

Switch Scheduling and Network Design for Real-Time Systems*

Sathish Gopalakrishnan Marco Caccamo Lui Sha
Department of Computer Science
University of Illinois at Urbana-Champaign

Abstract

The rapid need for high bandwidth and low latency communication in distributed real-time systems is driving system architects towards high-speed switches developed for high volume data transfer on the Internet. These switches employ complex scheduling algorithms for transferring data cells from the input line to the output line. From a real-time systems perspective, it is necessary to understand the behavior of these switching algorithms and obtain worst-case delay bounds for message transfer across these switches. Many researchers have derived average-case delay bounds for switching algorithms but mission-critical systems require guarantees for the worst-case. In this context, we derive upper bounds on cell delays across commonly available switches. Our bounds provide a starting point for research in this direction. Moreover, we use the delay bounds to construct low-cost networks of switches given a set of processors and their real-time communication requirements. Experiments with the heuristic algorithm that we propose for network design have produced encouraging results. Importantly, the algorithm is independent of the delay analysis technique and better techniques can be incorporated trivially. By addressing the network design problem, we hope to transform the system architecting process from a manual, ad-hoc operation to a simple and automated step that will result in better designs and cost savings.

1. Introduction

A real-time system can be defined as “a system in which the time at which an output is produced or an action is taken is significant.” This is usually because the input corresponds to some observation in the physical world and the output or action is related to the input from the physical world. When actions need to be taken in a timely fashion, a requirement of a real-time system is a high level of determinism. The system must behave

in a predictable manner, not only functionally, but also with respect to non-functional attributes, especially the time of completion of certain actions.

Many embedded systems are, by their very nature, real-time systems. An avionics system is a canonical example of a real-time system; many flight control and navigational activities need to be completed within well-defined temporal limits following the generation of inputs from appropriate sensors to maintain aircraft stability. The sensing activity itself is periodic and therefore most real-time systems have several periodic tasks and each instance of a periodic task needs to finish before a known deadline.

Current and next-generation embedded applications are being built on distributed systems with many communicating elements. System architectures have been evolving to take advantage of greater silicon integration and performance, and traditional bus-based communication architectures have reached their limits. Many applications now demand several Mbits/second bandwidth (Example: Avionics architecture definition [4] indicated in Table 1). Yet another application with high bandwidth and low delay requirements is synthetic vision for pilots. Cameras (image sensors, infrared sensors) placed at different positions of an aircraft need to deliver several Mbits of data within about 200 ms of acquiring the data. Such applications can also be envisioned in a smart building with cameras and sensors streaming data to a security center. An SXGA signal has a 1280×1024 pixels resolution: with a 16-bit color depth and thirty frames per second update rate, this signal requires a bandwidth of nearly 600 Mbits/s. The move to high-speed switched fabric networks is underway because of the much higher bandwidths they support, but precise characterization of message delays are hard to obtain. Riddel provides a clear picture of the emerging trends in avionics networks and illustrates a potential architecture [26].

A network switch is an element with multiple input and output ports – packets received at an input port are transferred to the appropriate output port (that leads to the destination) by creating a connection between the input and output ports. These switches transfer data in discrete units known as *cells* (which may

*This work was supported by NSF grants CCR-0237884, CCR-0325716 and NSF CNS-0509268, and by the MURI program under grant N00014-01-0576.

Application	Date Rate
Situation awareness	150-700 Mbts/sec
Navigation	150-700 Mbts/sec
Threat warning	150-700 Mbts/sec
Synthetic aperture radar	200-800 Mbts/sec
Special receiver	200-400 Mbts/sec
Laser warning	50-100 Mbts/sec

Table 1. Data rate projections for avionics applications by Year 2010

simply be packets or groups of packets). At each cell time, one cell can be transmitted on any link. A scheduler within the switch determines the sequence in which cells leave the switch. Many switch scheduling approaches have been proposed (and some have been implemented). These approaches differ with regard to where packets are queued (at the input port, the output port or at both ports) and how input and output ports are matched. These choices affect the delays experienced by packets traversing the switch. The question we would like to answer is: how do we design networks of such switches that can cope with the dual requirements of high bandwidth and low delay?

The transition towards using commercial off-the-shelf (COTS) networking technology has led to reductions in the development cycle for systems but has increased the complexity of analysis because components are being used in situations different from what they were originally designed for. Most commercial high-speed switches were designed for Internet traffic and the primary objective was to guarantee 100 percent utilization of network links. Internet operators such as AT&T, Sprint and Verizon maintain several expensive long-distance links and commercial considerations drove the focus of high link utilization. Delay was an important consideration concern, but not the primary concern at the design stage. Modern switches try to minimize the average packet delay but real-time systems require bounded (maximum) message delay.

Non-determinism cannot be tolerated in most time-critical applications (such as avionics and industrial automation) because the penalty of failure is enormous, and therefore it is necessary to bound the maximum message delay through a switch. In systems built using bus-based networks, it is possible to guarantee delays because of the many levels of priorities that are offered. The FutureBus+, for instance, has 258 levels of priorities for arbitration, consistent treatment of priorities during arbitration, message passing and DMA protocols [27]. The challenge with high-speed switches is that, even though they are capable of high bandwidth data transfers, they use scheduling algorithms that are hard to characterize and do not have many priority levels – as an example, the Cisco MDS 9000 family of switching modules offer only two priority levels for quality of service (QoS) [10]. Determining de-

Product	Ports	Price (\$)	Price/Port (\$)
MDS 9216	16	38,800	2,425
MDS 9216	32	71,599	2,237
MDS 9216	48	79,999	1,667
MDS 9509	64	176,187	2,753
MDS 9509	144	266,885	1,853
MDS 9509	224	357,783	1,597

Table 2. Prices for the Cisco MDS 9000 switch family [29]

lay bounds for real-time traffic with deadlines when the switch supports very few priority levels is difficult. A switch scheduling algorithm tries to maximize throughput by matching input ports to output ports; matching is a complex operation to analyze. The absence of a sufficient number of priority levels has other implications: non-real-time traffic, which could be scheduled at the lowest priority level in bus-based systems, can also affect high priority traffic and needs to be considered in the analysis.

Even when latency bounds are known, the design of the network (of switches) that connects the nodes of the distributed system requires considerable time and effort. As these distributed systems grow in scale, manual techniques are likely to result in expensive, over-provisioned designs. In a factory network with processing nodes in the hundreds, the cost of the network is of the order of a few hundred thousand dollars (Table 2) and poor network designs can increase this cost by as much as 50 percent. Architectural mistakes are often subtle and hard to detect, but the implications may be very expensive. Degraded performance is not uncommon and the impact of (poor) ad-hoc and manual choices are likely to be exacerbated with the increasing numbers of processing nodes in a real-time environment. The constraints on network design are two-fold: one, increasing the number of nodes that are connected to a switch increases the delay of messages routed through the switch, and, two, each switch and processing node is limited by the number of ports they can support. The design problem is one of constructing a network such that all messages can meet their deadlines while ensuring that no device (node or switch) violates its port constraint.

In this paper, we present an upper-bound for the worst-case latency that can be incurred by messages routed using typical commercially available high-speed switches. We use these bounds in an algorithm for designing low-cost switching networks for real-time applications. The delay bounds that we obtain tend to be highly pessimistic but provide initial results to aid the system architecting process. The algorithm for designing the network, however, performs admirably when compared with other algorithms that might use similar delay estimates. If, in the future, better estimates for delay can be obtained, the network design algorithm

can still be used with the minor change in the procedure for estimating the delays. We believe that this is the first work to consider message delays across high-speed switches in conjunction with the network design problem for real-time systems. We envision our work being integrated within a suite of tools to enable rapid design of large real-time systems.

2. Delay and Burstiness Bounds for FCFS Scheduling

With the limited priority levels available on many switches, it is important to consider the delay bounds and burstiness characteristics of first-come first-serve (FCFS) scheduling that is employed by many switches. FCFS is a special case of strict priority based scheduling – every task or job has the same priority.

We highlight two results that are essential to our characterization of message delays across high-speed switches. The first result is an upper bound on the delay of messages scheduled using the FCFS discipline. The second result is a bound on the burstiness at the output that results when jobs are scheduled with this discipline. When jobs are scheduled on an FCFS basis, even if the arrivals are strictly periodic, the departures from the queue need not be periodic; the burstiness is an indicator of the distortion in the periodicity of the jobs as they leave the queue.

We assume that a message i , of length C_i , arrives to the queue every T_i time units. The burstiness bound B_i indicates that over a time interval T , at most $\lfloor \frac{T}{T_i} \rfloor + B_i$ instances of task i arrive to the queue.

Theorem 1. *If tasks are scheduled using the FCFS policy at a queue, and each task is characterized by length C_i , period T_i and burst parameter B_i , an upper bound on the delay of a job is $\sum_i B_i C_i$.*

Proof. Let us define a *busy interval* of the queue to be any time interval during which the server is always busy. We only need to consider busy intervals because the queue is empty otherwise. Suppose that a busy interval of length T starts at $t = 0$. Let $N(t)$ be the amount of work in the queue at time t , $A(t)$ be the amount of work that arrived in $[0, t]$ and $S(t)$ be the amount of work completed in $[0, t]$.

For any $t, 0 \leq t \leq T$, we have

$$N(t) = A(t) - S(t).$$

Also,

$$A(t) \leq \sum_i (\lfloor \frac{t}{T_i} \rfloor + B_i) C_i \text{ and } S(t) = t.$$

We can easily obtain the following upper bound on

$A(t)$:

$$\begin{aligned} A(t) &\leq \sum_i (\lfloor \frac{t}{T_i} \rfloor + B_i) C_i \\ &\leq \sum_i (\frac{t}{T_i} C_i + B_i C_i) \\ &= t \sum_i \frac{C_i}{T_i} + \sum_i B_i C_i. \end{aligned}$$

Now, we have

$$\begin{aligned} N(t) &\leq t \sum_i \frac{C_i}{T_i} + \sum_i B_i C_i - t \\ &= t (\sum_i \frac{C_i}{T_i} - 1) + \sum_i B_i C_i. \end{aligned}$$

For stability of the queue, we must have $\sum_i C_i / T_i \leq 1$ and therefore,

$$N(t) \leq \sum_i B_i C_i. \quad \square$$

When jobs need to traverse a series of FCFS queues, the characteristics of the tasks, especially burstiness, are distorted by the multiplexing of different tasks. The following theorem describes the burstiness in traffic that departs from an FCFS queue. By applying such analysis in sequence, we can bound delays as jobs pass through a network of FCFS queues.

Theorem 2. *For a queue serving tasks with the FCFS discipline, when each task is characterized by a period T_i , burst parameter B_i and length C_i , the burstiness of task i as it exits the queue is*

$$\hat{B}_i \leq 1 + B_i + \frac{\sum_j B_j C_j}{T_i}.$$

Proof. Let us consider the output at an FCFS queue. For a task i , the number of jobs that can finish during a time interval of length T can be at most the number of jobs that arrived to the queue in an interval of length $T + \sum_j B_j C_j$.

To verify that this is true, consider the interval $[0, T]$. A job of task i can leave the queue after service at time 0 if it arrived at time $t' = -\sum_j B_j C_j - C_i$ after incurring the maximum delay. Similarly, the earliest time at which the job that finishes at time T could have arrived is $T - C_i$.

Now, in an interval of length $T + \sum_j B_j C_j$, the number of arrivals is at most

$$A_i = \left\lfloor \frac{T + \sum_j B_j C_j}{T_i} \right\rfloor + B_i.$$

The following inequality then holds: $A_i \leq (T +$

$\sum_j B_j C_j) / T_i + B_i$. From this inequality we deduce that

$$A_i \leq \lfloor T / T_i \rfloor + B_i + \sum_j B_j C_j / T_i + 1.$$

This gives us the the burst parameter

$$\hat{B}_i \leq 1 + B_i + \sum_j B_j C_j / T_i. \quad \square$$

To determine the worst-case end-to-end delay of a message that flows through multiple FCFS queues, we sum the worst-case delays at each queue; before computing the delay at a particular queue, the burstiness caused by the previous queue needs to be taken into account. This analysis can also be used to determine buffer sizes to ensure that buffer overflows do not occur.

With this understanding of latency and burstiness bounds for FCFS scheduling, we begin our study of switch scheduling algorithms. The results for FCFS scheduling will be applied in different ways for different switch scheduling algorithms.

3. Switch Scheduling

High-performance switches and routers are built around fast switching fabrics: input datagrams are divided into fixed-size cells and transferred to output ports. Time is slotted and a cell can be transmitted at an output port every time slot or cell time. There are several possible designs for high-speed switches. We will discuss some of the common approaches to transferring cells from the input ports to the output ports.

To simplify the analysis of switching delays, we introduce some notation summarized in Table 3.

3.1. Output Queueing

Conceptually the simplest switch design involves output queueing (OQ) (Figure 1): an $N \times N$ switch maintains a queue for each of the N output ports. At each cell time, cells that arrive at an input port are transferred to the appropriate output queue using a shared memory backplane. Cells may be transmitted to the output line (from the output queue) either first-come first-serve (FCFS) or through other more sophisticated and complex mechanisms such as strict priority scheduling or weighted fair queueing.

OQ switches are hard to scale because the shared memory requires a bandwidth of $2NR$ (N writes and N reads), where R is the input line rate. As an example, assume that arriving packets are split into 64-byte cells; let the line rate be $R = 10Gb/s$ and $N = 32$. With these parameters, the shared memory requires an access time, $A_t = \frac{64 \times 8}{2 \times 32 \times 10Gb/s} = 400ps$. This access time is well below the access time of static RAM (SRAM) devices and two orders of magnitude smaller than the access time of dynamic RAM (DRAM) making such designs imprac-

tical for very large, fast switches. At moderate sizes, however, shared memory switches are feasible and in fact have been deployed. The M40 backbone router from Juniper Networks [1] and the ATMS2000 chipset from MMC Networks [5] are commercial switches that utilize output queueing.

Delay Bounds for Output Queueing

With OQ, all cells arriving at an input port are directed to the appropriate output queue. The delay depends entirely on the scheduling discipline adopted at the output queue.

For the FCFS discipline, delays at output port j are bounded by

$$D_j \leq \sum_i \sum_k B_{i,j}^k C_{i,j}^k$$

following the theorems from Section 2. There is no differentiation between messages and each message may encounter delays upto D_j .

3.2. Input Queueing

An alternate approach to routing packets from input ports to output ports uses a crossbar switch that is configured using a centralized scheduler. A crossbar switch allows multiple cells to be transferred across the fabric simultaneously; the crossbar operates at the same rate as the input line rates and avoids the congestion present in a shared memory backplane.

Cells arriving at an input port are queued at the input and crossbar interconnection fabric is used to connect input ports to output ports. For this reason, this class of switches is also called an input queued (IQ) switch. Crossbar connections require scheduling: if multiple input ports have cells destined for the same output port, only one of the input ports can be served in one time slot. Traditional IQ solutions maintain one queue at each input and suffer from performance penalties due to head-of-line blocking [16]. Blocking can be removed by virtual output queueing (VOQ) which organizes buffers in each input port into a set of queues where cells awaiting access to the switching fabric are stored according to their output ports [23].

The main challenge in the design of VOQ switches (Figure 2) is the scheduling algorithm, which operates with some knowledge of the state of input queues and controls accesses to the switching fabric. The problem may be formalized as a maximum size or maximum weight matching on a bipartite graph in which the nodes represent the input and out ports and edges represent the cells to be switched. Maximal weight matching maximizes throughput [32] but its hardware implementation is complex. Many algorithms have been proposed for VOQ switches that have lower complexity: these algorithms [20, 22, 6] belong to the maximal/maximum size matching class. Interestingly, it has been shown that maximum size matching can be unstable for certain non-uniform traffic arrival patterns [23, 17], there-

Symbol	Definition
N	Switch size. A switch of size N has N input and N output ports.
$M_{in}(i)$	The number of output ports to which data needs to be sent from input port i .
$M_{out}(j)$	The number of input ports that are sending data to output port j .
$Dest(i)$	The set of output ports (destinations) that input port i needs to send data to. $ Dest(i) = M_{in}(i)$.
$Source(j)$	The set of input ports (sources) that output port j needs to receive data from. $ Source(j) = M_{out}(j)$.
$C_{i,j}^k$	The length (in cells) of the k -th message from input port i to output port j .
$B_{i,j}^k$	The burst parameter of the k -th message from input port i to output port j .

Table 3. Notation

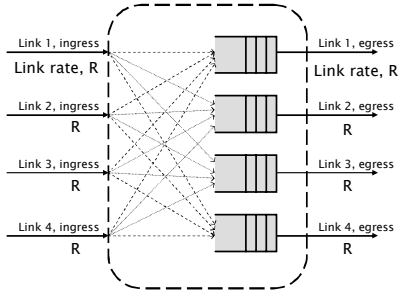


Figure 1. Output queued switch

fore traffic regulation is necessary to ensure that delays are bounded. In a real-time system, traffic is often controllable: schedulability tests provide the limits for schedulability and schedulability is a stronger requirement than stability. By operating within the schedulability region of a switch, we are within the stability region as well.

The iSLIP scheduling algorithm

The iSLIP algorithm [20] is a scheduling algorithm for VOQ switches that has been employed in many commercial routers [21, 2]. It is simple, fast, starvation-free and provides high throughput for best-effort traffic.

iSLIP is a variation on the simple round-robin matching algorithm. Each cell time, it follows three arbitration steps before making a matching:

Step 1: Request Each input sends a request to every output for which it has a queued cell.

Step 2: Grant If an output receives any requests, it chooses the one that appears next in a fixed round-robin schedule, starting from the highest priority element. The output notifies each input about whether its request was granted or not. The pointer to the highest priority element in the round-robin schedule is incremented (modulo N) to one location beyond the granted input if, and only if, the grant is accepted in Step 3.

Step 3: Accept If an input receives a grant, it accepts the one that appears next in a fixed round-robin

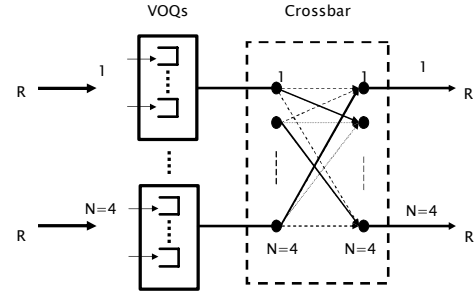


Figure 2. Virtual output queued switch

schedule starting from the highest priority element. The pointer to the highest priority element is incremented (modulo N) to one location beyond the accepted output.

The performance of iSLIP improves with multiple iterations of the arbitration step each cell time; each iteration tries to add connections not made in earlier iterations. *When multiple iterations are performed, the pointers to the round robin schedule are updated only on the first iteration.*

With multiple iterations, iSLIP has the following properties:

Property 1. Connections made in the first iteration become the lowest priority in the next cell time.

Property 2. No connection is starved.

Property 3. Under heavy loads, all queues with a common output have the same throughput.

Property 4. The algorithm will converge in at most N iterations.

Property 5. The algorithm will not necessarily converge to a maximum size match. At best, it will find a *maximal* match: the largest size match without removing connections made in earlier iterations.

In IQ switches, cells experience delays not just due to contention for the output port (essential contention)

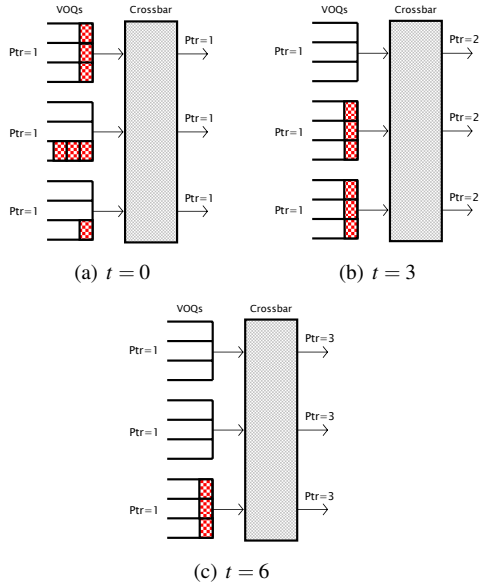


Figure 3. Example state of an iSLIP switch

but also due to contention at the input port for access to the crossbar. The extra contention creates greater uncertainty in computing delay bounds.

Delay Bounds for iSLIP

It is non-trivial to obtain an accurate, analytical characterization of delay in an iSLIP switch because of the possible mismatches in grant and accept pointers. At the worst, the delay a cell at the head of a VOQ may experience is N^2 because the output might have to serve $N - 1$ other inputs first, waiting at most N cell times to be accepted by an input. It is not hard to construct deterministic traffic arrival patterns that might lead to a cell experiencing this delay. In fact, Figure 3 is an example when a packet from input port 3 destined for output port 3 experiences a delay of 9 cell times. The example includes arrivals to the queues over time which result in the worst-case delay and this delay occurs even when the traffic is periodic and considerably light. If we focus on the cell queued at VOQ3 of input port 3, we have evidence of a cell that is delayed for the maximum possible amount of time. At $t = 0$, packets at input port 2 prevent the packet from input port 3 from being transmitted even if $N = 3$ iterations (of iSLIP) are applied. By $t = 3$, packets have arrived to VOQ1 and VOQ2 at input port 3 and this further delays the packet at VOQ3 of input port 3. Further arrivals between $t = 3$ and $t = 6$ delay the packet at VOQ3 of input port 3 to 9 cell times.

Notice that this service time bound leads to output link utilizations of $\frac{1}{N}$ (if each input link utilizes $1/N^2$ of each output link): far from the 100 percent throughput that iSLIP provides to uniform traffic.

It is true that under heavy loads, when each input has at least one cell pending for each output, an

iSLIP switch behaves in like a time division multiplexor (TDM) and each connection is served every N cell times [20]. Since this is a better service guarantee than the N^2 delay bound, one might argue that it is better to keep all queues uniformly loaded by sending dummy packets when necessary. This is definitely possible when a switch is not connected to another switch; when two switches are connected, traffic exiting a switch and entering the next switch cannot be regulated and the connection on the second switch cannot be guaranteed the same service latency.

In general, the maximum possible service latency for a connection from input port i to output port j is

$$L_{ij} = \sum_{i' \in \text{Source}(j)} M_{in}(i')$$

because of the delay that may be introduced by each VOQ. In the example (Figure 3), $M_{in}(i') = N = 3, \forall i'$.

When packets arriving at a VOQ are serviced FCFS, the delay for a message can be bounded by treating the VOQ as a queue with a deterministic service time of L_{ij} per cell. Applying the results from Section 2, we get the following delay bound for messages from port i to port j :

$$D_{ij} \leq L_{ij} \left(\sum_k B_{i,j}^k C_{i,j}^k \right).$$

This is, of course, a very pessimistic bound on the delay of a message (simulations indicate that worst-case delays may be much lower) but it is the best deterministic closed-form expression that can be provided for arbitrary traffic patterns. Even the periodicity of the messages does not simplify the analysis because of the round-robin scheduling and the need to maintain the state of the round-robin pointers. Without exhaustive simulation, it may well be impossible to obtain better bounds even for simple traffic patterns. On the other hand, average delay bounds can be computed with greater accuracy [18] – but for a real-time system average bounds are not sufficient. The analysis described in this section provides quick estimates for the delay encountered by messages traversing a network of switches.

4. Designing a Network of Switches

Having obtained delay bounds for commercially available high-speed switches, we turn our attention to the main problem: How do we design a network that can ensure that messages meet their deadlines in a real-time system? We assume that we are given an allocation of tasks to processors and a set of periodic messages. Each message is specified by its source, destination, periodicity, message size and deadline.

To obtain a network that can support the given traffic, a set of switches need to be selected and decisions

must be made about the connections between processors and switches. The constraints on the design problem arise from the limited number of ports that can be attached to a processor and the number of ports that are supported by a switch. The connections need to be made such that the messages meet their deadlines. Moreover, we would like to find low-cost network designs that can support the real-time message requirements.

As we noted earlier, message delays are a result of contention for the output, and, in the case of IQ switches, contention at the input as well. By minimizing contention at the inputs and outputs, we can provide better delay guarantees – this observation influences our design algorithm.

Preliminaries

The set of processors is represented by the set \mathcal{P} and the set of messages is represented by the set \mathcal{M} . Each message $m_i \in \mathcal{M}$ is associated with a source $s_i \in \mathcal{P}$ and a destination $d_i \in \mathcal{P}$, $s_i \neq d_i$. Each message is of length C_i (cells), and has period T_i and deadline D_i . \mathcal{S} is the set of possible switch types; each switch $s_j \in \mathcal{S}$ has a cost Y_j . The number of ports a device, be it switch or processor, can support is denoted n_i . For simplicity of exposition, we assume that all ports supports the same bandwidth; this is not a requirement of the algorithm or of the systems. There is also a per-switch processing time per message (header processing etc.) which we denote R_i ; this is less than 100ns per message.

Hardness

The network design problem, in the easiest setting of bandwidth allocation, is a generalization of the integral multicommodity flow problem, which is known to be NP-hard [14]. It is possible to reduce the network design problem to one of many NP-hard problems and we do not elaborate further on this aspect here. The hardness of the problem forces us to search for good heuristics that will produce “reasonably good” designs. We discuss one such heuristic in the next section.

5. The PortErr Algorithm

The major constraint on the network design is the number of ports available to a device. If this were not the case, one can theoretically assign a separate wire or link for every required connection and obtain a feasible solution. To build a system that scales in size and to reduce wiring costs, switching solutions are needed and port limitations become the guiding factor for design.

When a network design uses more ports than can be supported by a device, we say that we have a *port error* on the device where the port constraint is violated. Our algorithm iteratively reduces the number of port errors. By routing multiple message streams from (or to) a processor through a switch, we can conserve ports on the processor. When determining the flows that can be

routed along the same link, we compute the resulting delays and verify that no message will miss its deadline.

The first step in our algorithm is to assign individual links for each message stream. These links do not utilize any switch and are point-to-point links between the source and the destination.

We use the notion of a *message group*. A message group consists of messages that use the same network device (a point-to-point link or a switch) to reach their destination. Since each message uses a separate link at the start of the algorithm, each message is in its own message group.

Iteratively, we merge message groups to reduce the number of ports used. When two message groups are merged, we need to select a switch from set \mathcal{S} to route the messages in these message groups (unless the messages can use the same link). Initially, when each message uses an independent link, several port errors are likely to occur. With each iteration, we reduce the number of port errors. When messages in the two message groups share sources or destinations, mergers will reduce the number of port errors.

When multiple message groups can be merged, we choose the merger that will reduce the number of port errors on the processors with the most errors. In the case of ties, we pick the grouping of lowest cost, and if ties persist we pick a merger at random. A grouping is feasible if, and only if, the aggregation of the messages in the two groups will not lead to deadline misses for any of the messages.

Iterative PortErr

Algorithm *PortErr* (Algorithm 1) reduces port errors by one at each step but it is not guaranteed to find a network that eliminates all port errors because links are not created between new switches. In such cases, we iteratively apply *PortErr* by treating switches added in earlier iterations as processors and making the messages to the switches part of the message set. This allows links to be created between switches. The iterations might add links between switches to eliminate port errors. Since this process may introduce multiple hops, we adjust message deadlines by distributing the slack equally over the different hops.

Figure 4 serves as an example of the *PortErr* algorithm. In this example, we assume that all switches have 4×4 ports and each processor can have up to two ports. Before any application of *PortErr*, the dashed lines indicate the messages that need to be routed through switches. At this stage, all processors have port errors because they all have at least 4 direct links (using at least 4 ports). For simplicity, one may assume that the deadlines are large enough to route all messages from a source along the same link; in this case, bandwidth available at a link, and not delay, is the main link constraint. After the first application of *PortErr*, one port error exists, indicated by the circle. A second application of *PortErr* introduces a new switch that is con-

Algorithm 1 PortErr

Require: Processor set \mathcal{P} , switch set \mathcal{S} , message set \mathcal{M} .

Let the set of message groups be $\mathcal{X} = \{\{m\} : m \in \mathcal{M}\}$

while true do

Let the set of groupings be $\mathcal{X}' = \{(X_1, X_2) : X_1, X_2 \in \mathcal{X}\}$

Delete infeasible groupings from \mathcal{X}'

Determine the network element (switch/processor) with the maximum number of port errors

Let e be the maximum number of port errors at a network element

if no groupings exist then

Break

end if

if \mathcal{X}' is not empty then

for all $X' \in \mathcal{X}'$ do

Let $e_{X'} = 0$

for all nodes with e port errors do

if the grouping X' reduces the port errors on the node under consideration then

$e_{X'} = e_{X'} + 1$

end if

end for

end for

From \mathcal{X}' , delete elements X' that do not achieve

$e_{X'} = \max_{Y \in \mathcal{X}'}(e_Y)$

Reduce e by $\max_{Y \in \mathcal{X}'}(e_Y)$

for each $X' \in \mathcal{X}'$ do

Calculate the cost of X'

end for

Retain only the lowest cost grouping(s) in \mathcal{X}'

end if

Select any grouping \hat{X} from \mathcal{X}'

if \hat{X} reduces the port error on at least one node, or the grouping reduces the cost then

Perform merger; adjust \mathcal{X} appropriately

else

Break

end if

end while

nected to a switch from the first round to eliminate the unresolved port error.

Characterizing PortErr

PortErr is a simple, iterative grouping algorithm that combines links and switches while retaining schedulability. It is heuristic-driven approach to the NP-hard design problem. In terms of run-time complexity, *PortErr* is a $O(|\mathcal{M}|^2)$ algorithm because we start with at most $\binom{|\mathcal{M}|}{2}$ possible groupings of messages and iteratively reduce the number of message groups.

The algorithm is not guaranteed to find a feasible solution. The existence of a solution depends on the allocation of tasks to processors and bad task allocations will result in infeasible solutions. Moreover, there

may be situations when *PortErr* does not find a feasible solution even when such a solution exists. For the special case of bandwidth allocation, when no deadlines are specified for the messages, the algorithm will find a feasible solution provided there is some assignment of messages to ports on each processor such that the bandwidth constraints and port constraints are not violated on any processor. To verify that this is true, one simply needs to construct a 4×4 switch using a network of 3×3 switches and then inductively prove the proposition for larger networks.

Network design and task allocation need to work together to produce efficient systems. To aid task allocation, a good way to estimate delays by assigning messages to processor ports during the task allocation process and assuming the existence of an infinite port switch for measuring delay. Since *PortErr* builds networks with interconnections between switches, the delay estimate may be multiplied by some factor, $F > 1$, (for most systems, $F = 2$ should be sufficient) to ensure that the task allocation will lead to feasible network.

The relationship between task allocation and network design creates a feedback loop in the design process and iterating over the two operations will lead to reductions in system cost. This is one of the reasons a fast algorithm for design is desirable – multiple allocations can be tested and solutions can constantly improved.

6. Evaluation

To test the effectiveness of our network design algorithm, we ran 192 different experiments using our algorithm. For each experiment, we generated a random network design problem and compared solutions if OQ and iSLIP switches were used. These experiments were divided into 16 categories (12 experiments in each category) depending on the per-link utilization, the number of periodic messages and the number of processors in the system.

For the purpose of evaluation, all links were assumed to be full duplex, capable of operating at the same speed in either direction. The capacity of each link was 1 Gbps in each direction. Since switches may vary in number of ports and the number of switches in a design may be different, we compare designs on the basis of the cost of the network. Our switch costs are based on the prices of commercially switches. For these experiments, we used switches with at most 8/16/32/64 ports. We estimated switch costs based on the number of ports in a switch: the cost per port was set at \$1600. This estimate based on statistics for the Cisco MDS 9000 family of switches (Table 2). In fact, this is a conservative estimate because \$1600 is the lowest per-port price among all the switches in the MDS 9000 family.

The link utilization metric is a measure of the bandwidth required by the messages. A high link utilization metric indicates that the average bandwidth required by a processor that needs to communicate is 75 percent

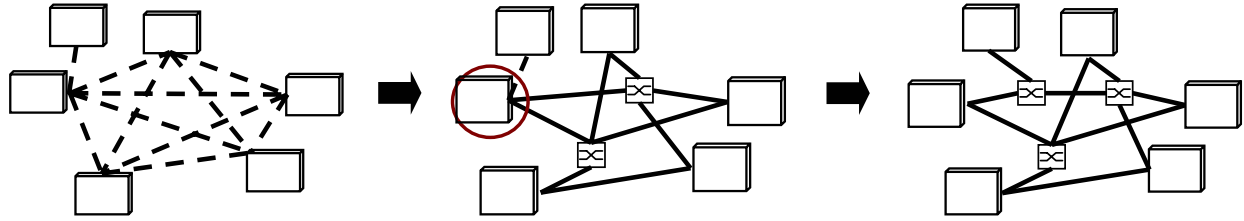


Figure 4. Illustration depicting the *PortErr* algorithm

of its output capacity. A low link utilization indicates that the average bandwidth required by a processor is 35 percent of its output capacity. (Capacities are scaled appropriately for VOQ switches that use iSLIP because we are able to guarantee delays for smaller bandwidth requirements only.)

The source and destination for each message was chosen uniformly at random from the processors. Once the set of source-destination pairs were chosen, the message sizes, periodicity and deadlines were chosen to meet the link utilization target and the feasibility requirement. The message parameters were chosen such that the communication requirements would be met assuming dedicated links. The design algorithm would then generate a network of switches that can satisfy the schedulability criteria and lower the cost of the network.

We report the average costs of the networks built and the running times of our algorithm for all cases when a feasible design was found. For comparison, we implemented a branch and bound algorithm that searches the entire architectural space for an optimal design. The optimal algorithm requires exponential time and runs in reasonable time for only small systems. We compare the costs and running times of our algorithm with the optimal branch and bound algorithm only for small-size problems. In our study, we also compared the costs of networks built using OQ switches and VOQ switches (scheduled using the iSLIP algorithm).

It is worth noting that our heuristic approach found a feasible solution for all the small problems for which we obtained optimal solutions using the branch and bound approach. For larger problems, there were few instances (5 in 192) when a feasible solution was not obtained but those cases were quickly detected and deadlines adjusted to make the instances feasible. Our primary objective was to evaluate the *PortErr* algorithm in terms of run time and cost when it found a solution.

6.1. Comparing Design Costs

In general, we discovered, not unpredictably, that OQ switches lead to less expensive designs (Figure 5). The delay bounds used for OQ switches are tighter than the bounds for VOQ switches and this is a major reason for improvements in design. One might have expected VOQ switch designs to actually be far more expensive because of the pessimism in the delay computation, but

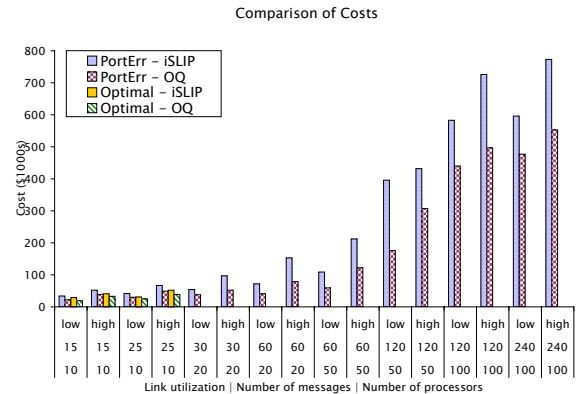


Figure 5. Comparing network costs

experimental evidence suggests that the designs are at most about 45 percent more expensive than OQ switch designs.¹ This is because we control the number of messages using a particular link, and with fewer messages through a link, the delay bounds are much better. For high link utilization, when we increased the number of messages from 120 to 240, we do not see a significant cost increase because the increase results in more messages with lower per-message link utilization, which causes only a marginal increase in system cost.

The branch and bound algorithm, which we ran only for the test cases with 10 processors, does produce designs that are better than our heuristic algorithm. We do, however, note that designs generated by our heuristic are not more than 27 to 33 percent over the optimal cost. We have not been able to prove any theoretical bounds on the performance of our approximation algorithm but the results of the experiments are encouraging.

One reason for the high costs of networks for larger systems is the inaccuracy in our estimate of the switching delay. With better delay measures, costs are likely to be lower. In all cases, the standard deviation of the network cost was not more than 20 percent of the average cost.

Remark. OQ switches, as we noted earlier, may be more expensive in comparison to VOQ switches. We

¹Very often, deployed networks are over-provisioned to the extent of being twice or more as expensive as they need to be.

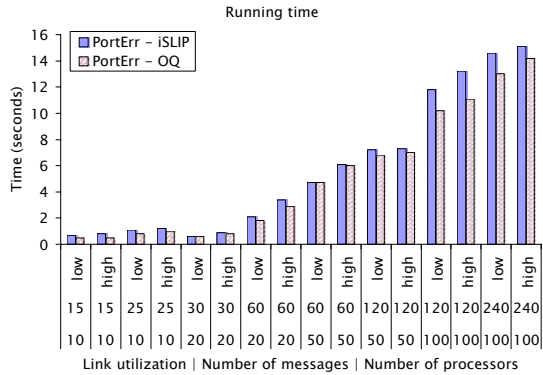


Figure 6. Comparing running times of PortErr

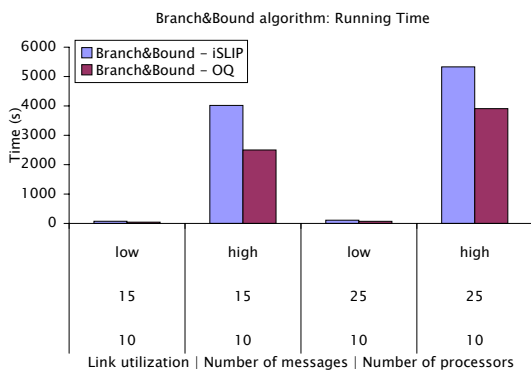


Figure 7. Runtime for the Branch&Bound algorithm

do not distinguish between the costs in our experiments, but this type of evaluation – with exact market costs – can allow architects to determine the cost boundaries for employing OQ or VOQ switches. The use of exact costs, which were not available to us, would also quantify the difference between OQ switches and VOQ switches in terms of scalability.

6.2. Comparing Design Running Times

We would like to incorporate network design as a stage in a design process for real-time systems. The speed at which designs can be generated and evaluated is an important factor in the adoption of such tool chains. Our experiments (performed on a 2GHz Pentium IV processor with 512 MB RAM) showed that our approximation algorithm is extremely efficient, taking only a few seconds for small problem instances and consuming no more than 16 seconds to solve large instances (Figure 6). On the other hand, finding an optimal solution takes several hours even for small problems (Figure 7). The best performance of the branch and bound algorithm was for small problems with low link utilizations, when solutions were found in about 100 seconds

(This number is significantly smaller than the thousands of seconds for the case of high link utilization and does not appear clearly on the graph). The reason the branch and bound algorithm takes many minutes to complete is the need to test designs with many switches: with higher utilizations, multiple switches may be required and since the number of switches is not known ahead of time, the number of switches and types of switches have to be changed and each configuration tested.

7. Improving Predictability

Our analysis and experiments indicate that switching algorithms which provide higher throughput may not be suitable for real-time systems because of their unpredictability. The high variance in data transfer with iSLIP switches motivates further work in designing switches and routers for real-time systems.

Rexford, Hall and Shin have presented a router architecture for real-time communication [25]. Such designs are hard to scale to high data rates. On the other hand, we speculate that minor modifications to algorithms like iSLIP, while resulting in lower throughputs, will increase predictability and reduce the network cost. For instance, iSLIP provides time-division multiplexing under uniform traffic and high loads. If the hardware design was altered slightly to mimic uniform traffic, then each connection would be guaranteed. Such an implementation would be trivial to implement when compared to iSLIP’s request-grant-accept handshake. This modification would drastically cut down the pessimism in worst-case delay – for transmitting one cell that has reached the head of the queue, the delay would be at most N cell slots and not N^2 cell slots.

8. Related Work

Obtaining average delay bounds for high-speed switches has been well-explored beginning with early work by Karol et al. [16] and more recently by Shah and Kopikare [28] and Leonardi et al. [18]. The search for deterministic bounds has been harder, and in systems with limited (or no) prioritization, we have attempted to obtain some gross bounds to aid system designers. There has been some thrust with the real-time systems community to analyze delays in some networks; for instance, Li, Raha and Zhao [19] have developed delay bounds in stable ATM networks. Parts of our work are also related to Cruz’s delay calculus [11, 12]. We distinguish our work from prior research by focussing on modern high-speed switches and by tackling the network design problem.

The area of network topology design has also been extensively researched from the early days of the Internet [15]. Many techniques have been proposed for centralized networks that required tree topologies [13]. In distributed systems, no efficient and exact solutions have been found for the minimum cost topology de-

sign problem. Heuristics such as the Branch and Exchange [31] and Concave Branch Elimination [36] have been proposed and improved using Cut-Saturation [3] or Concave Link Elimination [30]. These techniques do not, however, take a higher-level view and do not deal with switch and port configurations.

An approach to topology design based on source-destination load statistics was proposed by Qiao and Ni [24]. They used block designs to develop an initial topology and propose schemes for tuning network behavior. The overall goal is not to minimize network cost but to reduce path lengths. Similarly, most of the earlier work in network design focus on path lengths or bandwidth and not on overall network cost due to switching and routing elements. Other work in the area of network design, such as access network design [8] and storage area network design (Minerva [7] and Appia [33]), tackle related problems and develop similar solution methodologies. In comparison to these research projects, our work stresses latency regulation (as opposed to bandwidth allocation) and a more careful understanding of switch scheduling mechanisms. Also, unlike traditional hardware/software co-design techniques for real-time systems such as those proposed by Wolf [34, 35] where buses are employed, we reason about high-speed switches and the challenges they pose.

9. Conclusion

Obtaining tight deterministic bounds for high-speed switch scheduling algorithms is important given their wide-spread deployment in distributed real-time systems. However, many switching algorithms are not amenable to simple analysis for most common situations. Although mean delays can be obtained for message delays, upper bounds have been hard to compute. We presented some upper bounds for message delay across some common switches, which, albeit being pessimistic, provide a starting point for the design of networks for real-time systems.

Using the obtained bounds, we presented a heuristic for the NP-hard network design problem. The goal of the design process is to create a network topology that can satisfy the communication requirements of the system while minimizing network cost. Our heuristic performs admirably for many test problems. For small problems, where we were able to obtain optimal solutions using a branch and bound algorithm, the heuristic approach is not more than about 30 percent away from optimality.

Based on the experiments we have conducted, we can conclude that greater predictability leads to considerable cost savings and gains in bandwidth that may be attained by using scheduling algorithms that are hard to predict are not beneficial when the worst-case delay needs to be bounded. iSLIP, which was designed for VOQ switches running at higher speeds, is more difficult to predict and results in systems that are more

expensive and operate at low utilization. On the other hand, even when OQ switches operate at slightly slower speeds, their greater predictability implies that we can utilize the bandwidth more effectively and obtain lower cost networks. Though it is true that OQ switches do not scale very well, switches that use combined input/output queueing (CIOQ) can emulate the performance of OQ switches exactly (by running the switch fabric with small speedups) [9] and scale to greater bandwidths and port counts. The increased deployment of such switches will allow us to utilize the results for OQ switches without any change. We also note that making small modifications to switch scheduling algorithms such as iSLIP may reduce the achievable throughput but provide greater predictability which will result in cost savings for safety-critical applications.

The actual technique for constructing the network is independent of the delay analysis performed and better analysis methods can be incorporated easily. Our future work includes strengthening the bounds on delay given predictable traffic patterns and to incorporate a limited number of priority levels. Another extension to this work is the design of survivable networks that can tolerate a finite number of failures.

We believe that our solution will aid system architects – our algorithm is fast and can be used to test multiple system configurations quickly. As a part of a tool chain, we expect such techniques to become integral to the design process. Further, with many systems moving towards COTS components, we expect that work such as ours will help bridge the *system integration gap* when devices not originally intended for real-time applications are used in real-time systems.

References

- [1] M-series platform datasheet. Juniper Networks, Sunnyvale, California. Available online: <http://www.juniper.net/products/mseries/100042.html>.
- [2] Performing Internet routing and switching at Gigabit speeds. GSR 12000 Technical Product Description, Cisco Systems, San Jose, California. Available online: <http://www.cisco.com/warp/public/733/12000/index.shtml>.
- [3] Issues on large network design. Network Analysis Corporation, ARPA Report, 1974.
- [4] Avionics architecture definition, version 1.0. Joint Advanced Strike Technology Program, 1994.
- [5] ATMS2000: 5 Gb/sec switch engine chipset. MMC Networks, Sunnyvale, California, 1996. Available online: <http://mmcnet.com/Solutions/ATMS2000.asp>.
- [6] AJMONE MARSAN, M., BIANCO, A., LEONARDI, E., AND MILIA, L. RPA: A flexible scheduling algorithm for input buffered switches. *IEEE Transactions on Communication* 47, 12 (Dec. 1999), 1921–1933.
- [7] ALVAREZ, G., BOROWSKY, E., GO, S., ROMER, T., BECKER-SZENDY, R., GOLDING, R., MERCHANT, A., SPASOJEVIC, M., VEITICH, A., AND WILKES, J. Minerva: an automated resource provisioning tool for large-scale storage systems. *ACM Transactions on Computer Systems* 19, 4 (Nov. 2001), 483–518.

- [8] ANDREWS, M., AND ZHANG, L. The access network design problem. In *Proceedings of the Annual Symposium on Foundations of Computer Science* (Nov. 2003), pp. 40–49.
- [9] CHUANG, S.-T., GOEL, A., MCKEOWN, N., AND PRABHAKAR, B. Matching output queueing with a combined input output queued switch. *IEEE Journal on Selected Areas in Communication* 17, 6 (Dec. 1999), 1030–1039.
- [10] CISCO SYSTEMS, INC., SAN JOSE, CALIFORNIA. *Cisco MDS 9000 Multilayer Switches Reference Guide*, 2005. <http://www.cisco.com/>.
- [11] CRUZ, R. L. A calculus for network delay, part I: Network elements in isolation. *IEEE Transactions on Information Theory* 37, 1 (Jan. 1991), 114–131.
- [12] CRUZ, R. L. A calculus for network delay, part II: Network analysis. *IEEE Transactions on Information Theory* 37, 1 (Jan. 1991), 132–141.
- [13] FRANK, H., FRISCH, I., CHOU, W., AND SLYKE, R. Optimal design of centralized computer networks. *Networks* 1 (1971), 43–57.
- [14] GAREY, M. R., AND JOHNSON, D. S. *Computers and intractability: a guide to the theory of NP-completeness*. W. H. Freeman, 1979.
- [15] GERLA, M., AND KLEINROCK, L. On the topological design of distributed computer networks. *IEEE Transactions on Communication* 25 (1977), 48–60.
- [16] KAROL, M., HLUCHYJ, M., AND MORGAN, S. Input versus output queueing on a space division switch. *IEEE Transactions on Communication*, 12 (Dec. 1987), 1347–1356.
- [17] KESLASSY, I., ZHANG-SHEN, R., AND MCKEOWN, N. Maximum size matching is unstable for any packet switch. *IEEE Communication Letters* 7, 10 (Oct. 2003), 496–498.
- [18] LEONARDI, E., MELLIA, M., NERI, F., AND AJMONE MARSAN, M. Bounds on delays and queue lengths in input-queued switches. *Journal of the ACM* 50, 4 (July 2003), 520–550.
- [19] LI, C., RAHA, A., AND ZHAO, W. Stability in ATM networks. In *Proceedings of IEEE Infocom* (1997).
- [20] MCKEOWN, N. iSLIP: A scheduling algorithm for input-queued switches. *IEEE Transactions on Networking* 7, 2 (Apr. 1999).
- [21] MCKEOWN, N., IZZARD, M., MEKKITTIKUL, A., ELLERSICK, B., AND HOROWITZ, M. The Tiny Tera: a small high-bandwidth packet switch core. In *Proceedings of Hot Interconnects IV* (Aug. 1996), pp. 161–173.
- [22] MCKEOWN, N., AND MEKKITTIKUL, A. A practical scheduling algorithm to achieve 100% throughput in input-queued switches. In *Proceedings of IEEE Infocom* (1998), pp. 792–799.
- [23] MCKEOWN, N., MEKKITTIKUL, A., ANANTHARAM, V., AND WALRAND, J. Achieving 100% throughput in an input-queued switch. *IEEE Transactions on Communications* 47, 8 (Aug. 1999), 1260–1272.
- [24] QIAO, W., AND NI, L. Network planning and tuning in switch-based LANs. In *Proceedings of the International Conference on Parallel Processing* (1998), pp. 287–294.
- [25] REXFORD, J., HALL, J., AND SHIN, K. G. A router architecture for real-time communication in multicomputer networks. *IEEE Transactions on Computers* 47, 10 (Oct 1998), 1215–1227.
- [26] RIDDEL, D. Infiniband as an avionics net: a proposal. *COTS Journal* (Oct. 2003), 85–89.
- [27] SHA, L., RAJKUMAR, R., AND LEHOCZKY, J. P. Real-time scheduling support in Futurebus+. In *Proceedings of the IEEE Real-Time Systems Symposium* (1990), pp. 331–340.
- [28] SHAH, D., AND KOPIKARE, M. Delay bounds for approximate maximum weight matching algorithms for input queued switches. In *Proceedings of IEEE Infocom* (2002), pp. 1024–1031.
- [29] SPANGLER, T. HP moves hard on Cisco. Byte and Switch, June 2003. Available online: http://www.byteandswitch.com/document.asp?doc_id=35013.
- [30] STACEY, C., EYERS, T., AND ANIDO, G. A concave link elimination (CLE) procedure and lower bound for concave topology, capacity and flow assignment network design problems. *Telecommunication Systems* 13 (2000), 351–371.
- [31] STEIGLITZ, K., WEINER, P., AND KLEITMAN, D. J. The design of minimum cost survivable networks. *IEEE Transactions on Circuit Theory* (Nov. 1969), 455–460.
- [32] TARJAN, R. E. *Data structures and network algorithms*. Society for Industrial and Applied Mathematics, 1983.
- [33] WARD, J., O’SULLIVAN, M., SHAHOUMIAN, T., AND WILKES, J. Appia: automatic storage area network design. In *Proceedings of the Conference on File and Storage Technologies* (Jan. 2002), pp. 203–217.
- [34] WOLF, W. Object-oriented cosynthesis of distributed embedded systems. *ACM Transactions on Design Automation of Electronic Systems* 1, 3 (July 1996), 301–314.
- [35] WOLF, W. *Readings in hardware/software co-design*. The Morgan Kaufmann Systems On Silicon Series. Kluwer Academic Publishers, 2001, ch. An architectural co-synthesis algorithm for distributed, embedded computing systems, pp. 338–349.
- [36] YAGED, B. Minimum cost routing for static network models. *Networks* 1 (1971), 139–172.