Conclusions of the ARTIST2 Roadmap on Control of Computing Systems

Karl-Erik Årzén[*], Anders Robertsson, Dan Henriksson
Dept of Automatic Control, LTH
Lund University, Box 118, SE-221 00 Lund, Sweden

Mikael Johansson, Håkan Hjalmarsson, Karl Henrik Johansson
Dept of Signals, Sensors and Systems
Royal Institute of Technology, SE-100 44 Stockholm, Sweden

1. Background

The use of control-based methods for resource management in real-time computing and communication systems has gained a substantial interest recently. Applications areas include performance control of web-servers, dynamic resource management in embedded systems, traffic control in communication networks, transaction management in database servers, error control in software systems, and autonomic computing. Within the European EU/IST FP6 Network of Exellence ARTIST2 on Embedded System Design a roadmap on Control of Real-Time Computing Systems has recently been completed. The focus of the roadmap is how flexibility, adaptivity, performance and robustness can be achieved in a real-time computing or communication system through the use of control theory. The item that is controlled is in most cases the allocation of computing and communication resources, e.g., the distribution or scheduling of CPU time among different competing tasks, jobs, requests, or transactions, or the communication resources in a network. Due to this, control of computing systems also goes under the name of *feedback scheduling*.

The roadmap is divided into six research areas: control of server systems, control of CPU resources, feedback scheduling of control systems, control of communication networks, error control of software systems, and control middleware. For each area an overview is given and challenges for future research are stated. The aim of this paper is to summarize the conclusions concerning these research challenges. A preliminary version of the complete roadmap can be found on `http://www.control.lth.se/user/karlerik/roadmap1.pdf`

1.1 Motivation

Feedback-based approaches have always been used in engineering systems. One example is the flow and congestion control mechanisms in the TCP transport protocol. Typical of many applications of this type is that feedback control is used in a more or less *ad hoc* way without any connections to control theory. During the last 5-10 years this situation has changed. Today control theory is beginning to be applied to real-time computing system in a more structured way. Dynamic models are used to describe how the performance or quality of service (QoS) depend on the resources at hand. The models are then analyzed to determine the fundamental performance limitations of the system. Based on the model and the specifications control design is performed. In some cases the analysis and design are based on optimization. A recent textbook, [Hellerstein *et al.*, 2004], addresses the question of how to introduce control theory for computer science students.

The publications in the area are rapidly increasing, see e.g., [Hellerstein *et al.*, 2005] and the references therein. However, so far most of the work presented in literature have been conducted by scientists working *either* in the real-time computing or telecommunication fields, *or* in the automatic control field. Unfortunately, this has sometimes led to erroneous models and strange results. In order to achieve good results a multi-disciplinary approach is necessary.

1.2 Modeling

Control of computing systems introduces new types of problems that are not present when controlling physical plants. A main problem is the lack of first principles models. When controlling a physical plant the laws of nature decide to a large degree the behaviour of the plant and can be used to derive dynamical models. A computing system, on the other hand, is a man-made artifact whose internal behaviour is not governed by any laws of nature, at least not on the macroscopic level. This means that it is, generally, not possible to derive any first principles models. One exception, where theoretical models are available is queuing theory [Kleinrock, 1975]. These models have also been used with some success in the design of computing-system controllers. A drawback with queuing models is that they in most cases only hold in the average case and that they assume certain statistical properties, e.g., Poisson traffic.

Computing systems are discrete-event dynamic systems (DEDS). This makes it natural to use a timed discrete-event modeling formalism, such as timed automata or timed Petri nets. This is, however, in many cases too fine-grained and easily leads to state-space explosion. Another issue is the types of problems that these formalisms typically lend

[1]Corresponding author. Email: karlerik@control.lth.se

themselves to. Automata-based formalisms are well-suited for expressing and analyzing safety properties and blocking properties. These properties are, however, not the main objectives of performance control. Instead, issues such as stability, performance, and robustness are the main objectives. For these types of problems a time-driven approach is more natural. However, the lack of first principles knowledge necessitates a system identification-based approach, in which, e.g., a discrete-time difference equation model is derived from measured input and outputs. The models and controllers derived in this way are based on periodic sampling. Although periodic controllers are, to a large extent, the approach that is mostly used in applications, it is from many respects more natural to invoke the controller in an event-driven fashion. For example, in a queue-length control problem it makes more sense to calculate a new control action when a request is queued or dequeued, or every nth enqueue/dequeue event, rather than periodically in time. A problem with aperiodic or event-based systems and aperiodic control of this type, though, is the lack of theory and tools for analysis and design.

2. Control of Server Systems

More and more business and services rely on Internet and server technology. Queue management is important in all servers, e.g., web servers. Server requests are stored in an input queue, the server or worker thread servicing the requests are stored in the ready queue or in different waiting queues, e.g., in order to access memory. Many aspects of the real-time performance of server systems can be inferred from the behaviour of queues.

In a queue it is the difference between the service rate and the arrival rate that determines the delay experienced by the requests. Two types of actuators can be used. An *enqueue* actuator influences the arrival rate of the queue. One example of this is *admission control.* A *dequeue* actuator instead influences the service rate of the requests. Examples of this type of actuator mechanisms are different forms of quality adaptation.

A queue can be modelled in various ways. Using queuing theory, several types of models can be developed. One example is Tipper's nonlinear flow model [Tipper and Sundareshan, 1990]. At a high level, a queue can be seen as an integrator. This can be modeled using, e.g., a difference equation and then analyzed with control theory. Both Tipper's model and integrator-based models can be used as the basis for control design, e.g., [Robertsson *et al.*, 2003].

Flow models of queuing systems approximate the steady-state behaviour of the queue and are typically more accurate the higher the load is on the server. However, for small and medium loads these types of models are less appropriate. An open question is how to combine queuing models with control-theoretic methods. A common

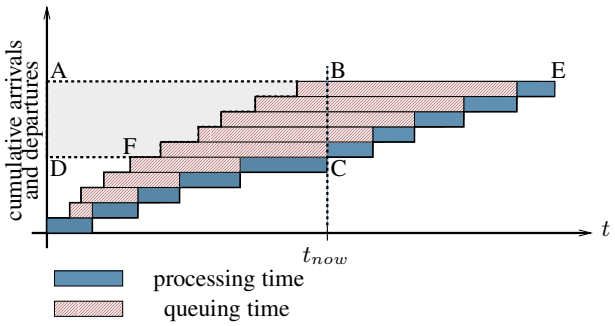

Figure 1 Server queuing and processing delay over time.

approach in delay control is to use nonlinear models from queuing theory for feedforward combined with simple feedback control of PID type, e.g., [Sha *et al.*, 2002]. The aim of the feedforward path is to provide fast setpoint responses, whereas the role of the feedback controller is to compensate for disturbances and incorrect modeling assumptions. An example of the latter is incorrect assumptions about the stochastic nature of arrivals and departures.

In [Henriksson *et al.*, 2004a] an improved feedforward scheme is presented, that makes no assumptions about the statistical properties of the traffic. Instead, it predicts future delays as a function of instantaneous measurements of the situation in the server queue. This includes current queue length and the arrival times of the queued requests, which are assumed to be recorded for use in the prediction, see Fig. 1. The basic idea with the predictor is to choose the service rate that achieves a desired average delay of the requests in the system taking into account their average queuing delay up until the current time. By continuously updating the predictor as requests enter and leave the queue, sudden variations are taken care of more rapidly than using the queuing-theoretic models. A similar approach can also be applied to admission control.

Much work has also been performed on multi-class queuing systems using priority queues. Here it is the ratio between the average delays of adjacent service classes that is subject to control. In [Liu *et al.*, 2006] recent work on multi-tiered web applications using adaptive control is reported.

2.1 Research Challenges

The main challenge in control of servers and software systems in general is to derive a unified theory and framework for performance control of queuing systems that combine elements from control theory and queuing theory and allow an integration of both time-driven liquid model formalisms and event-driven formalisms. Modeling plays a major role here. Which is the right or optimal abstraction level for this type of control problem is still a question with no clear answer. Models at different levels and types need to be com-

bined. We also need better insight in how one should correctly abstract a real server by a suitable queuing model. It is further desirable to combine time-based models with event-based discrete models. Better understanding is needed for which models types that are best suited for a particular application. It is also possible that new models types must be derived for this type of problems.

The challenges for control are connected to the modelling challenges. How do we develop a control theory based on this type of models? The combination of time-driven control design with event-driven implementation is one major issue. In control in general and process control in particular, the characteristics of different types of control loops and control problems are well known and even in some cases formally categorized. Similarly a number of well-defined controller structures exist, e.g., cascade control and ratio control. The same type of classification is necessary also in control of computer systems. One possibility is to make use of ideas from design patterns to create well-defined patterns for server control problems.

Large eCommerce servers are multi-tier systems consisting of web server front-ends, business logic in the intermediate layers, and database servers as back-ends. The overall system is a MIMO system where control is needed at several layers. Model-based predictive control (MPC) is an interesting possibility here. MPC also explicitly handles constraints on control signals and state variables, which is common in queueing problems, e.g., buffer size limits. In [Xu *et al.*, 2006] different predictive control algorithms have been compared and applied to resource allocation.

Our current notion of dynamics is based on the behaviour of physical systems, e.g., mechanical systems. It is not necessarily so that this type of dynamics also suits software systems. The same holds for stability. It is not completely clear what an unstable software system really means or what type of stability definitions that makes sense. Related to this is the question of how we design or program software systems in such a way that they are observable and controllable. Which types of sensor and actuators makes most sense for this type of systems.

In order to make control of server systems applicable on a wider industrial scale it is necessary to have built-in support for this in operating systems and/or middleware. On which level this should be handled is not clear. Should there be a special POSIX/Control standard defined?

3. Control of CPU Resources

Feedback scheduling of CPU resources is an area where fairly much research has been performed, especially for embedded real-time systems. In feedback scheduling the allocation of CPU resources is based on a comparison of the actual resource consumption by, e.g., a set of tasks, with the desired resource consumption. The difference, or control error, is then used for deciding how the resources should be allocated to the different users. The resources can include CPU-time, memory, I/O bandwidth etc. Dynamic feedback-based schemes are already part of several commodity operating systems such as Linux and Solaris, but they have typically not been designed by applying control theory.

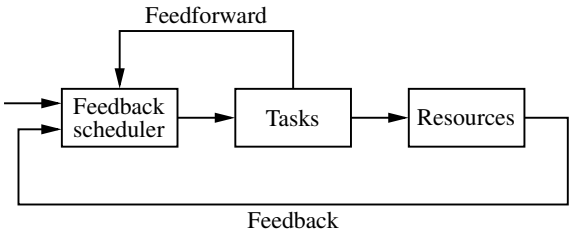

Figure 2 A general feedback scheduling system. The scheduler adjusts the tasks' demands based on feedback from the current use of critical resources. The tasks may also inform the scheduler that they are about to consume more resources (feedforward).

Feedback scheduling is primarily suited for applications with soft or adaptive real-time requirements. This includes different types of multimedia applications, but also a large class of control applications. Feedback scheduling of CPU resources has strong relationships with the queue control employed for server systems and many of the results in one area can be directly applied in the other area.

An early result is given in [Stankovic *et al.*, 1999] where the Feedback Control EDF scheduling algorithm is presented. A PID controller is used to regulate the deadline miss-ratio for a set of soft real-time tasks with varying execution times, by adjusting their CPU time utilization. The approach has later been extended with an additional PID controller that controls the CPU utilization.

Many scheduling techniques that allow QoS adaptation have been developed. An interesting mechanism for workload adjustments is given in [Buttazzo *et al.*, 1998], where an elastic task model for periodic tasks is presented. A large amount of feedback-based or adaptive global QoS management systems have also been proposed. Some examples are [Chu and Nahrstedt, 1999; Aparah, 1998]. In [Yuan and Nahrstedt, 2003a] issues of QoS and energy savings are experimentally evaluated using the CPU scheduler GRACE-OS.

Control-based ideas have also been used for dynamic allocation of bandwidth in aperiodic task servers and for dynamic allocation of resource reservations in reservation-based scheduling. The main application area for these techniques is multimedia applications, e.g., streamed audio and video. The idea behind resource reservation is to explicitly control the computing resources assigned to a given activity (job, task, or application). Each activity receives a fraction (reservation), U_i, of the processor capacity and will behave as if it was executing alone on a slower, virtual processor. The motivation for feedback is the need to cope with incor-

rect reservations, to be able to reclaim unused resources and distribute them to more demanding tasks, and to be able to adjust to dynamic changes in resource requirements. Hence, a monitoring mechanism is needed to measure the actual demands and a feedback mechanism is needed to perform the reservation adaptation. Two types of feedback are possible. On a global, system-wide level a QoS controller adjusts the size of the individual reservations given to the different activities based on the measured performance and resource utilization. On a task or activity level, local feedback is employed to adjust the resource requirements of the individual tasks based on the experienced QoS levels and the amount of resources available to the task, as decided by the global QoS controller. The local resource usage can be adjusted through rate adaptation, by executing the task at different service levels, and by job skipping.

3.1 Research Challenges

In addition to several of the challenges for server systems, the following items are important for control of CPU resources. Multiprocessor systems will become common in the near future also for certain embedded applications. So far very little of the control-based methods to CPU resource management have been applied to multiprocessor systems. Power saving is becoming increasingly important in all computer applications, including server systems. Adjusting the CPU speed using, e.g., Dynamic Voltage Scaling (DVS) techniques, is an alternative way of adjusting the service requirements of a task, e.g. [Yuan and Nahrstedt, 2003b]. Minimizing the power consumption is also an important goal in itself for many networked embedded systems, e.g., sensor networks. The joint optimization problem of minimizing energy while still meeting real-time constraints already today receives considerable attention from the research community. However, it is an important area also for the future. Resource management in distributed systems where an activity spans multiple nodes is also an important issue. How do we adapt the resources individually in the different nodes in order to obtain a good global behaviour, e.g., acceptable end-to-end response times?

Hierarchical resource allocation schemes based on dynamic reservations in combination with local feedback control loops for the individual tasks is an interesting and promising approach where more research is needed. How do we enforce the notion of virtual CPUs that execute within a real CPU with, possibly, different scheduling policies, and where the share that each virtual CPU receives of the total CPU resources is dynamically adjusted based on resource requirements and availability?

One of the goals of feedback scheduling is to better make use of scarce resources. If this should be achieved it requires that the feedback scheduling mechanism itself does not consume too much resources. Hence, efficient feedback scheduling mechanisms are of great importance.

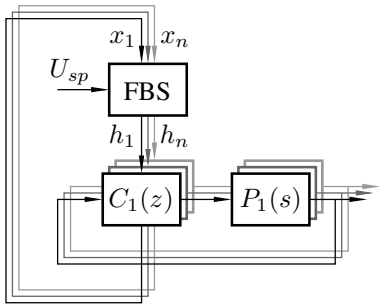

Figure 3 Feedback scheduling of control loops.

4. Feedback Scheduling of Control Systems

Feedback-based resource scheduling is of particular interest for control systems. Here the common assumption is that a control system involving multiple control loops is implemented as a multi-tasking system with each controller being realized as a separate periodic task. The main resource of concern in these types of problems is the CPU-time. The objective for the feedback scheduler is to dynamically adjust the CPU utilization of the controller tasks so that the task set remains schedulable and the stability and performance requirements of the individual controllers are met.

The structure is shown in Fig. 3. The controllers are denoted $C_i(z)$ and the physical plants are denoted $P_i(s)$. Control is used at two levels: to control a number of physical plants and to control the resource allocation to the controllers.

In this approach the control performance can be viewed as a QoS parameter. The feedback scheduling problem is often stated as an optimization problem where the objective is to maximize the global control performance according to some criterion, subject to resource and schedulability constraints.

There are several reasons why feedback scheduling can be applied to control systems. One reason is the uncertainty associated with the worst-case execution time (WCET) estimation. An overly pessimistic WCET estimate may cause the designer to chose a more powerful processor, which then will be under-utilized. Alternatively, the designer will reduce the utilization by increasing the task periods, which may lead to poor control performance. In some control applications, e.g., hybrid and switching controllers and controllers employing on-line optimization, the computational workload can change dramatically over time as different control algorithms are switched in and out when the external environment changes, and from job to job due to the varying number of iterations that are needed in the optimization.

An optimization-based approach to feedback scheduling requires performance metrics that are parameterized with

scheduling-related parameters, e.g., task periods. For general applications this is normally not available. However, for control application such performance metrics often exist. For example, using tools such as Jitterbug, [Lincoln and Cervin, 2002], it is possible to evaluate variance-type performance indices for linear control systems as a function of sampling periods.

4.1 Actuators & Sensors

The actuators of a feedback scheduler are the means with which the scheduler can modify the CPU utilization. For a controller task the task period is a natural actuator. Depending on the performance requirements one can either adjust the controller parameters when the task period is changed, i.e., use gain scheduling with respect to the sampling interval, or use the same controller parameters independently on the task period.

An alternative actuator is the execution time of the controller. This can be realized using a *multiple-versions* approach or using an *anytime* approach. In a multiple version approach one may use multiple control algorithms with different execution time demands or one may occasionally skip the execution of parts of the control algorithm. The anytime approach can be applied to controllers in which the computations in the control algorithm or the sensors can be expressed in an iterative way that may be terminated after an arbitrary number of iterations, and where the control performance increases with the number of iterations. One example where this is applicable is model-predictive controllers (MPC) in which a quadratic optimization problem is solved in every sample. In the case of overload, the optimization may be terminated early and still produce acceptable results. In [Henriksson *et al.*, 2004b], value-based dynamic scheduling of multiple MPC controllers is considered using the optimization cost function as the value function. However, in general, changing the task periods is more natural for control algorithms than changing the execution time demands.

The sensor in this type of feedback scheduler is a measurement of the actual CPU utilization. This assumes that the processor and RTOS are equipped with the means to perform such measurements. In order to avoid control actions caused by spurious measurement outliers ("noise") a low-pass filter may be included in the sensor. The low-pass filter can also be used to calculate an average of the utilization over a certain time period, e.g., the sampling period of the feedback scheduler.

4.2 Stationary Feedback Scheduling

In [Eker *et al.*, 2000] it was shown that a simple linear proportional rescaling of the nominal task periods in order to meet the utilization set-point is optimal with respect to the stationary control performance under certain assumptions. It holds if the control cost functions, $J_i(h_i)$, where h_i is the sampling period, are quadratic, i.e.,

$$J_i(h_i) = \alpha_i + \beta_i h_i^2$$

or if they are linear,

$$J_i(h_i) = \alpha_i + \gamma_i h_i,$$

and if the objective of the feedback scheduler is to minimize the sum of the control cost functions or a weighted sum of the control cost functions. Linear or quadratic cost functions are quite good approximations of true cost functions in many cases.

The advantage of this approach is a simple and fast calculation that easily can be performed on-line. The linear rescaling also has the advantage that it preserves rate-monotonic ordering of the tasks and, thus, avoids any changes in task priorities in the case that rate-monotonic fixed priority scheduling is used. It is also possible to add more constraints to the optimization problem and still retain a simple solution. For example, one can use the nominal task periods as minimum task periods and use these whenever the utilization is less than the utilization set-point.

It is also possible to assign maximum periods to certain tasks. This leads however to an iterative computation (LP-problem) in order to find the total rescaling of all the tasks.

4.3 Optimal Feedback Scheduling

A drawback with the previous approach is that it does not consider the actual control performance. The optimization only concerns the stationary performance. Disturbances acting on the control loops will not be taken into account in the optimization. In [Henriksson and Cervin, 2005] an alternative approach is proposed. Instead of basing the optimization on stationary cost functions it is based on finite-horizon cost functions related to the sampling period, the current state of the control loop, and the period at which the feedback scheduler is invoked. The optimization horizon corresponds to the period of the feedback scheduler. Hence, rather than having cost functions that only are a function of the task periods, i.e.,

$$J_i(h_i) = \alpha_i + \beta_i h_i^k,$$

the cost functions now can be expressed as

$$J_i(h_i, x_i, T_{fbs}) = \alpha_i(x_i, T_{fbs}) + \beta_i(x_i, T_{fbs}) h_i^k.$$

The intuition behind this formulation is that a process in a transient phase, for example, during a setpoint change or exposed to an external disturbance, may require more computing resources than a process in stationarity. In [Henriksson and Cervin, 2005] a stationary noise model was assumed,

causing the current plant state to have a quite small influence on the assigned periods. In [Castañé *et al.*, 2006] the approach is extended to also handle non-stationary noise processes.

The approach is formulated for arbitrary linear controllers. The optimization objective is to minimize the combined performance of all the control loops,

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

subject to the utilization bound given by the schedulability condition

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

This problem is a convex problem if the functions $J_i(1/f_i, x_i, T_{fbs})$ are convex in f_i. If all the cost functions have the same shape then an explicit solution exists. If that is not the case analytical solutions exist only for special cases. In other cases the cost functions are approximated as linear functions at the current sampling period and the cost function derivatives for each controller are computed off-line and stored in look-up tables.

An important design parameter is the feedback scheduler period. The shorter period, the more responsive the system will be to external disturbances. However, the execution of the feedback scheduler induces overhead and consumes CPU time from the control tasks.

4.4 Value-Based Feedback Scheduling

Most of the feedback scheduling approaches proposed for control applications are indirect. By adjusting the task parameters, e.g., period and execution time, one makes sure that the task set is schedulable and has certain timing properties (latencies and jitter).These timing properties will then indirectly determine the performance of the application. The problem with this is the relationship between the timing parameters and the cost/performance. Often the relationship only holds in stationarity and in a mean-value sense.

An interesting but still largely unexplored approach is to instead use value-based or direct feedback scheduling. Here, the idea is to base the decision of which task to execute on an instantaneous cost function. This cost function should grow the longer the control loop executes in open loop and decrease when a control action is issued. The instantaneous cost could then be used as a dynamic task priority similar to the deadline in EDF. The resulting system would be a special case of an aperiodic event-triggered sampled system.

4.5 Research Directions

The challenges and resource directions for feedback scheduling of control tasks include all the challenges and research direction of control of CPU resources. Additionally, the following items are important. Temporal robustness indices are needed that allow us to decide how the control performance depends on the computing resources, e.g., on the sampling period. Although work has and is being done in this area more work is necessary. Frameworks must be developed that allow dynamic negotiation about resources and control performance between the control applications and the QoS manager. These frameworks must be able to express the control-specific aspects of the problem in addition to the computing and scheduling-specific aspects. It must also be able to express the performance requirements of the different control loops in a flexible way.

Formal performance guarantees on control loops is something that today require a fairly static implementation of the control system with respect to resource utilization, e.g., a statically scheduled time-triggered control loop has a very predictable performance. It is an open question whether it is possible to combine the flexibility implied by feedback scheduling with formal guarantees and, in that case, what type of formal guarantees.

5. Control of Communication Networks

The success of the Internet as a worldwide information carrying network can be attributed to the feedback mechanisms that control the data transfer in the transport layer of the IP stack. These algorithms have historically managed to distribute network resources among contending users in a sufficiently fair and resource-efficient way. An explanation to this is that the control is allocated at the end-systems (users) and hence obey a decentralized structure. Furthermore, together with the source control, buffers have played a key-role during the evolution of the Internet. Since end-users base control action on limited, corrupt and delayed information; buffers are used at links inside the network to smooth out errors in the control, hence making the system more robust. Auxiliary control from the network interior has also been introduced by "intelligent" links that mark or drop packets depending on the traffic load. This is referred to as Active Queue Management (AQM).

Historically, congestion control algorithms have been designed by computer scientists outside the framework of control theory. The tremendous complexity of the Internet makes it extremely difficult to model and analyze, and it has been questioned if mathematical theory can offer any major improvements in this area. Recently, however, significant progress in the theoretical understanding of network congestion control has been made following seminal work by Kelly and coworkers [Kelly *et al.*, 1998]. The key is to work at the correct level of aggregation, which is fluid flow models with validity at longer time-scales than the round-trip time. By explicitly modeling the congestion measure signal fed back to sources, posing the network flow control

as an optimization problem where the objective is to maximize the total source utility, it is shown that the rate control problem can be solved in a completely decentralized manner [Kelly *et al.*, 1998; Low and Lapsley, 1999]. This assumes that each source has a (concave) utility function of its rate.

To ensure that the system will reach and maintain a favorable equilibrium, it is important to assess the dynamical properties, such as stability and convergence, of the schemes. Stability of the basic schemes, which allow dynamic rate control and static marking, or dynamic queue management schemes and static source rate control, was established already in [Kelly *et al.*, 1998; Low and Lapsley, 1999] but under idealized settings. A unifying framework for global stability of congestion control laws based on passivity has been proposed in [Wen and Arcak, 2004].

The above results have all ignored the effect of network delay, and assumed that price information is available instantaneously at the source, that the sources take immediate action, and that the new rates affect the link prices instantaneously. However, stability of the protocols in equilibrium depends critically on the feedback delay. Recent research therefore focuses on source- and link control laws that guarantee stability for more general network configurations and delay distributions.

Wireless networks are specially interesting from a resource control point of view. Whereas the link capacities in wireline networks are fixed, the capacities of wireless links can be adjusted by the allocation of communication resources, such as transmit powers, bandwidths, or time-slot fractions, to different links. Adjusting the resource allocation changes the link capacities, influences the optimal routing of data flows, and alters the total utility of the network. Hence, optimal network operation can only be achieved by coordinating the operation across the networking stack. This is often referred to as *cross-layer optimization*.

A basic question is whether it is worthwhile to introduce advanced resource management and coordination schemes. One way of attacking this problem is to try to determine the *information-theoretic capacity*, which includes optimization over all possible modulation and coding schemes and involves many of the unsolved problems of network information theory. An alternative approach is to focus on *network layer capacity*, where coding and modulation schemes are fixed, and one optimizes over some critical parameters, such as power allocations and scheduling decisions.

5.1 Research Challenges

Control-based approaches in communication networks is a very large research field, in particular if wireless systems, e.g., sensor networks, are included. In order to be able to control the network performance it is necessary to measure and modify the network parameters. The current ISO-OSI stack layer is not ideally supported for cross layer designs where information from the lower layers must be made available at the application layer and where the application layer must be able to modify the behaviour of the lower layer protocols dynamically. Hence, new protocols and protocol models are needed that simplify this.

Theories and engineering principles for dynamically allocating resources in wireless ad hoc networks to ensure quality of service are needed for a wide range of applications. One interesting suggestion is to have a formal, possibly optimization-based, theory for the design of network protocols based on a model of the underlying network and a specification of the application requirements.

While the use of mathematical decomposition techniques as guiding principle for organizational design is well-known in economics and operational research (e.g., [Holmberg, 1995]), the application of such ideas to networked systems has just begun to appear [Chen *et al.*, 2005; Chiang, 2005; Lin and Shroff, 2004; Johansson and Johansson, 2006].

Breaking up the layered structure of the networking stack may also have negative consequences, partially in terms of maintenance and compatibility issues but also in terms of the resulting performance. In particular, it has been observed that cross-layer coordination protocols can introduce dependency relations and unintended interactions [Kwadia and Kumar, 2005]: in some situations, adaption mechanisms in different layers can start working against each other, leading to worse practical performance than in a layered network. It is thus important to develop control-theoretic tools for analyzing protocol dynamics in order to guarantee stable and efficient overall behaviour.

The control of network performance often requires access to network state variables, such as available bandwidth, round-trip times, and packet loss. These variables are typically not immediately available, but must be estimated from other quantities. The design of reliable and efficient estimators for network state is thus instrumental for many applications, and requires the development of simple and flexible models of network dynamics together with the associated advances in estimation theory.

Improving congestion control is intimately linked to the quality of the used models. The development of accurate fluid flow models will help understanding the limitations communication networks are subject to and provide a basis for new control laws. Present fluid flow models disregard important system aspects and very little has been done in terms of experimental validation of the proposed models.

6. Error Control of Software Systems

The development of completely defect-free complex software systems is extremely difficult, if not impossible. At the same time several large existing software systems are remarkably stable and reliable in the presence of thousands

or maybe millions of residual software bugs, e.g., the telecom networks or the Internet. Hence, rather than focusing the development effort on trying to eliminate all bugs at design time it is important to develop methods that allow us to develop safe and stable software systems that still can utilize COTS-quality software components with a considerable amount of residual bugs. Hence, the focus should be on detection and recovery from software errors at run-time, in addition to elimination of software errors at design-time.

The idea behind error control of software is to use techniques from feedback control in order to detect malfunctioning software components and, in that case fall back on, a well-tested core software component that is able to provide the basic application service with guarantees on performance and safety. Hence, the basic idea assumes that a certain amount of defect-free components are available, that can be used to implement the fall-back safety core service. The second key idea is to always design your system to have a simple and well-formed dependency tree, with a minimal number of dependency relations among components. This is necessary in order to be able to identify the core services and keep them small. The background to several of the key ideas of the area is given in [Sha, 2001]. In [Liu *et al.*,] a fault tolerance architecture is presented based on these ideas.

In software error control, our view of what control is has to be broadened substantially. Control is normally concerned with the temporal behaviour of systems. The ideas behind software error control are, however, not restricted to the temporal behaviour. The same approach can in principle also be used for applications that only contain functional requirements. In this case software error control has strong relationships to techniques that are commonly associated with fault tolerance, e.g., hardware and software redundancy and diversity through replication and N-version programming. However, the principles behind software error control have so far mainly been applied to reactive applications, i.e., avionics control systems.

6.1 Research Challenges

The major challenge is to develop a new paradigm for software stability control, based on an integration of concepts from fault-tolerant computing and control, that is applicable to a wide range of application types. The number of documented examples where software error control has been applied is small. In order to increase the understanding for the subject and to develop the necessary methods and theory, more documented applications must be developed. The relationships to the methods within the traditional fault tolerance area must also be clarified. It is further necessary to investigate for which application types, other than feedback control, the approach is suitable.

7. Control Middleware

Applying control techniques to a computer software system requires certain generic services. Often these have to do with the sensor and actuator interface of the control loop. For example, in queue delay control it is necessary to be able to measure arrival and departure times of requests and calculate average delays. Rather than implementing the support for this in every application or provide the support in the operating system (often not possible) an alternative is to use middleware technology.

Several middleware technologies are available, e.g., Java-RMI, Microsoft's COM, and CORBA. Middleware frameworks are also available for real-time and embedded systems, e.g., RT-CORBA and Embedded CORBA. There are also a large number of research middleware framework developed for pervasive networked embedded system applications, e.g., sensor networks. Examples of these are GAIA [Romn *et al.*, 2002] and AURA [Garlan *et al.*, 2002].

A few middleware solutions have been developed explicitly for control purposes. ControlWare, [Zhang *et al.*, 2002], is a middleware QoS-control architecture originally designed to help programmers apply control theory to control software performance. It allows the user to express QoS specifications off-line, maps these specifications into appropriate feedback loops, and connects the loops to the right performance sensors and actuators in the application code such that the desired QoS is achieved [Abdelzaher *et al.*, 2003]. The aim of ControlWare is to isolate the software application programmer from control technology issues while still utilizing the theory.

The basic abstraction provided in ControlWare is a component. Components are connected via their ports, and communicate with each other via an infrastructure named Softbus. Properly connected, several components (various number of sensors, actuators and controllers) form a control loop. Two main types of software sensors and actuators are supported: passive and active. A passive sensor or actuator is simply a function or software component that returns sensor data or accepts a command to perform an actuation when called by a controller. An active sensor, on the other hand, is a thread that usually is awakened periodically by the operating system to perform sensing or actuation. The topology of a control loop is described by a template. Essentially, a template describes a general solution to a type of QoS guarantee. Several QoS performance control templates are supported. For example, absolute convergence guarantees, relative differentiated service guarantees, prioritization, and optimization guarantees. The absolute convergence guarantee ensures that some performance metric R converges asymptotically to a desired value and that the error is bounded at all times. In [Zhang *et al.*, 2002] the corresponding control loop templates for relative differentiated service, prioritization, and utility maximization are presented.

A related example is IBM's AutoTune Agents, see [Diao *et al.*, 2002], where an agent-based solution is proposed which automates the tuning of the IT environment for eCommerce applications and also automatically designs an appropriate tuning mechanism for the target system. Using AutoTune agents are constructed to automate a control-theoretic methodology that involves model building, controller design, and run-time feedback control.

The Agilos (Agile QoS) architecture, [Li and Nahrstedt, 1999], is a middleware control architecture designed to provide middleware services to assist application-aware QoS adaptations. Agilos is designed as a three-tier architecture: In the first and lowest tier, application-neutral adaptors and observers maintain tight relationships with individual types of resources, and react to changes in resource availability. In the second tier, application-specific configurators are responsible for making decisions on when and what adaptive mechanisms are to be invoked. In the third tier, a gateway and negotiators are introduced to control adaptation behavior in an application with multiple clients and servers, so that dynamic reconfigurations of client-server mappings are possible and tuned to the best interests of the application. The adaptation algorithm in Agilos is based on PID control.

FCS/nORB is a feedback control real-time scheduling service on nORB, a small footprint Object Request Broker (ORB) middleware for networked embedded systems, [Lu *et al.*, 2003]. FCS/nORB provides support for real-time performance portability across platforms and robust performance guarantees in face of workload/platform variations. Three types of control loops are supported: control of CPU utilization, control of deadline miss ratio, and combined control of utilization and miss ratio. The same group is also developing CAMRIT, a control-based adaptive middleware framework for real-time image transmission in distributed real-time embedded systems, [Wang *et al.*, 2004].

There are also other types of middleware associated with control. However, the majority of these are intended for real-time control, i.e., control of some physical system using some type of networked embedded control system. One system worth mentioning, however, is Etherware, [Baliga *et al.*, 2004], a messaging middleware for networked control loops.

7.1 Research Directions

The most important research item for control middleware is to develop these systems from research prototypes to commodity systems. It is still an open question whether the middleware only should be passive, i.e., provide sensing and actuation services that the application can use to itself implement the feedback control, or if it should be active, i.e., provide the actual control loops. Both of these approaches have advantages and disadvantages. Another unclear issue is when the support for feedback control should be provided by a middleware and when the support should be provided by the underlying OS. Clearly for severely resource constrained applications the addition of a middleware layer may not be an option.

8. Acknowledgements

This work has been funded by the EU/IST FP6 NoE ARTIST2. Important input to the roadmap were provided by the participants of the Lund Workshop on Control for Embedded Systems, June 2005. We also gratefully acknowledge the valuable comments of the reviewers.

9. References

Abdelzaher, T., J. Stankovic, C. Lu, R. Zgang, and Y. Lu (2003): "Feedback performance control in software services." *IEEE Control Systems Magazine*, **23:3**.

Aparah, D. (1998): "Adaptive resource management in a multimedia operating system." In *Proceedings of the 8th International Workshop on Network and Operating System Support for Digital Audio and Video.*

Baliga, G., S. Graham, L. Sha, and P. Kumar (2004): "Service continuity in networked control using Etherware." In *Proceedings of Middleware 2004.*

Buttazzo, G., G. Lipari, and L. Abeni (1998): "Elastic task model for adaptive rate control." In *Proc. 19th IEEE Real-Time Systems Symposium*, pp. 286–295.

Castañé, R., P. Martí, M. Velasco, A. Cervin, and D. Henriksson (2006): "Resource management for control tasks based on the transient dynamics of closed-loop systems." In *Proceedings of the 18th Euromicro Conference on Real-Time Systems.* Dresden, Germany.

Chen, L., S. Low, and J. Doyle (2005): "Joint congestion control and media access control design for ad hoc wireless networks." In *Proc. of IEEE Infocom.* IEEE, Miami, FL.

Chiang, M. (2005): "Balancing transport and physical layers in wireless multihop networks: Jointly optimal congestion control and power control." *IEEE JSAC*, **23:1**, pp. 104–116.

Chu, H. and K. Nahrstedt (1999): "CPU service classes for multimedia applications." In *Proceedings of the IEEE International Conference on Multimedia Computing and Systems.*

Diao, Y., N. Gandhi, J. Hellerstein, S. Parekh, and D. Tilbury (2002): "MIMO control of an Apache web server." In *Proceedings of the American Control Conference.*

Eker, J., P. Hagander, and K.-E. Årzén (2000): "A feedback scheduler for real-time control tasks." *Control Engineering Practice*, **8:12**, pp. 1369–1378.

Garlan, D., D. P. Siewiorek, A. Smailagic, and P. Steenkiste (2002): "Aura: Toward distraction-free pervasive computing." *IEEE Pervasive Computing.*

Hellerstein, J., Y. Diao, S. Parekh, and D. Tilbury (2005): "Control engineering for computing systems." *IEEE Control Systems Magazine*, **25:6**, pp. 56–68.

Hellerstein, J. L., Y. Diao, S. Parekh, and D. M. Tilbury (2004): *Feedback Control of Computing Systems*. John Wiley.

Henriksson, D. and A. Cervin (2005): "Optimal on-line sampling period assignment for real-time control tasks based on plant state information." In *Proceedings of the 44th IEEE Conference on Decision and Control and European Control Conference ECC 2005*. Seville, Spain.

Henriksson, D., Y. Lu, and T. Abdelzaher (2004a): "Improved prediction for web server delay control." In *Proceedings of the 16th Euromicro Conference on Real-Time Systems (ECRTS 04)*. Catania, Sicily, Italy.

Henriksson, D., Y. Lu, and T. Abdelzaher (2004b): "Improved prediction for web server delay control." In *submission to Euromicro Conference on Real-Time Systems*. Catania, Sicily, Italy.

Holmberg, K. (1995): *Design Models for Hierarchical Organizations: Computation, Information, and Decentralization*, chapter Primal and dual decomposition as organizational design: Price and/or resource directive decomposition, pp. 61–92. Kluwer Academic Publishers.

Johansson, B. and M. Johansson (2006): "Mathematical decomposition techniques for distributed cross-layer optimization of data networks." *IEEE Journal on Selected Areas in Communications*, November. (to appear).

Kelly, F., A. Maulloo, and D. Tan (1998): "Rate control in communication networks: shadow prices, proportional fairness and stability." *Journal of the Operational Research Society*, **49**, pp. 237–252.

Kleinrock, L. (1975): *Theory, Volume 1, Queuing Systems*. Wiley-Interscience.

Kwadia, V. and P. R. Kumar (2005): "A cautionary perspective on cross layer design." *W. IEEE Wireless Communication*, **12:1**, pp. 3–11.

Li, B. and K. Nahrstedt (1999): "A control-based middleware framework for quality of service adapations." *IEEE Journal on Selected Areas in Communications*, Sep, pp. 1–19.

Lin, X. and N. B. Shroff (2004): "Joint rate control and scheduling in multihop wireless networks." In *Proc. of IEEE Conference on Decision and Control*. IEEE, Paradise Island, Bahamas.

Lincoln, B. and A. Cervin (2002): "Jitterbug: A tool for analysis of real-time control performance." In *Proceedings of the 41st IEEE Conference on Decision and Control*. Las Vegas, NV.

Liu, X., H. Ding, K. Lee, L. Sha, and M. Caccamo "Feedback based real-time fault tolerance – issues and possible solutions.". First International Workshop on Feedback Control Implementation and Design in Computing Systems and Networks, FEBID'06.

Liu, X., J. Heo, L. Sha, and X. Zhu (2006): "Adaptive control of multi-tiered web applications using queueing predictor." In *Proc. 2006 IEEE/IFIP Network Operations & Management Symposium*. Vancouver, Canada.

Low, S. H. and D. E. Lapsley (1999): "Optimization flow control – I: Basic algorithm and convergence." *IEEE/ACM Transactions on Networking*, **7:6**, pp. 861–874.

Lu, C., X. Wang, and C. Gill (2003): "Feedback control real-time scheduling in orb middleware." In *Proceedings of the 9th IEEE Real-Time and Embedded Technology and Applications Symposium*.

Robertsson, A., B. Wittenmark, and M. Kihl (2003): "Analysis and design of admission control in web-server systems." In *Proceedings of the 2003 American Control Conference (ACC'03)*, pp. 254–259. Denver, Colorado, USA.

Romn, M., C. Hess, R. Cergueira, R. Campbell, and K. Nahrstedt (2002): "Gaia: A middleware infrastructure to enable active spaces." *IEEE Pervasive Computing*, Oct-Dec.

Sha, L. (2001): "Using simplicty to control complexity." *IEEE Software*, **18:4**, pp. 20–28.

Sha, L., X. Liu, Y. Lu, and T. Abdelzaher (2002): "Queuing model based network server performance control." In *IEEE Real-Time Systems Symposium*.

Stankovic, J. A., C. Lu, S. H. Son, and G. Tao (1999): "The case for feedback control real-time scheduling." In *Proc. 11th Euromicro Conference on Real-Time Systems*, pp. 11–20.

Tipper, D. and M. K. Sundareshan (1990): "Numerical models for modeling computer networks under nonstationary conditions." *IEEE Journal on Selected Areas in Communications*, **8:9**, pp. 1682–1695.

Wang, X., H.-M. Huang, V. Subramonian, C. Lu, and C. Gill (2004): "Camrit; control-based adaptive middleware for real-time image transmission." In *Proceedings of the IEEE Real-Time and Embedded Technology and Applications Symposium*.

Wen, J. T. and M. Arcak (2004): "A unifying passivity framework for network flow control." *IEEE Transactions on Automatic Control*, **49:2**, pp. 162–174.

Xu, W., X. Zhu, S. Singhal, and Z. Wang (2006): "Predictive control for dynamic resource allocation in enterprise data centers." In *Proc. 2006 IEEE/IFIP Network Operations & Management Symposium*. Vancouver, Canada.

Yuan, W. and K. Nahrstedt (2003a): "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems." In *19th ACM Symposium on Operating Systems Principles*. Bolton Landing, NY.

Yuan, W. and K. Nahrstedt (2003b): "Energy-efficient soft real-time CPU scheduling for mobile multimedia systems." In *Proc. of the 19th ACM Symposium on OPerating Systems Principles, Bolton Landing, NY*.

Zhang, R., C. Lu, T. Abdelzaher, and J. Stankovic (2002): "Controlware: A middleware for feedback control of software performance." In *Proceedings of the 2002 International Conference on Distributed Computing Systems, Vienna, Austria*.