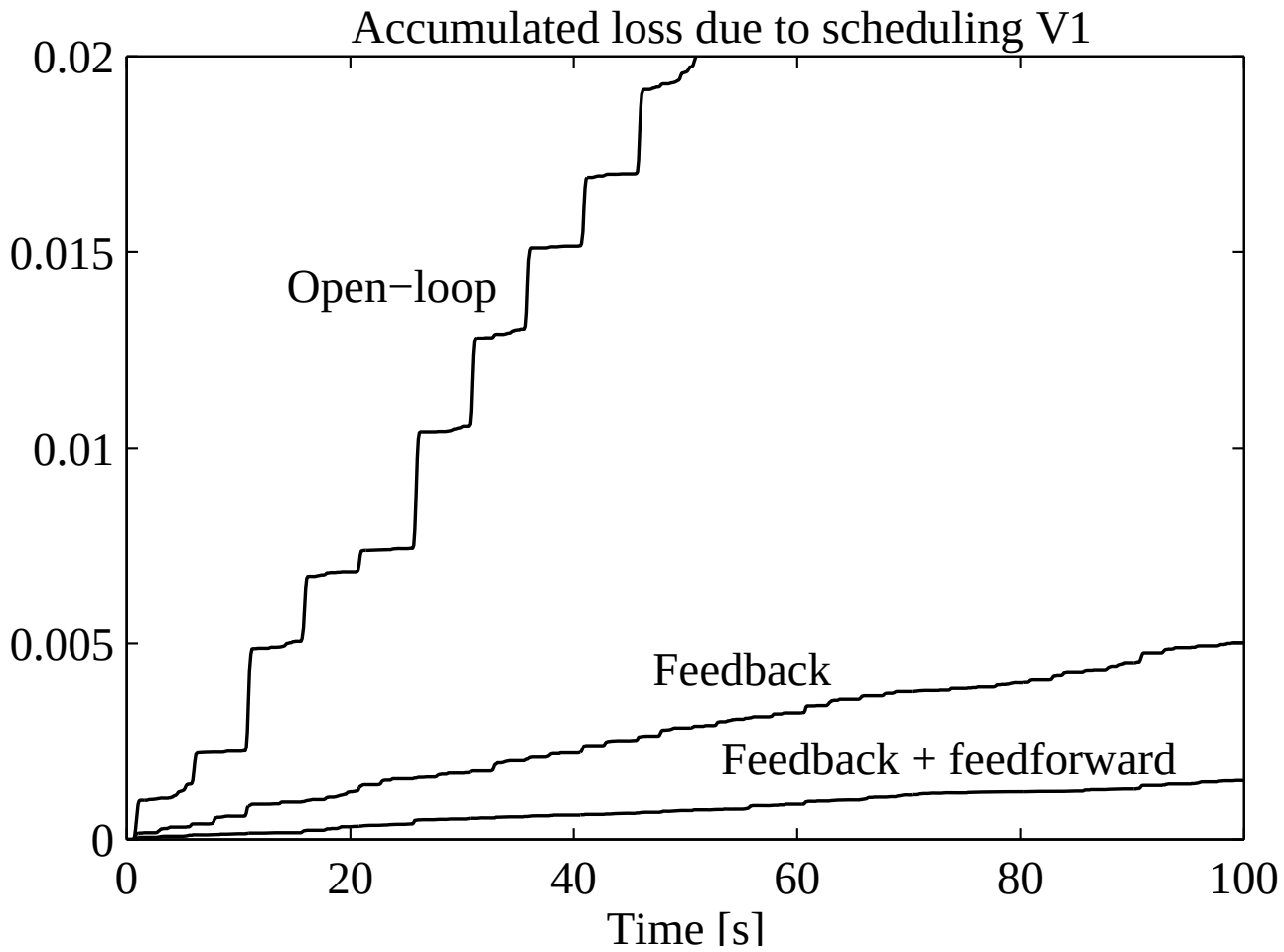
Feedback + Feedforward Scheduling



Feedback + Feedforward Scheduling



Control Performance Evaluation



Disclaimer

But:

- Cost functions over infinite horizons
- Does not take the plant state into account

Outline

- Overview
- General Observations
- Nice Control of Linux
- Feedback Scheduling of Control Tasks
  - Infinite Horizon
  - **Finite Horizon**
  - MPC
- Control of Web servers

Feedback Scheduling Version 2

- Finite time horizons
- Takes the state of the controlled plant into account

Optimal Period Assignment

Assume that the performance of each controller i can be described by a cost function $J_i(x_i, h_i, T_{FBS})$

- x_i – the current state of plant i
- h_i – the sampling period of controller i
- T_{FBS} – the optimization horizon of the feedback scheduler

The objective is to minimize the combined performance with respect to the utilization bound:

$$\min_{h_1 \ldots h_n} \sum_{i=1}^{n} J_i(x_i, h_i, T_{FBS})$$

$$\text{subj. to } \sum_{i=1}^{n} \frac{C_i}{h_i} \leq U_{sp}$$

Optimal Period Assignment, cont'd

- Convex problem if functions $J_i(x_i, 1/f_i, T_{FBS})$ convex in f_i.
- Explicit solution if all cost functions have the same shape,

$$J_i = \alpha_i + \beta_i h_i^{\nu}$$

  - $\nu = 1$:

$$h_i = \left(\frac{C_i}{\beta_i}\right)^{1/2} \frac{\sum_{j=1}^{n} (C_j \beta_j)^{1/2}}{U_{sp}}$$

  - $\nu = 2$:

$$h_i = \left(\frac{C_i}{\beta_i}\right)^{1/3} \frac{\sum_{j=1}^{n} C_j^{2/3} \beta_j^{1/3}}{U_{sp}}$$

- Linear cost functions ($\nu = 1$) are often good approximations

Linear-Quadratic Controllers

The cost function for an LQ controller is given by

$$J(x, h, T_{fbs}) = x^T S(h) x + \frac{T_{fbs}}{h}\left(\operatorname{tr} S(h) R_1(h) + J_v(h)\right)$$

- $S(h)$ – solution to the LQ Riccati equation
- $R_1(h)$ – sampled process noise variance
- $J_v(h)$ – inter-sample cost term

Example: Integrator Process

- Process: $dx = u\,dt + dv_c$
  - v_c – Wiener process with unit incremental variance
- Design cost function: $J = \int_0^{T_{fbs}} x^2(t)\,dt$
- Resulting cost:

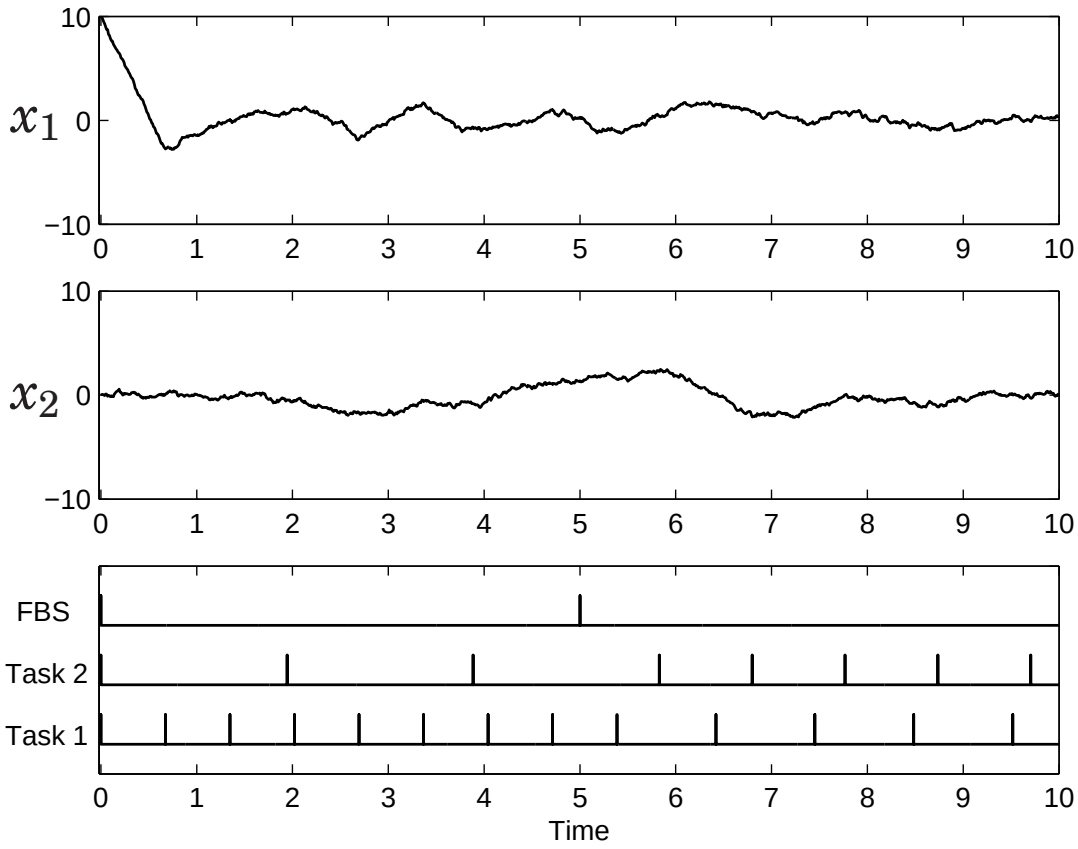
$$J(x, h, T_{fbs}) = \left(x^2 \frac{\sqrt{3}}{6} + T_{fbs} \frac{\sqrt{3}+3}{6} \right) h$$

  - Linear in h
  - Explicit solution for multiple controllers:

$$h_i \propto \sqrt{\frac{C_i}{x_i^2 + T_{fbs}(1+\sqrt{3})}}$$

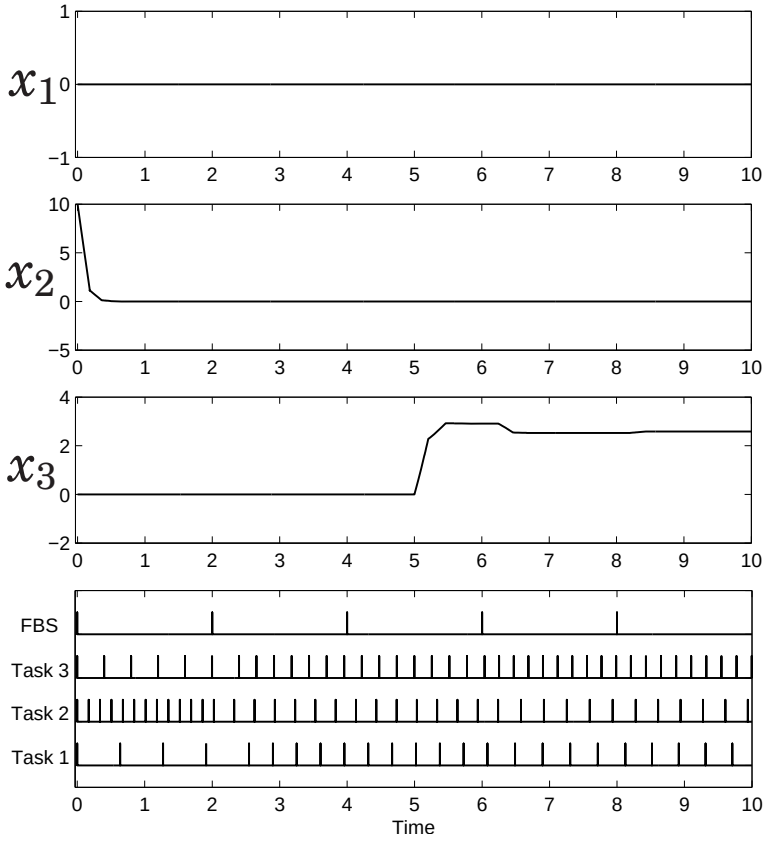
Simulation Example

- Two integrator processes with different initial conditions
  - $x_1(0) = 10,\ x_2(0) = 0,\ C_1 = C_2 = 0.5,\ U_{sp} = 1,\ T_{fbs} = 5$

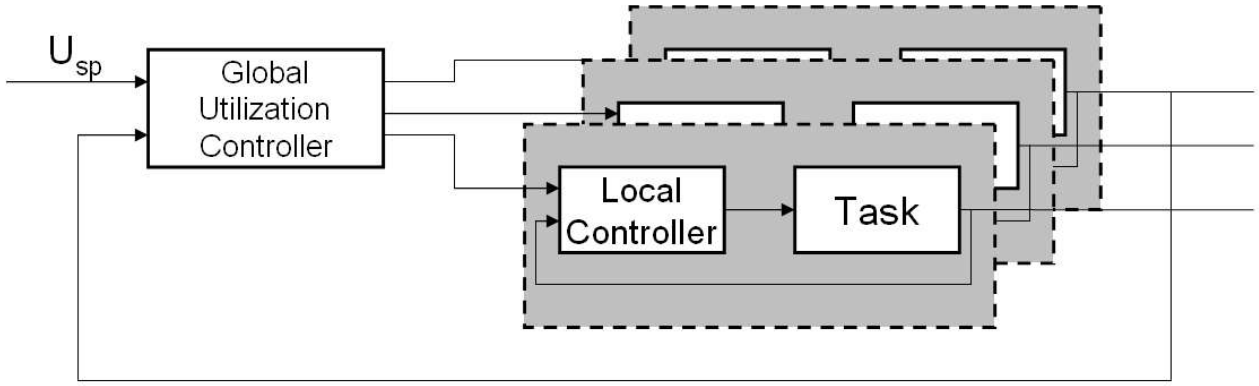


Simulation Example

- Three first-order plants with $a = -1$, $a = 0$, and $a = 1$
- Load disturbance affecting plant 3 at time $t = 5$
- $x_1(0) = 0$, $x_2(0) = 10$, $x_3(0) = 0$, $C = 0.1$, $U_{sp} = 1$, $T_{fbs} = 2$



Feedback Scheduling Structures



Cascaded/Layered Structure:

- Global utilization controller that outputs the desired utilization share for each task
- Local controllers that adjust the task parameters accordingly
- Combine with reservation-based scheduling to provide temporal protection, cp. the Control Server Model

Outline

- Overview
- General Observations
- Nice Control of Linux
- Feedback Scheduling of Control Tasks
  - Infinite Horizon
  - Finite Horizon
  - **MPC**
- Control of Web servers

Feedback Scheduling Actuators

Two main actuators:

- Changing the task periods
- Changing the allowed execution times

Task periods:

- Easy for simple controllers, e.g. PID & state feedback
- More difficult for complex controllers
- Update the internal state of the controller appropriately

Execution times:

- Not applicable to most controllers

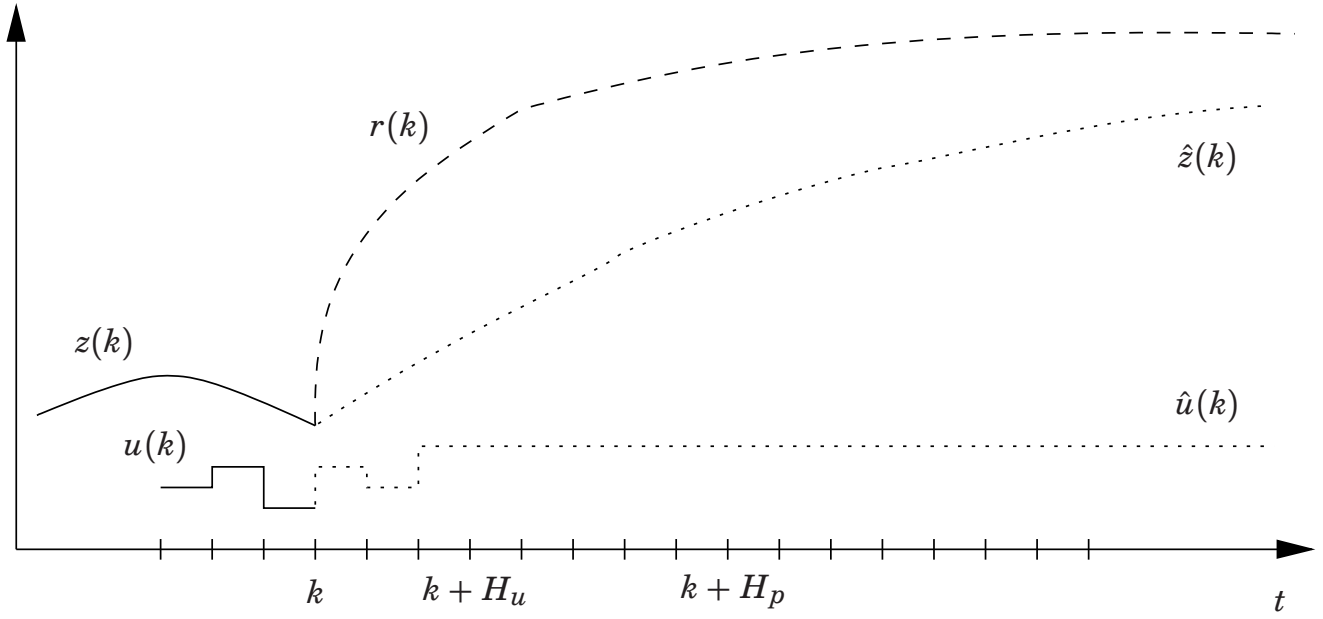
Anytime Controllers

Controllers where the quality of the control signal is gradually refined the more time that is available

Model-based Predictive Control (MPC)

- On-line convex optimization problem solved each sample
- Highly varying execution times
- For fast processes the latency may effect the control performance considerably
- The control algorithm is based on a quality-of-service type cost measure, cp instantaneous cost
- As long as a feasible control signal has been found the iterative search can be aborted before it has reached completion
- Maps well to the imprecise task model
  - Mandatory part
  - Optional part

Model Predictive Control

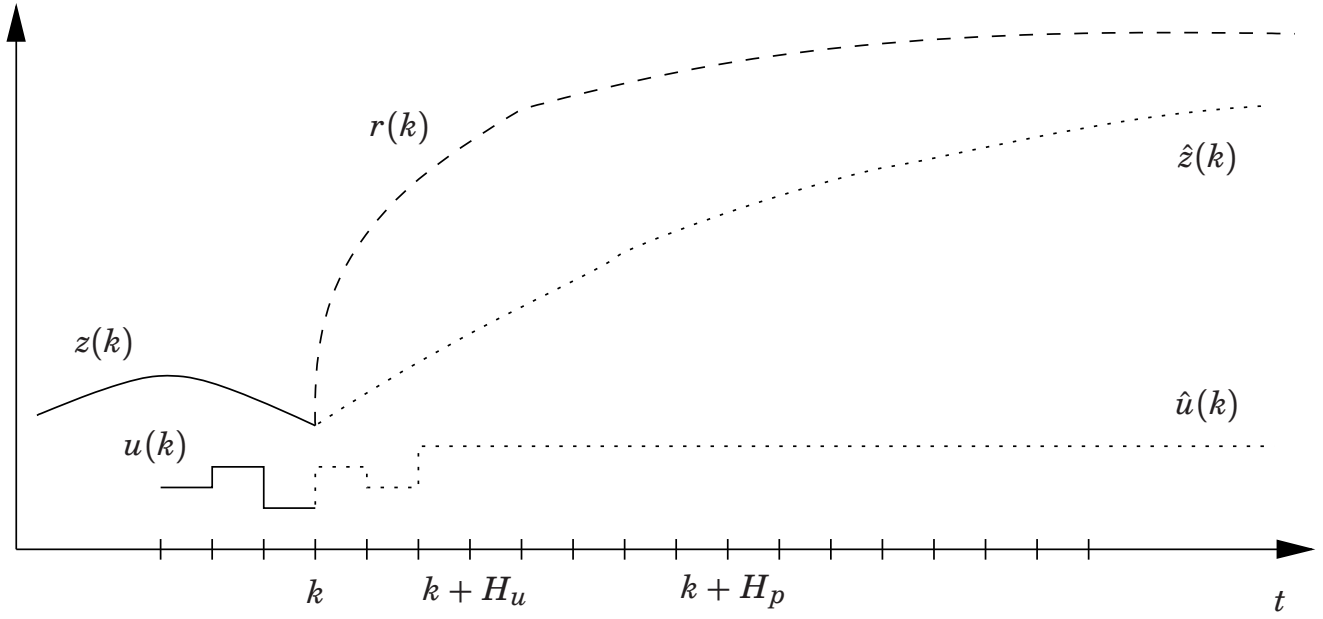


In *each sample*, find $\Delta\hat{u}(k) \ldots \Delta\hat{u}(k + H_u - 1)$ minimizing the cost

$$V(k) = \sum_{i=1}^{H_p} \|\hat{z}(k+i) - r(k+i)\|_Q^2 + \sum_{i=0}^{H_u-1} \|\Delta\hat{u}(k+i)\|_R^2$$

given constraints on control signals and controlled variables.

Model Predictive Control



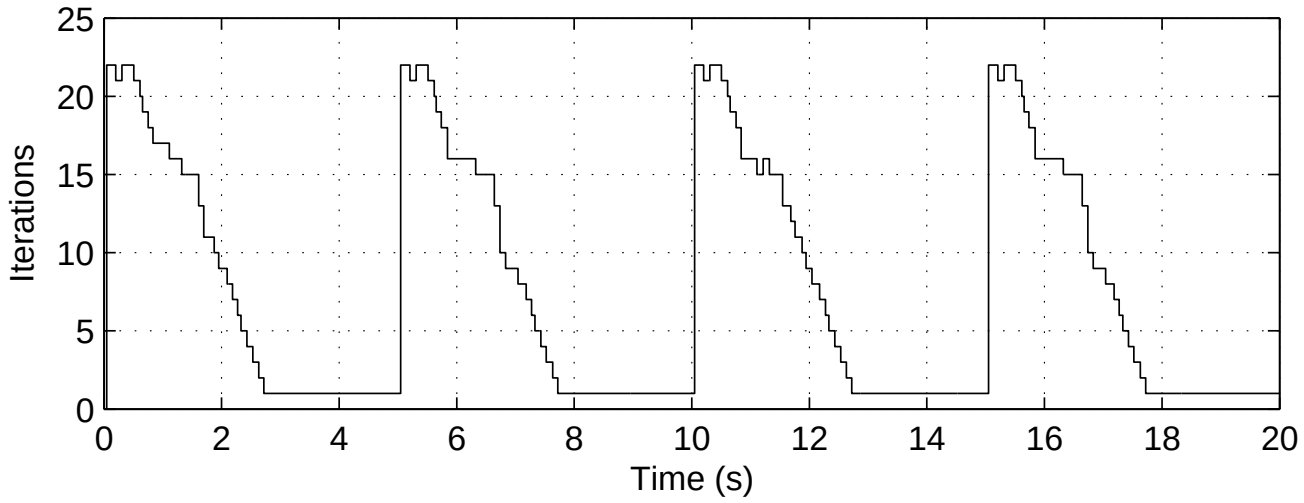
The formulation leads to a quadratic programming problem with linear inequality constraints

$$\begin{aligned} \text{minimize} \quad & \Delta\mathcal{U}^T(k)\mathcal{H}\,\Delta\mathcal{U}(k) - \Delta\mathcal{U}^T(k)\mathcal{G}(k) + \mathcal{C} \\ \text{subject to} \quad & \Omega\,\Delta\mathcal{U}(k) \leq \omega(k) \end{aligned}$$

being solved for $\Delta\mathcal{U}(k) = [\ \Delta\hat{u}(k) \ldots \Delta\hat{u}(k + H_u - 1)\ ]$.

Properties

- Convex optimization problem solved each sample
- Highly varying execution times → worst-case pessimistic
- Execution time depends on external factors such as reference signals and disturbances



Premature Termination

- Optimization may be aborted any time after a feasible solution has been obtained
- Based on recent stability results [Scocaert et. al. 1999]
- A solution is feasible if it fulfills the constraints and obtains a lower cost than in the previous sample

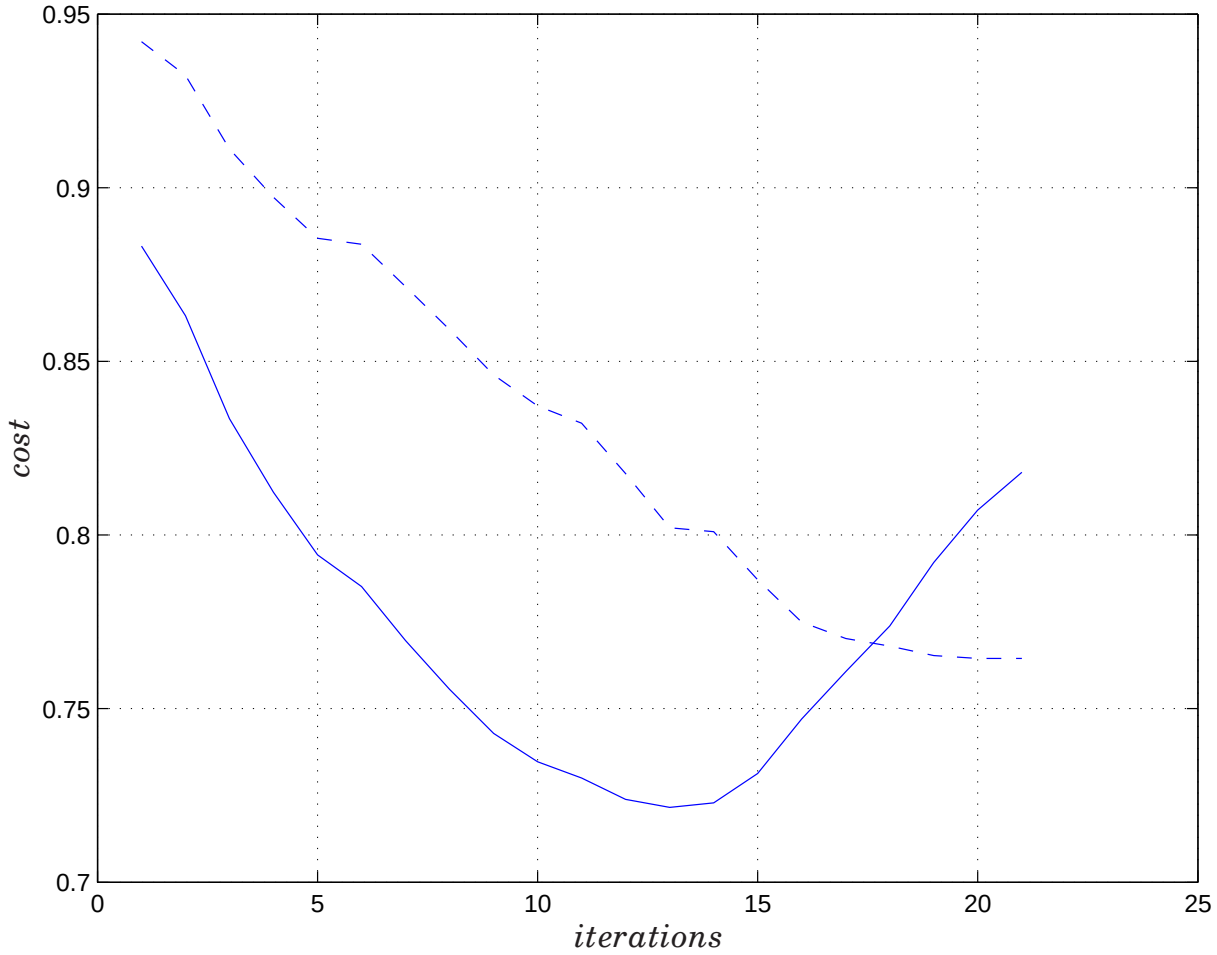
Trade-off Cost vs Delay

- Assuming a constant process delay, $\tau < h$, over the prediction horizon
- Leads to an augmented process model
- The matrices in the cost function are computed as functions of the delay, τ

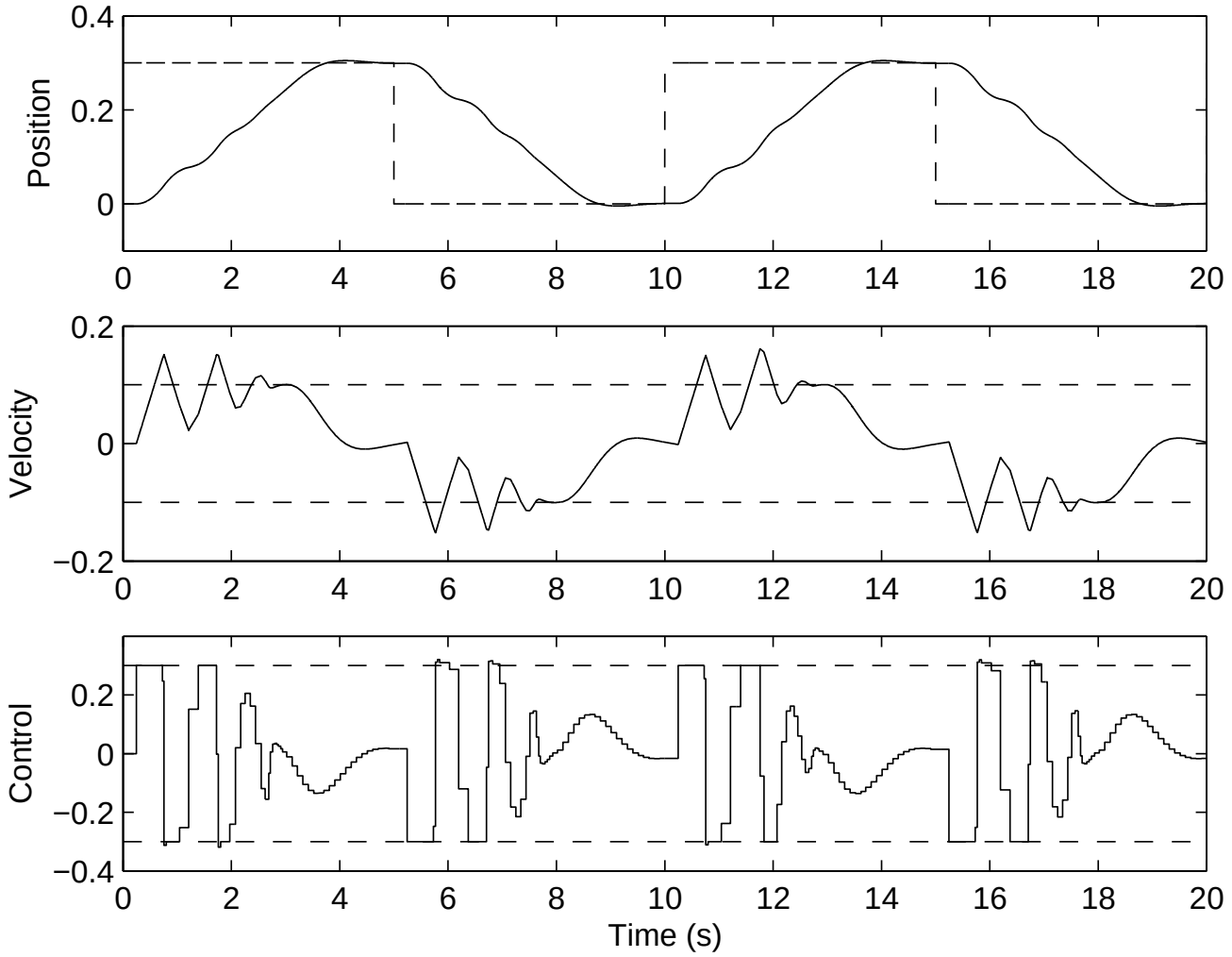
$$J_d(\Delta\mathcal{U}_i, \tau) = \Delta\mathcal{U}_i^T \mathcal{H}(\tau)\Delta\mathcal{U}_i - \Delta\mathcal{U}_i^T \mathcal{G}(\tau) + \mathcal{C}(\tau)$$

- The optimization algorithm is terminated based on this delay-dependent cost

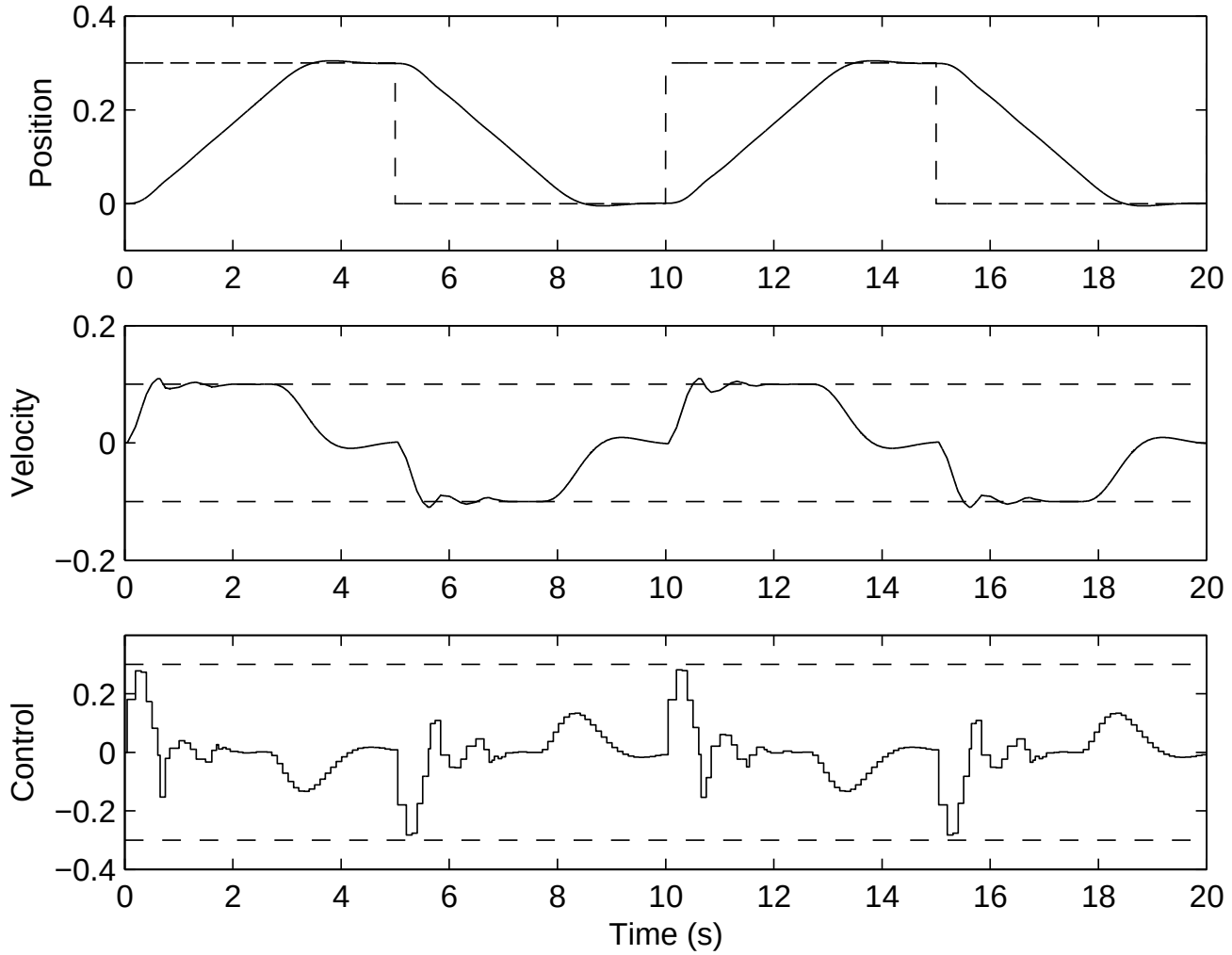
Trade-off Cost vs Delay



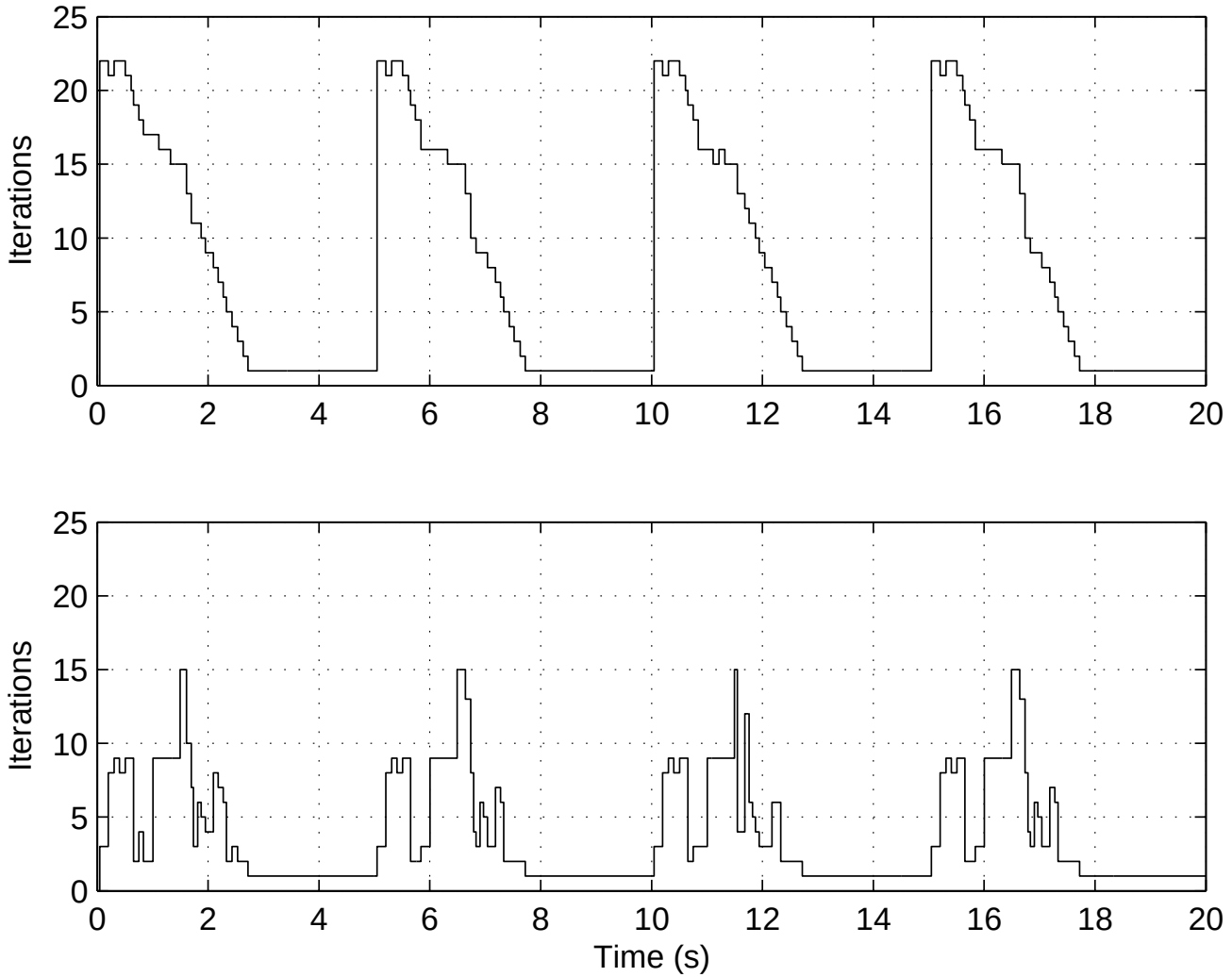
Full Optimization



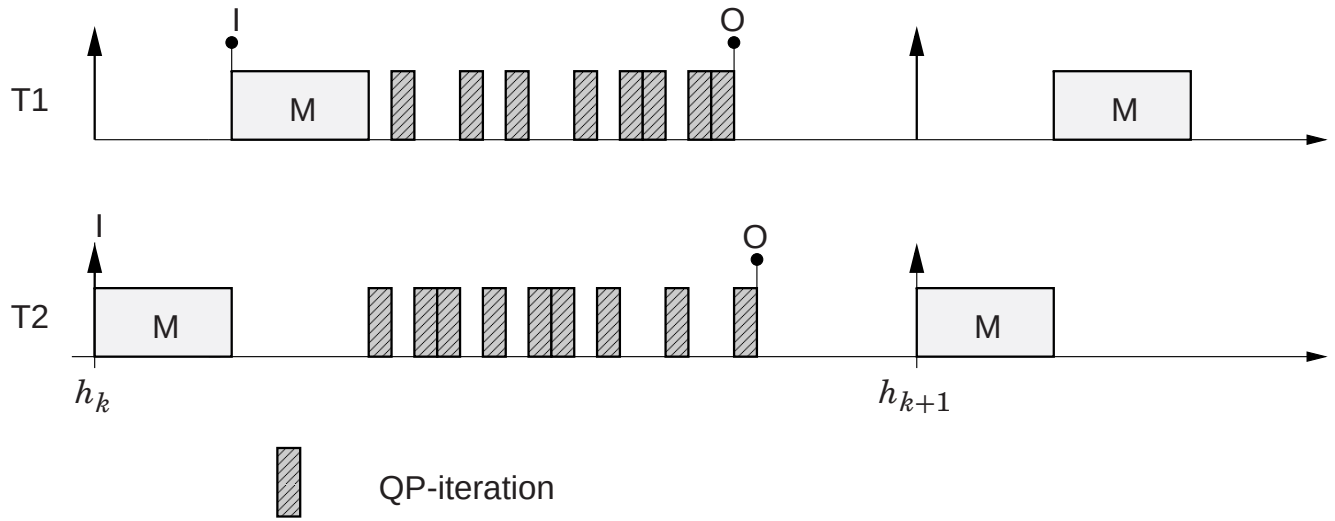
Exploiting Trade-off



Iteration Comparison

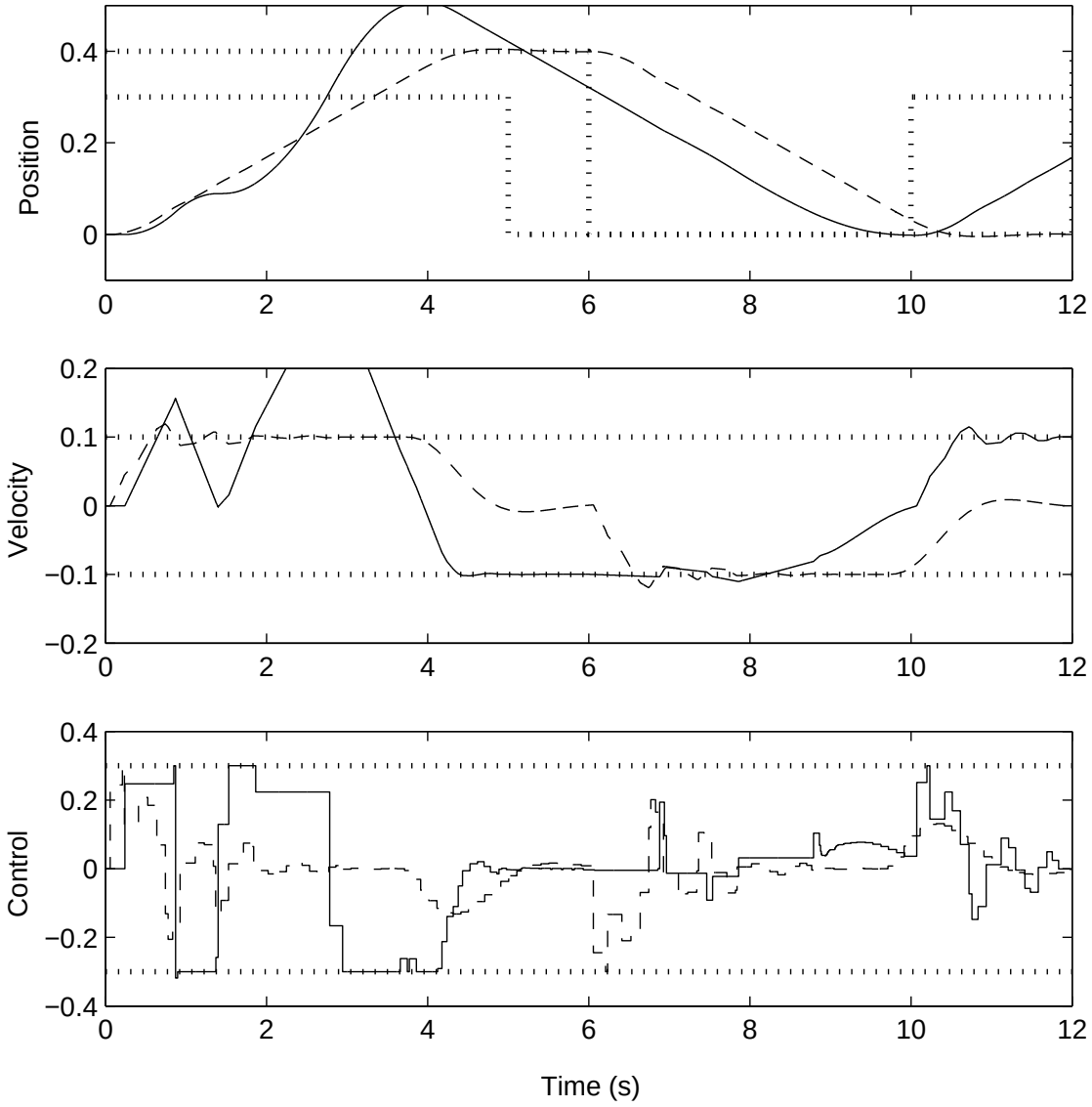


Dynamic Scheduling of MPCs

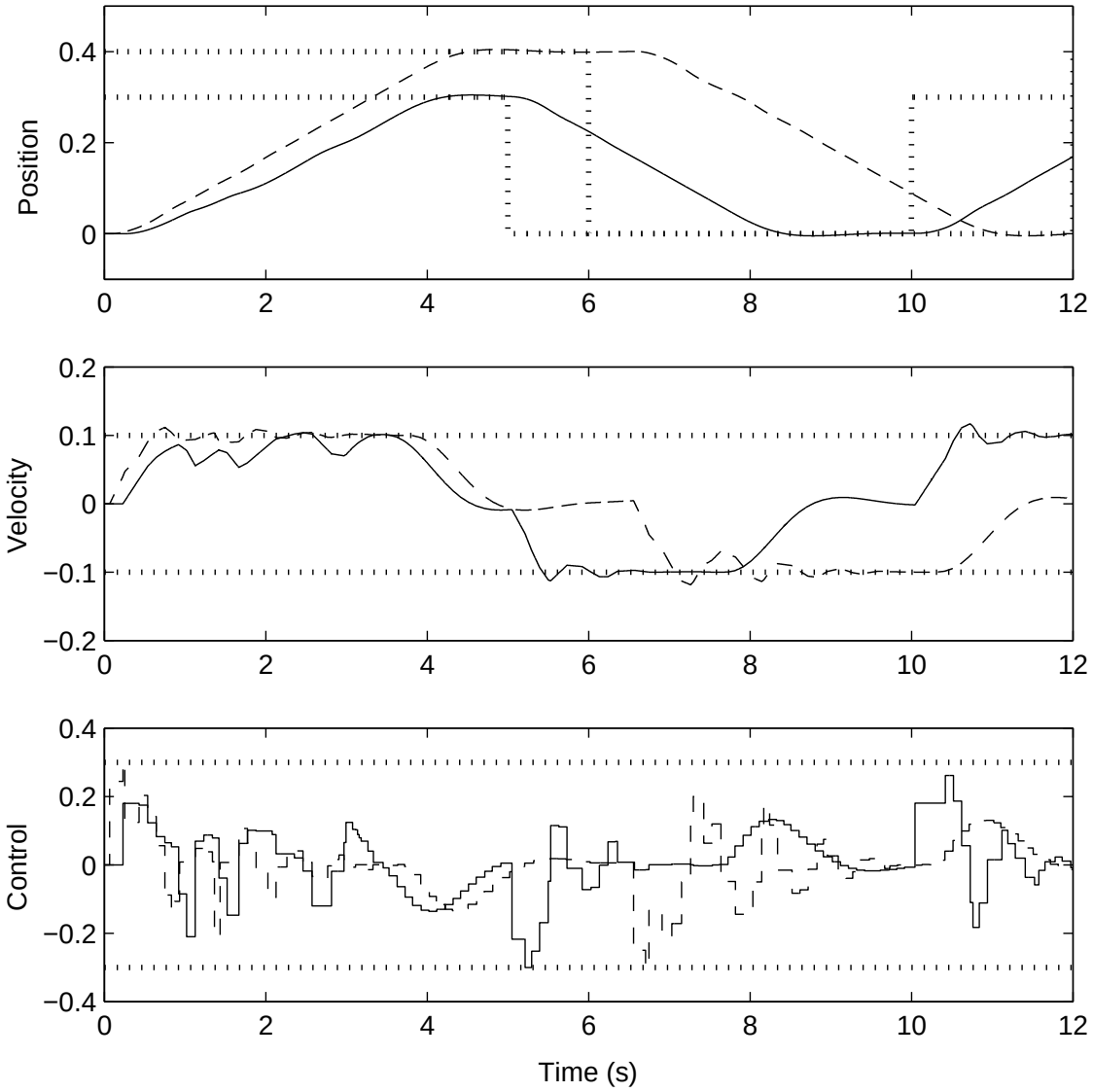


- Mandatory part consists of finding a feasible solution
- Remaining QP-iterations scheduled using the cost functions as dynamic priorities
- Reflects the relative importance of the tasks

Fixed-priority Scheduling



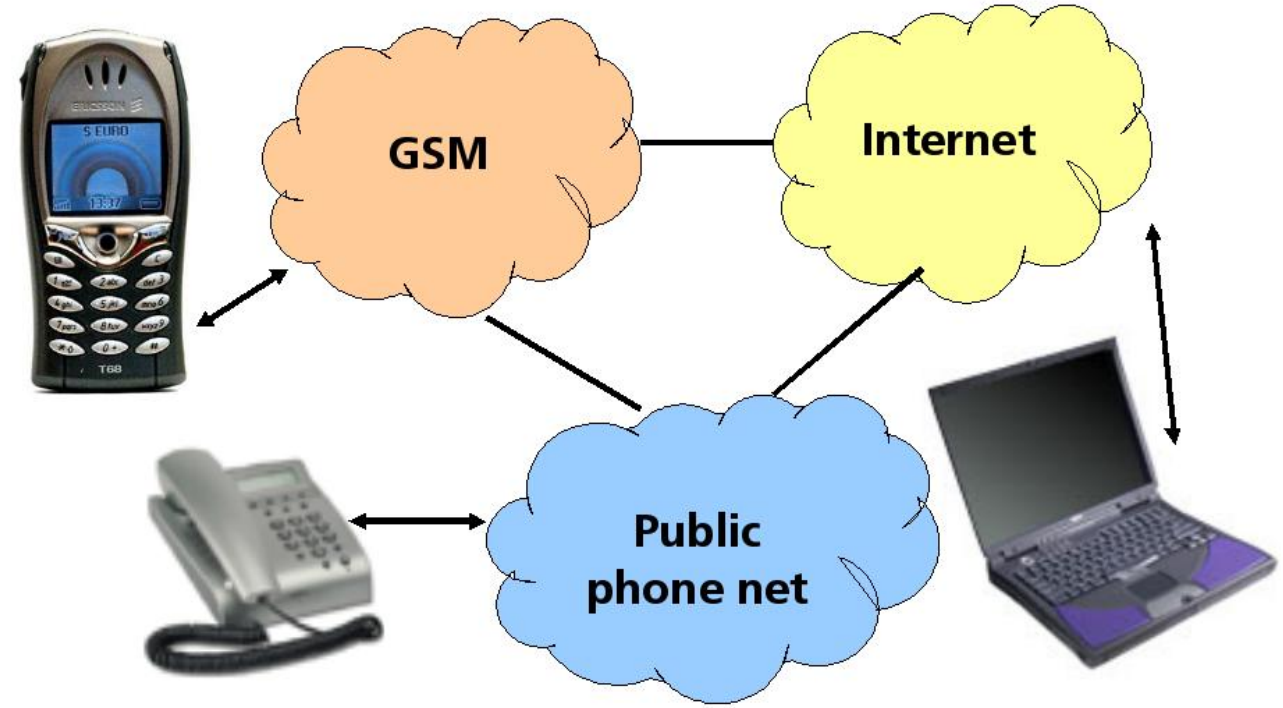
Feedback Scheduling

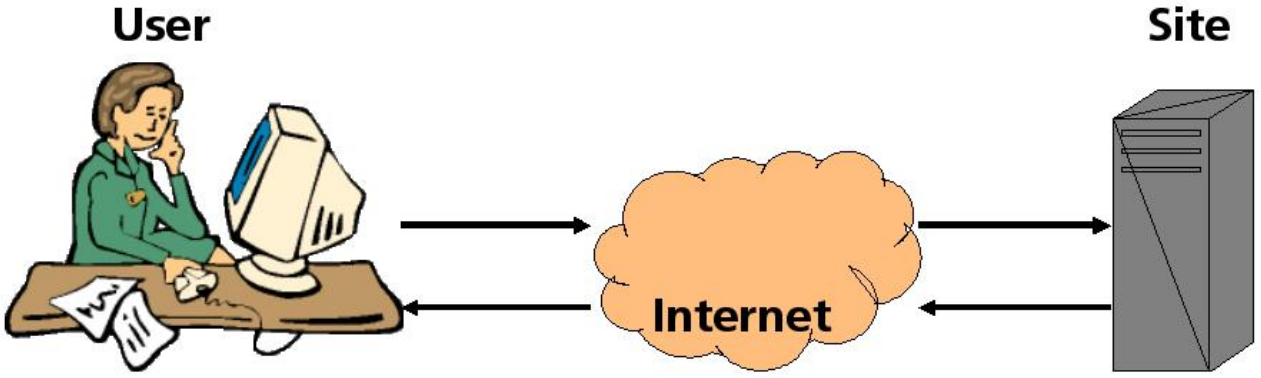


Outline

- Overview
- General Observations
- Nice Control of Linux
- Feedback Scheduling of Control Tasks
  - Infinite Horizon
  - Finite Horizon
  - MPC
- **Control of Web servers**

Control at Different Levels



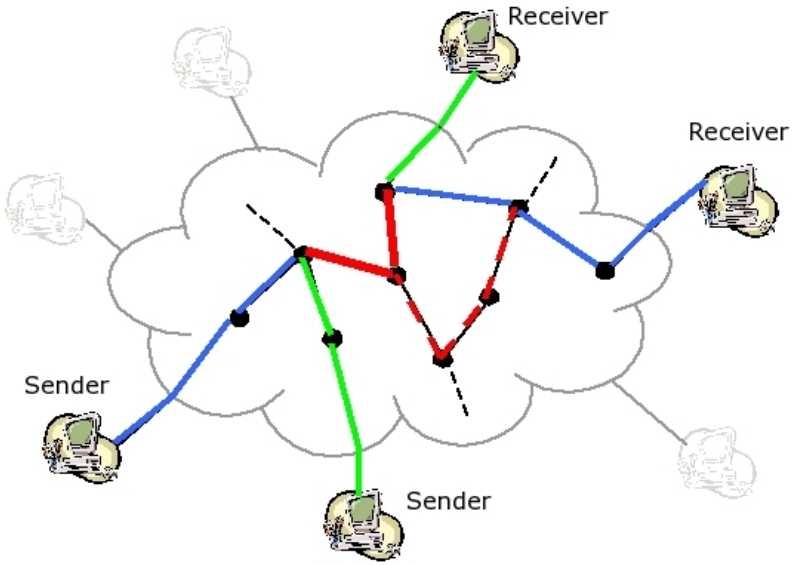


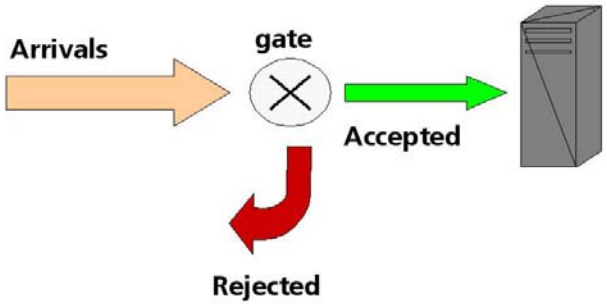
Controlling Computer Systems

- Feedback control is embedded in the TCP protocol in the form of a sliding window mechanism.
- Introduced in the 70's to solve the congestive failure problems that had brought down the network.
- We have not experienced system-wide congestive failures again even though the network has grown orders of magnitude.
- This is a testament of the effectiveness of feedback control in a highly dynamic, decentralized, and fast changing environment.
- Can feedback control be applied to accurately control the performance of web server systems?

What to Control?

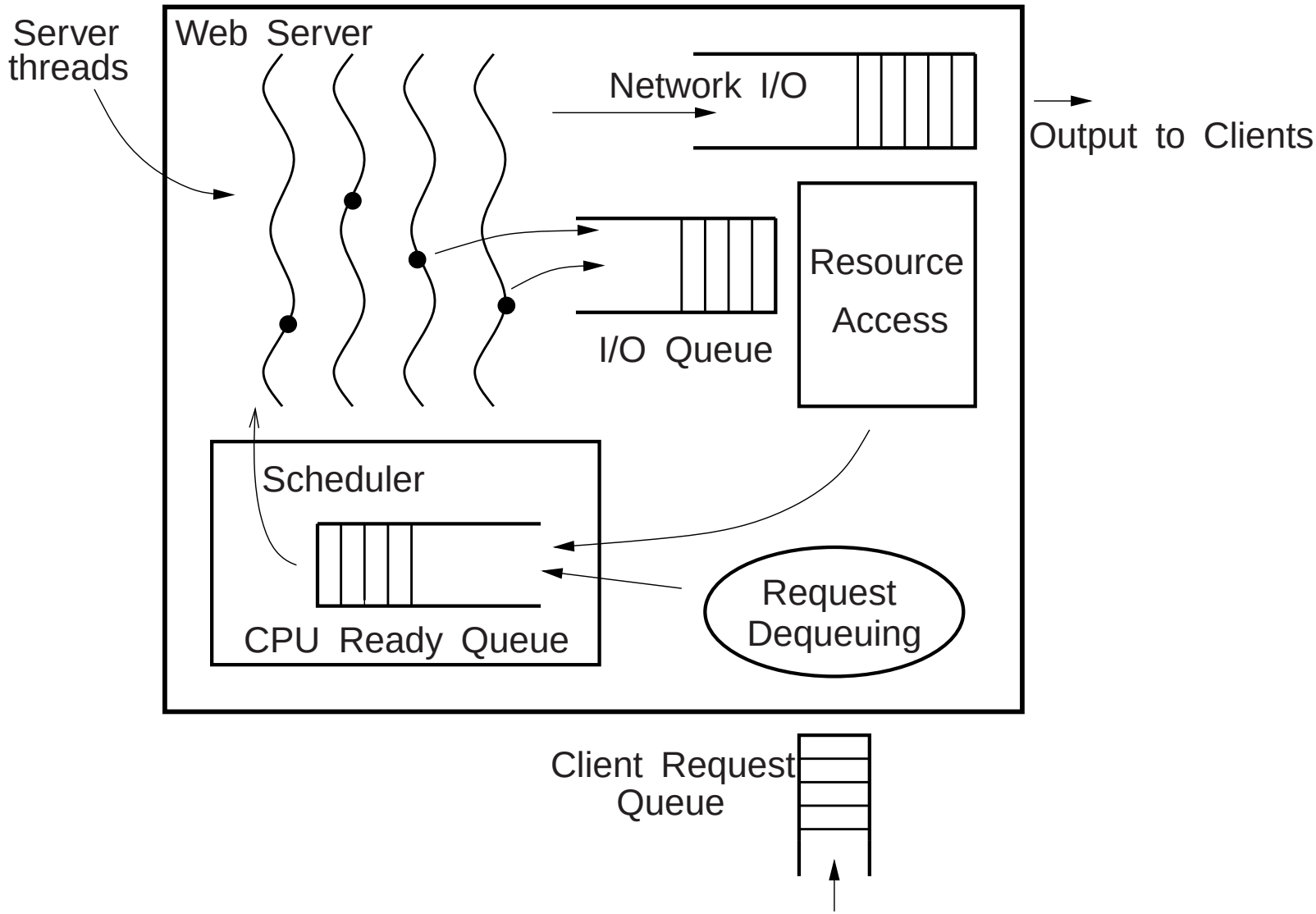
- Temporal
  - local (at server)
  - global (End-to-End/TCP)
- Spatial (routing)





We will focus on temporal control issues at the server.

Web Service Performance Control



Difficulties

- Web server systems are stochastic with highly non-linear behavior.
  - Response times increase exponentially with utilization at heavy load.
  - Input and output saturations.
- The parameters of the stochastic process, e.g. arrival rate, can change abruptly without warning.
- How should the server system be modeled?
- What is the control objective?
- How can we influence the system, i.e., which actuators are available?

Web Server Modeling

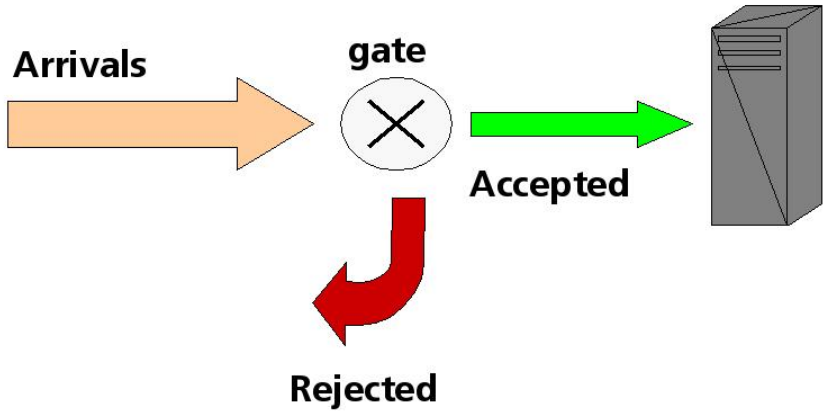
- Queuing theory models:
  - Discrete-event models
  - Markov chains
- Control theory models:
  - Non-linear flow models (continuous time)
  - Discrete-time models
- Differential (or difference) equation models traditionally used in control theory have their limitations.
- Works well in the case of heavy workload when the web server can be modeled using fluid approximations.

Control Objective

- The main objective is to control the service delay of individual requests.
- Can be controlled directly or indirectly by manipulating the server queue lengths.
- The stochastic nature of the system requires averaging (inherent in the non-linear flow model).
- Want to be able to control both long-term averages and transient responses.

Actuator Mechanisms

- The difference between the service rate, μ, and the arrival rate, λ, determines the delay experienced by the requests.
- Changing the arrival rate, admission control:



- Changing the service rate:
  - Number of server threads
  - Quality adaptation
  - Dynamic voltage scaling

Absolute Delay Control

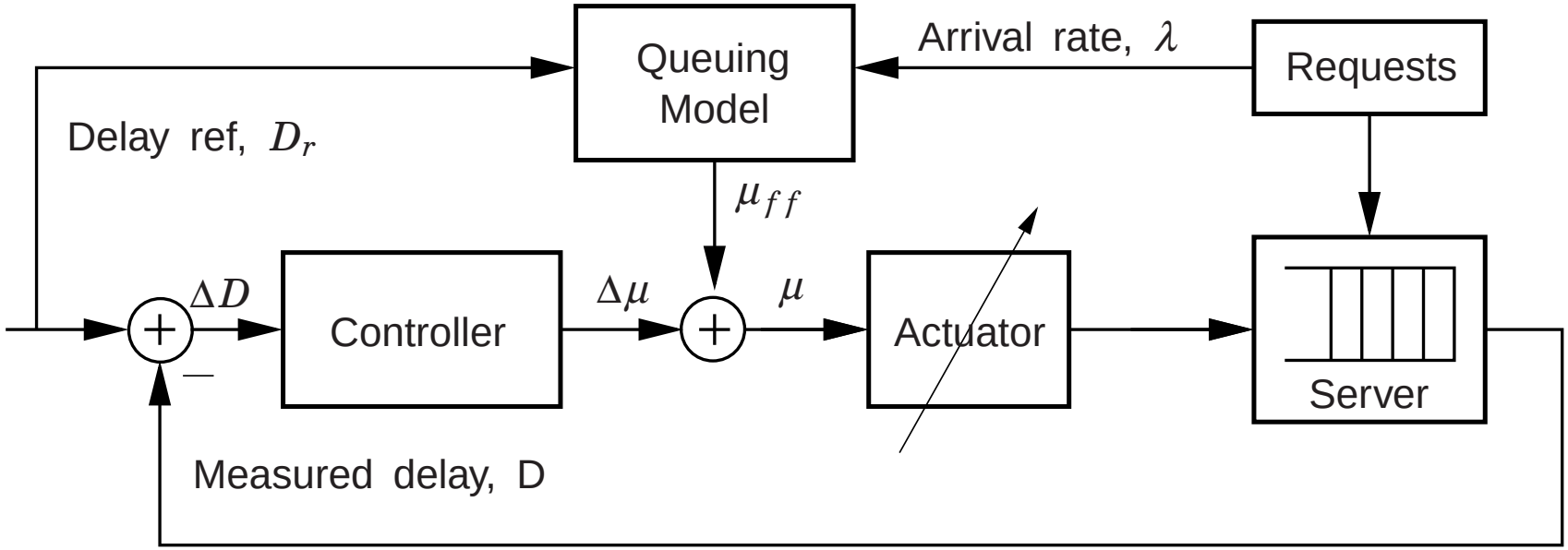
Queuing Model Based Absolute Delay Control

- L. Sha, X. Liu, UIUC and Y. Lu, T. Abdelzaher, UVa

Control Objective

- Want to keep the average timing delay experienced by users close to a desired value, D_r.
- The delay specification, D_r, relates to the QoS agreement with the end user.
- Delays consistently longer than the specification are unacceptable to the users,
- and delays consistently shorter than the specification indicate over-provisioning of resources.

Absolute Delay Control



Key Ideas

- Use queuing theory to model the non-linear behavior of the web server.
- Use the steady-state solution of the queuing model as feed-forward control to bring the system to an equilibrium point near the desired delay set-point.
- Example: M/M/1 queuing model where $\hat{D} = \frac{1}{\mu - \lambda}$. Use feed-forward control, $\mu_{ff} = \frac{1}{D_r} + \lambda$.
- Use linear feedback control to suppress approximation errors and transient errors around the operating point.

Problems

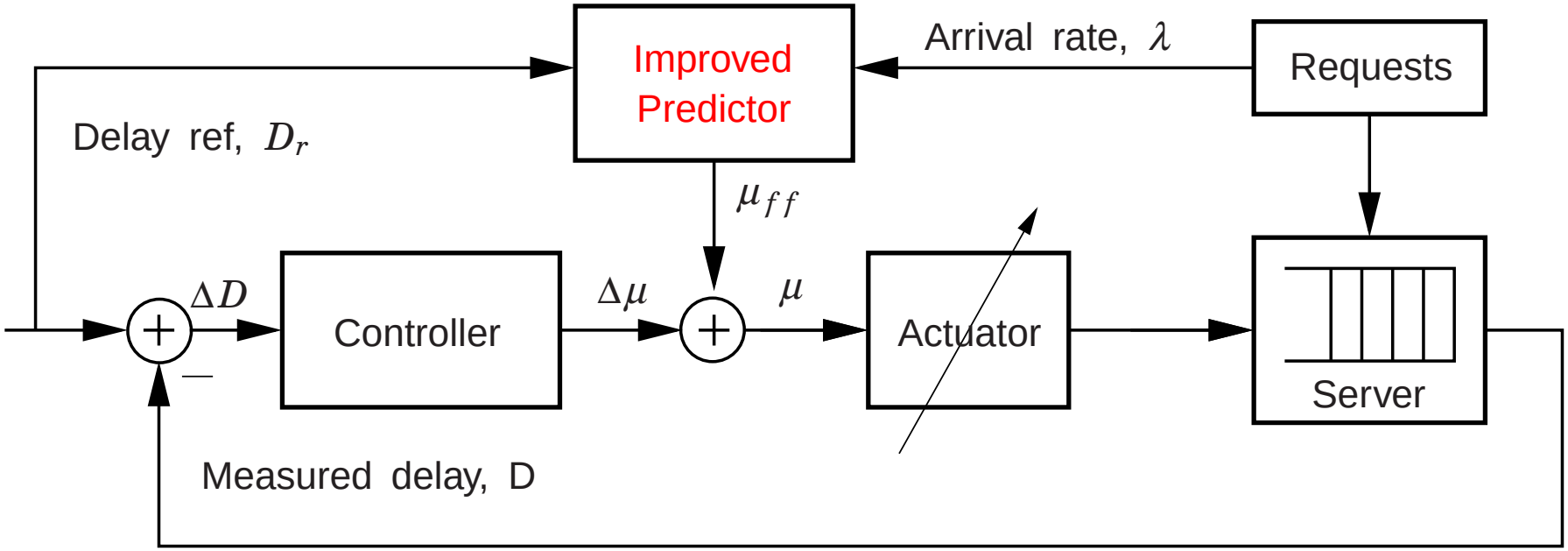
- Queuing theory predicts delay as a function of arrival and service rates.
- The prediction applies only to long-term averages.
- Insensitive to sudden load changes and does not handle transient responses very well.
- Internet load is very bursty and may change abruptly in a frequent manner.
- Inaccurate assumptions in the queuing model, e.g., Poisson distributed arrival and service processes.

Improved Feed-forward Prediction

Improved Feed-forward Prediction

- Y. Lu, T. Abdelzaher, UVa and D. Henriksson, LTH

Improved Feed-forward Predictor



- Based on instantaneous measurements instead of long-term averages.

Notation

$\hat{C}$ = average number of processor cycles required by a request

μ = server speed

N = number of waiting requests

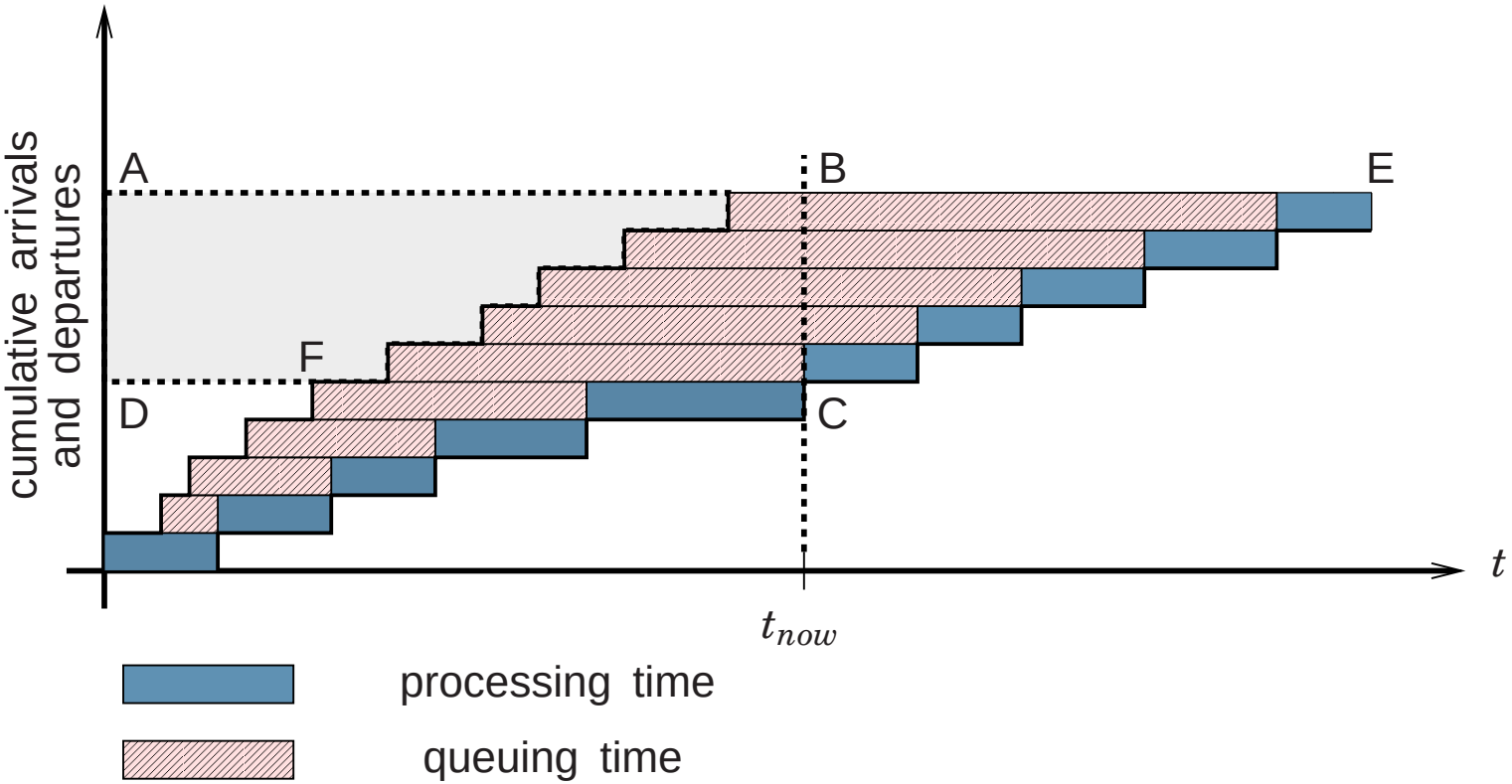
$\hat{D}$ = average delay experience by the N requests

$N\hat{D}$ = total delay experienced by the N requests

$\hat{A}_i = \frac{1}{N}\sum_{k=i}^{i+N-1} A_k$ = the average arrival time

$Q_i = t_{now} - \hat{A}_i$ = average queuing time for the requests being dequeued in the i'th sample

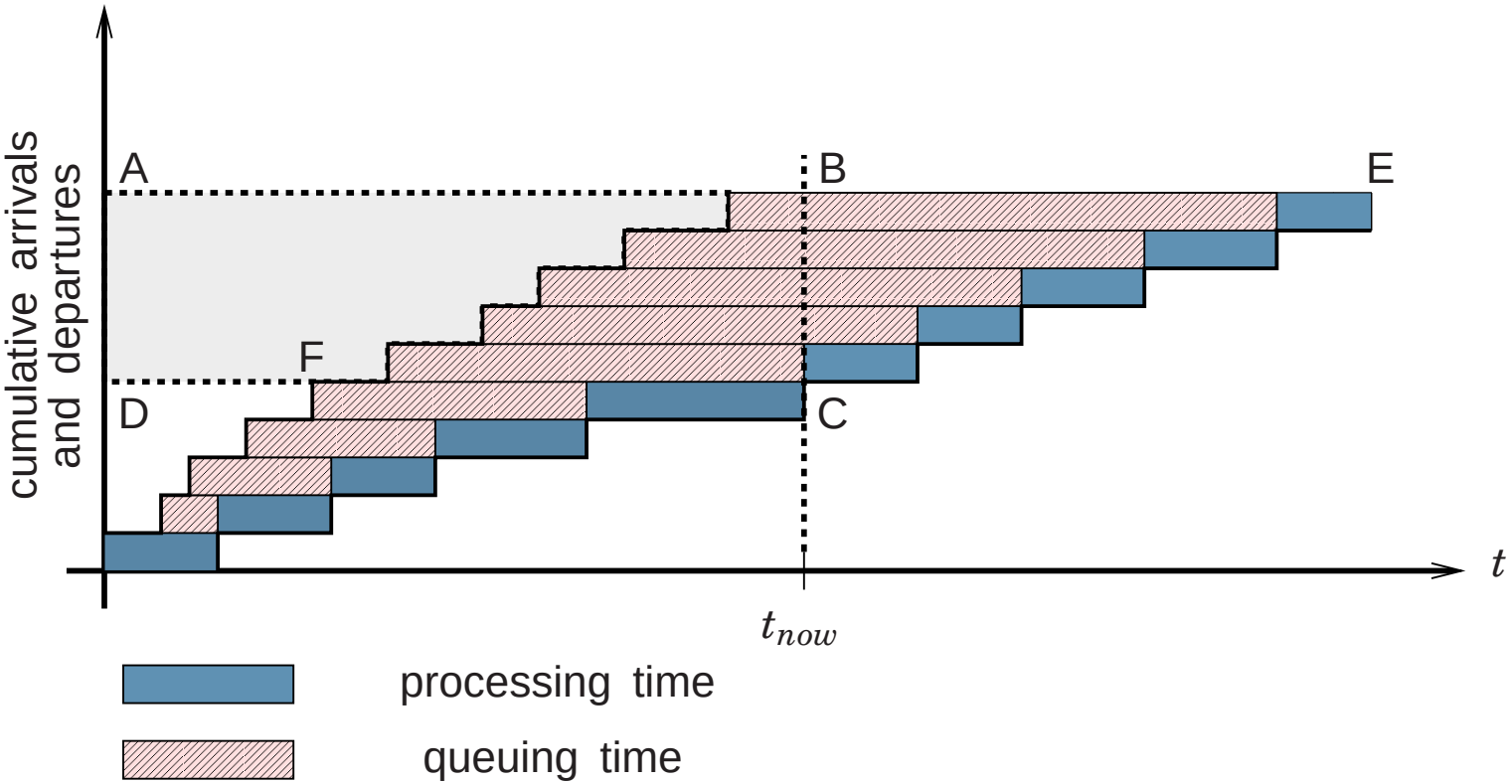
The Predictor



$$BECF = ABCD + BEC - ABFD$$

$$N\hat{D} = N \cdot t_{now} + \frac{N \cdot (N\hat{C}/\mu)}{2} - \sum_{k=i}^{i+N-1} A_k$$

The Predictor



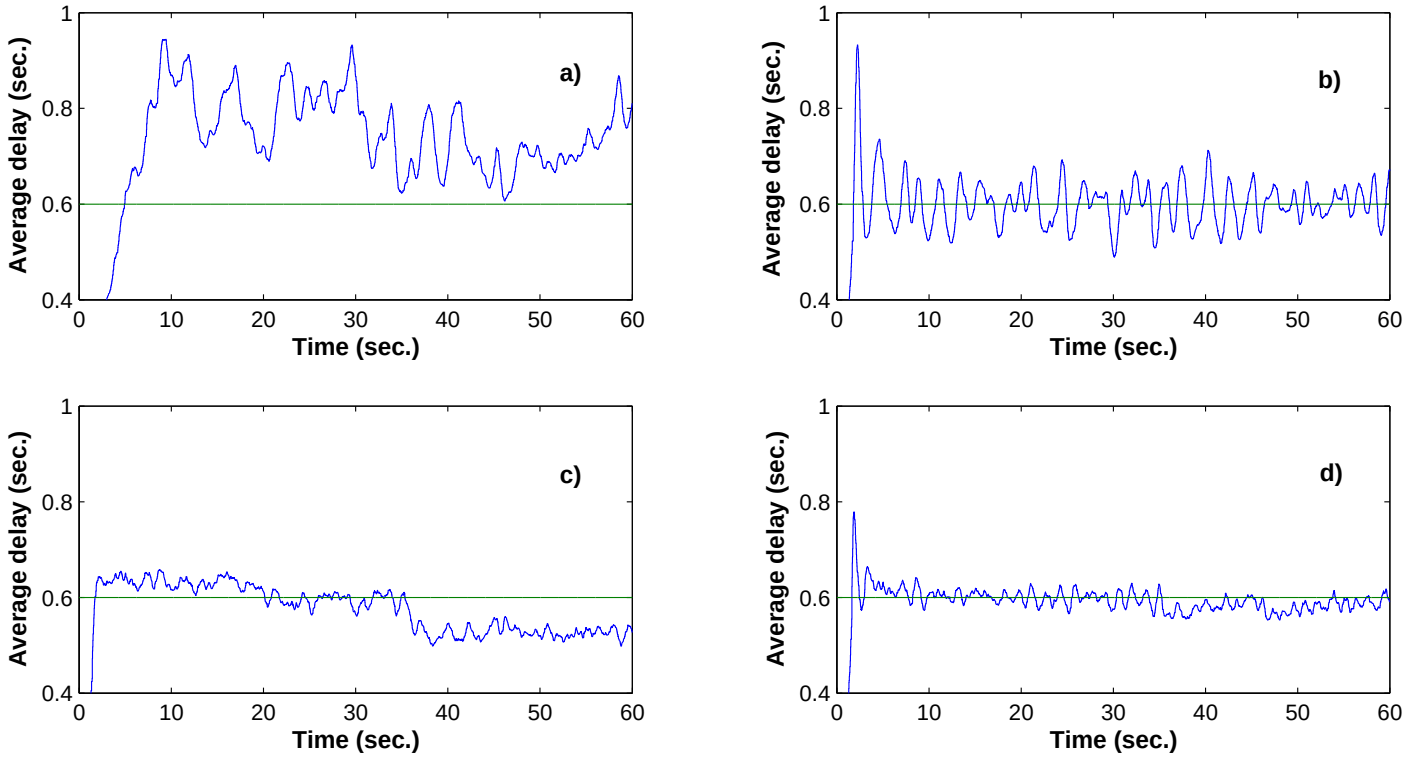
$$\hat{D} = t_{now} - \hat{A} + \frac{N\hat{C}}{2\mu} = \hat{Q} + \frac{N\hat{C}}{2\mu}$$

$$\mu_{ff} = \frac{N\hat{C}}{2(D_r - \hat{Q})}$$

The Feedback Controller

- Event-triggered PI-controller with sliding window action.
- Need a long observation window, N_{obs}, to accurately estimate the average values of arrival rates and processing times of requests.
- Long observation window does not imply slow control action. Control updated every $N < N_{obs}$ event (request departure).
- Quick update steps reduce the variance and control efforts in each sample.
- The PI-controller is implemented using gain-scheduling
  - tuned for different operating points (arrival rate and delay set-point, D_r).
- Anti-windup crucial.

Simulations

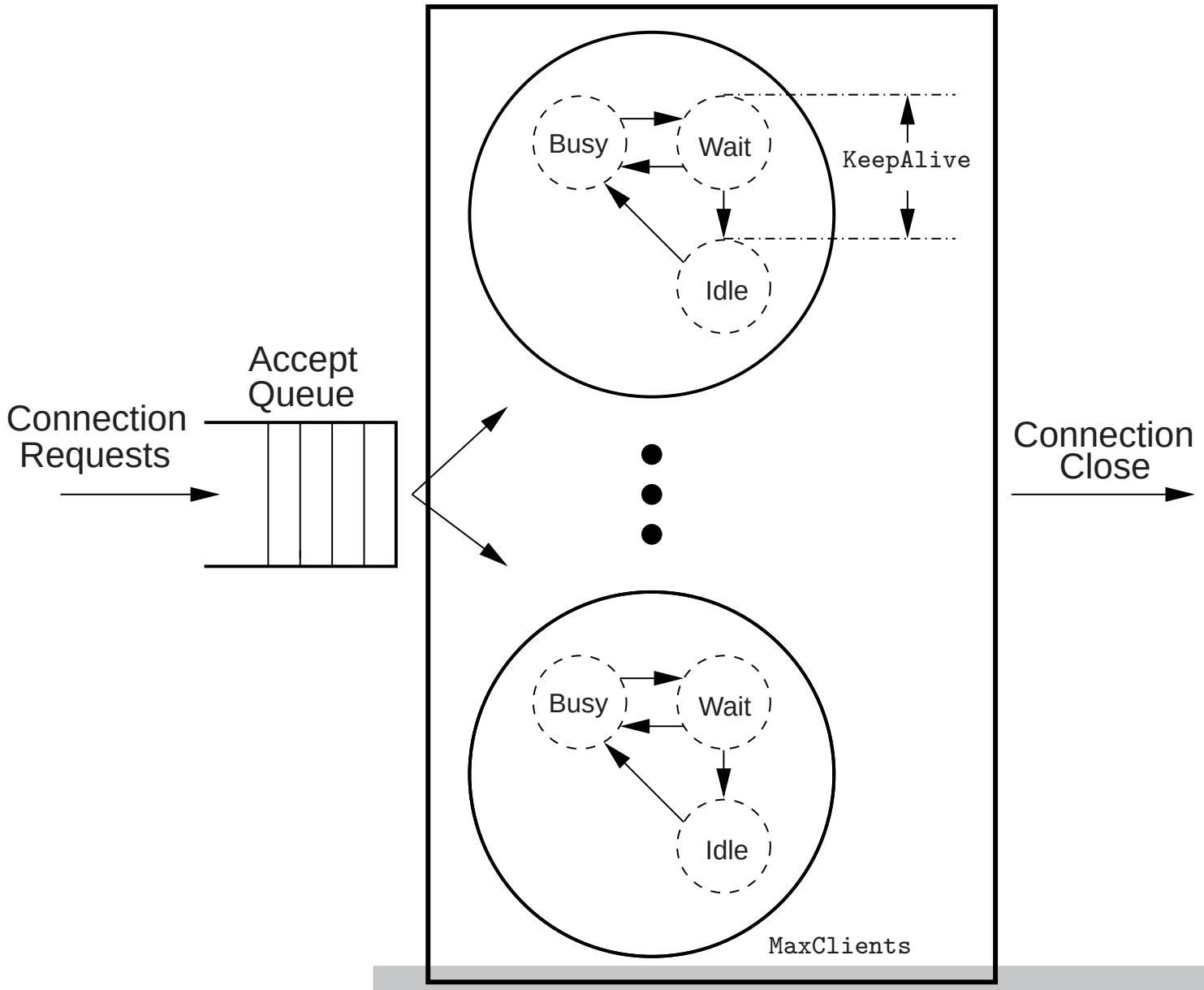


- Performed with TrueTime
- a: M/M/1, b: M/M/1 + PI, c: Predictor d: Predictor + PI

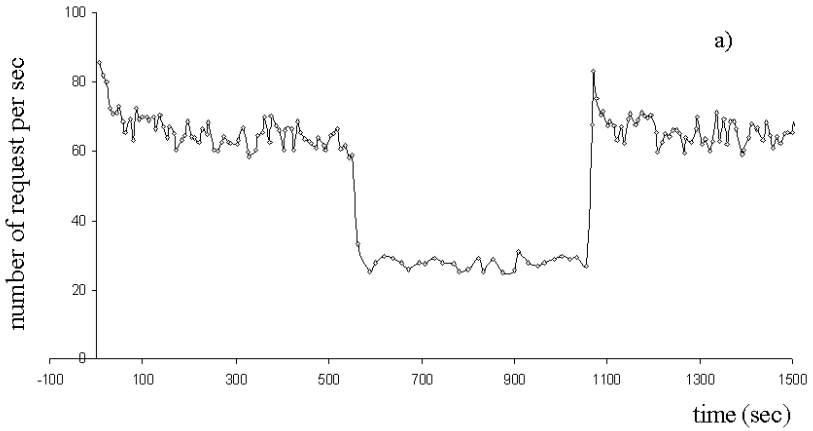
Experiments

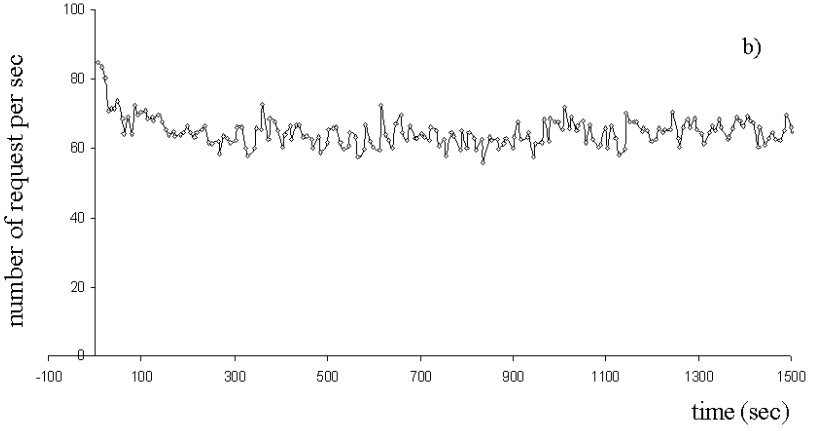
- Performed at an Apache web server test-bed at the University of Virginia.
  - Sensor: average response time of incoming requests
  - Actuator: number of server processes
- Load generation: Scalable URL Reference Generator (SURGE). `http://www.cs.bu.edu/faculty/crovella/links.html`
- Platform: Linux-based PC cluster on 100 Mbit Ethernet.
- 4 machines of which one ran the server with HTTP 1.1, and the rest ran clients to stress the server.

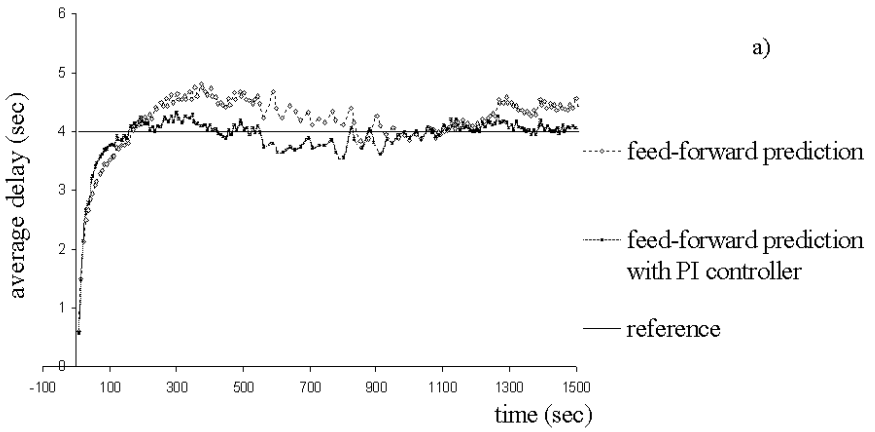
Apache Web Server Session Flow

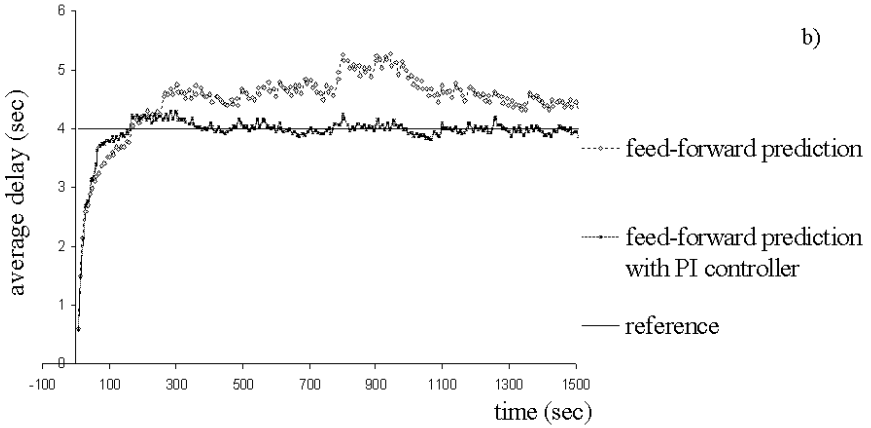


Results


a)
number of request per sec
time (sec)


b)
number of request per sec
time (sec)


a)
average delay (sec)
time (sec)
feed-forward prediction
feed-forward prediction with PI controller
reference


b)
average delay (sec)
time (sec)
feed-forward prediction
feed-forward prediction with PI controller
reference