Halmstad University Post-Print

Switched real-time Ethernet in industrial applications - deadline partitioning

Hoai Hoang and Magnus Jonsson

N.B.: When citing this work, cite the original article.

©2003 IEEE. Personal use of this material is permitted. However, permission to reprint/republish this material for advertising or promotional purposes or for creating new collective works for resale or redistribution to servers or lists, or to reuse any copyrighted component of this work in other works must be obtained from the IEEE.

Hoang H, Jonsson M. Switched real-time Ethernet in industrial applications - deadline partitioning. In: APCC2003: the 9th Asia-Pacific Conference on Communications in conjunction with 6th Malaysia International Conference on Communications (MICCO3) : 21-24 September 2003, City Bayview Hotel, Penang, Malaysia. IEEE; 2003. p. 76-81.

DOI: <http://dx.doi.org/10.1109/APCC.2003.1274315>

Copyright: IEEE

Post-Print available at: Halmstad University DiVA
<http://urn.kb.se/resolve?urn=urn:nbn:se:hh:diva-396>

Switched Real-Time Ethernet in Industrial Applications - Deadline Partitioning

Hoai Hoang and Magnus Jonsson.

School of Information Science, Computer and Electrical Engineering, Halmstad University, Halmstad, Sweden, Box 823, S-301 18, Sweden. {Hoai.Hoang, Magnus.Jonsson}@ide.hh.se, <http://www.hh.se/ide>

Abstract

This paper presents work on a switched Ethernet network extended to allow for periodic real-time traffic, using earliest deadline first (EDF) scheduling. A scheme of asymmetrically dividing deadlines of real-time channels between the in and outgoing links to/from the switch is proposed (ADPS). The scheme is compared with the simpler approach of dividing the deadlines in two (SDPS). The results of several software simulations of setting up RT channels over a full-duplex switched Ethernet network are presented. The simulations show that the ADPS performs better than the SDPS when master-slave communication is assumed over the network.

1 Introduction

An important trend in the networking community is to involve more switches in the networks (e.g., LAN, Local Area Networks) and a pure switched-based network becomes more and more common. At the same time, the industrial communication community has a strong will to adapt LAN technology (e.g. Ethernet) for use in industrial systems. The involvement of switches does not only increase the performance; the possibility to offer real-time services is also improved. Now, when the cost of LAN switches has reached the level where pure switched-based networks have become affordable, the collision possibility in IEEE 802.3 (Ethernet) networks can be eliminated and methods to support real-time services can be implemented in the switches without changing the underlying widespread protocol standard.

Several protocols to support real-time communication over shared-medium Ethernet have been proposed [1] [2] [3]. However, these protocols are either changing the Ethernet standard or do not add guaranteed real-time services. Real-time communication over switched Ethernet has also been proposed (called EtheReal) [4]. The goal of the EtheReal project was to build a scaleable real-time Ethernet switch, which support bit rate reservation and guarantee over a switch without any hardware modification of the end-nodes. EtheReal is throughput oriented which means that there is no or limited support for hard real-time communication, it has no explicit support for periodic traffic so it is not suitable for industrial applications. A review of research on real-time guarantees in packet-switched networks is found in [5].

This paper presents work on a previously proposed full duplex switched Ethernet network with support for both bit

rate and timing guarantees for periodic traffic [6]. Only a thin layer is needed between the Ethernet protocols and the TCP/IP suite in the end-stations. The switch is responsible for admission control, while both end-stations and the switch have EDF (Earliest Deadline First) scheduling [7]. The deadlines of messages over the network are end-to-end based, insofar as it is the maximum time to deliver, from the release time in the source node, to the arrival in the destination.

In this paper, we assume a single switch, with one node connected to each physical port. The messages originating from the source do therefore traverse two links, and we need to provide guarantees for the time to deliver over both links. We approach this problem by dividing the end-to-end deadline into two, one for the source to the switch, and one from the switch to the destination. The deadline can be partitioned in a number of ways. The method we choose affects the system. The paper is concerned with analyzing the partitioning of deadlines, and to propose a way that is more suitable for master slave communication, which is a common demand in industrial applications.

The results, and indeed the method in its current form, do not refer to a mixed topology. The network topology is confined to a star, with one centralized switch connected to one node on each physical port. A full-duplex network is assumed.

The rest of the paper is organized as follows. The network architecture and real-time traffic handling are presented in Section 2. In Section 3, a feasibility analysis is introduced. In section 4, two deadline partitioning schemes (DPS) are presented: the Symmetric Deadline Partitioning Scheme (SDPS) and the Asymmetric Deadline Partitioning Scheme (ADPS). In Section 5, practical considerations are discussed. In Section 6, a simulation is presented. The paper is concluded in Section 7.

2 Network architecture and traffic handling

We consider an example of a network with full-duplex switched Ethernet. The end-nodes are occasionally referred to as master nodes and slave nodes. This is only to reflect typical industrial networks and is not a system requirement. In other words, the network can be used for arbitrary traffic situations. Both the switch and the nodes have added software (a thin RT layer) to support guarantees for real-time traffic. All nodes are connected to the switch and master nodes can communicate with slave nodes over logical real-time channels (RT channels), each being a virtual connection between two nodes in the system (Figure

Figure 1: Network configuration, master-slaver traffic pattern with RT channels

1). Each master node is responsible for a number of slave nodes and RT channels carry traffic from master nodes to slave nodes. Depending on what nodes are connected to the switch, and what channels are present, we can see a difference of System State. We define the System State (SS) as follows:

$$SS = \{N, K\} \quad (1)$$

where N is the set of nodes in the system and K is the set of channels running on the system at present.

The network supports dynamic adding of RT channels to guarantee periodic real-time traffic. An RT channel with index i is characterized by:

$$\{Source_i, Destination_i, P_i, C_i, d_i\} \quad (2)$$

where P_i is the period of data, C_i is the amount of data per period, and d_i is the relative deadline used for the end-to-end EDF scheduling. P_i , C_i , and d_i are expressed as the number of maximal sized frames, i.e., the number of T_{frame} . $Source_i$ and $Destination_i$ are Ethernet MAC addresses for channel i 's source and destination nodes respectively. It is evident that both $Source_i$ and $Destination_i$ are parts of the system, i.e. $Source_i \in N$; $Destination_i \in N$.

In our network configuration, both the switch and the end-nodes use the Earliest Deadline First (EDF) algorithm for traffic control. The switch is responsible for admission control, MAC functions, frame buffering and traffic scheduling. The switch periodically sends synchronization frames to the end-nodes at an interval, T_{cycle} , of ten maximum sized frames, T_{frame} , i.e.,

$$T_{cycle} = 10T_{frame} \quad (3)$$

Figure 2: Layers and output queues.

Figure 3: Data frame sent over an RT channel.

In this way, every node has a uniform comprehension about global time, with the resolution of T_{frame} . In this paper, we assume Fast Ethernet (100 Mbit/s) with a maximum frame size of 12144 bits which, with some extra time for timing uncertainties and for simplicity, gives $T_{frame} = 125 \mu s$, which just happens to match the time resolution of many telecommunication systems.

The function of and interaction with the RT layer etc shown in Figure 2 is explained below. When an application wants to setup an RT channel, it interacts directly with the RT layer (1). The RT layer then sends a request to the RT channel management software in the switch (2). Outgoing real-time traffic from the end-node uses UDP and is put in a deadline-sorted queue in the RT layer (3). Outgoing non-real-time traffic from the end-node typically uses TCP and is put in an FCFS-sorted (First Come First Serve) queue in the RT layer (4). In the same way, there are two different output queues for each port on the switch too (5).

When a node wants to establish an RT channel, it sends a request frame (ReqF) with source and destination node MAC and IP addresses and $\{P_i, C_i, d_i\}$ to the switch. A connection ID to distinguish between several possible connection requests is also added. When receiving a ReqF, the switch will determine if the channel can be added to the system, i.e. whether the real-time guarantees can be upheld. If the switch determines the channel to be acceptable, the ReqF is forwarded to the destination node, after adding a network unique ID in the RT channel ID field. The destination node responds with a response frame (ResF) to the switch telling whether the establishment is accepted or not. The switch will then, after taking notation of the response, take necessary steps towards updating the system.

The RT layer in an end-node prepares outgoing real-time IP datagrams by changing the IP header before letting the Ethernet layers sending it (see Figure 3). The IP source address and the 16 most significant bits of the IP destination address, 48 bits together, are set to the absolute deadline of the frame. A 48 bit absolute deadline with a resolution of $T_{frame} = 125 \mu s$, gives a "life time" longer than one thousand years. The 16 least significant bits of the IP destination are set to the RT channel ID for the RT channel to which the frame belongs. The MAC destination address, is set to a special address that all nodes use for real-time traffic, while the Type of Service (ToS) field is always set to value 255. One (or both) of these fields can be used to filter out real-time frames for correct service treatment.

The switch exchanges the source and destination IP addresses and the MAC destination address of an incoming real-time frame with the correct ones (as stored in the

switch when the RT channel was established) for delivery to the final destination.

3 Feasibility Analysis

A channel must by definition traverse two physical links, one from the source to the switch, and one from the switch to the destination (hereafter denoted as upload and download respectively). For a given channel, it is required for the switch to provide guarantees for both the *uplink* and the *downlink* part.

Theoretically, there is a gain to be made by dividing the concept of a channel into two parts, upload and download. The reason is that one can then look upon each part of the channel as a periodic task, and the corresponding link would constitute a CPU or processing system (from a scheduling point of view). The capacity, C_i , would be the worst-case-execution-time (WCET) for the task. Furthermore, because the system is full duplex, each link would constitute two independent CPUs, one executing the download parts of all channels traversing the link, and the other executing the upload parts (hereafter we will refer to one full duplex link as two links; one upload and one download).

The duty of the download link is then to 'execute' the set of tasks assigned to it, in the order decided by the switch, i.e. to carry out the EDF schedule set forth by the switch. For the upload link the EtherDaemon software handles the scheduling, but otherwise the same is true.

In our system we use EDF as the scheduling algorithm, for both the switch and the end-nodes. Liu and Layland [7] showed the advantage of using EDF, as it is the optimal uniprocessor scheduling algorithm. The one big drawback of EDF is the complexity of the feasibility test, compared with other scheduling algorithms such as the rate-monotonic[10]. Before we describe the feasibility test, we make the following definitions:

- "The *Utilization factor*"

According to basic EDF theory [7] the *utilization* of periodic real-time traffic is defined as

$$U = \sum C_i / P_i \quad (4)$$

- "The *Hyperperiod*"

The *Hyperperiod* for a set of periodic tasks is defined as the length of time from when all tasks' periods start at the same time, until they start at the same time again

- "The *BusyPeriod*"

a *BusyPeriod* is any interval of time in which a link is not idle.

- "The *workload function*"

$h(n, t)$ is the sum of all the capacities of the tasks with absolute deadline less than or equal to t , running on link n , where t is the number of timeslots elapsed from the start of the hyperperiod. It is calculated as follows

$$h(n, t) = \sum_{i \in K_n} \left(1 + \left\lfloor \frac{t - d_i}{P_i} \right\rfloor \right) C_i \quad (5)$$

where K_n is the set of tasks running on link n

- A *feasible link* is a link with a set of channels traversing it that can be feasibly scheduled using EDF.
- A *feasible system state* is a system state with every link in the system being feasible.

Following the discussion from above, and the new definitions, the problem for the switch to test if the channel can be added is therefore equivalent to testing if the new state is still feasible, given that the new channel has been added. The feasibility test of a link is done in two steps, each step being a test of its own.

- (First Constraint)

The utilization of the link has to be less than or equal to one (100%)

- (Second Constraint)

For all values of t , the workload function $h(n, t)$ has to be less or equal to t

The second constraint, in the form given above, does not lend itself out particularly well to computation. It is shown in [11] how to reduce the time and memory complexity of the second constraint check. If $h(n, t) \leq t$ in the first busy period of the *hyperperiod* in the supposed schedule to come, then $h(n, t) \leq t$ for all t . The following *upperbound* would therefore be an improvement of the algorithm above:

$$t, \text{ such that } 1 \leq t \leq \text{BusyPeriod}(n) \quad (6)$$

where $\text{BusyPeriod}(n)$ is the first *BusyPeriod* in the schedule at the start of the hyper-period. Further more, one needs not check every integer from the first timeslot, but only the integers t where

$$t \in \bigcup_{i=1}^n \{mT_i + d_i : m = 0, 1, \dots\} \quad (7)$$

Deadline Partitioning

We have mentioned above the method of looking at the links as processing units, having tasks to perform. This method is devised in the interest of forcing the test of system feasibility, down to the level of successive tests on links.

For this approach to work, we need to derive two supposed tasks from each channel. A pair of supposed tasks for the upload and download part of a channel, T_{iu} and T_{id} , is defined as:

$$T_{iu} = \{\text{Source}_i, C_i, d_{iu}, P_i\} \quad (8)$$

$$T_{id} = \{\text{Destination}_i, C_i, d_{id}, P_i\} \quad (9)$$

where Source_i , Destination_i , C_i and T_i are the parameters of the channel i . d_{iu} and d_{id} are the relative deadlines for the tasks. The only new information in the tasks, compared with the channel, is the relative deadlines. Considering the relative deadlines of the tasks, d_{iu} and d_{id} , as the guaranteed worst case time to deliver from the source to the switch, and from the switch to the destination respectively, we come to the following conclusion. Creating T_{iu} and T_{id} from channel i is accomplished by partitioning the deadline of the channel into 2 parts: d_{iu} and d_{id} where

$$d_i = d_{iu} + d_{id} \quad (10)$$

$$d_{iu}, d_{id} \geq C_i ; \text{ (if } d_i \geq 2C_i \text{)} \quad (11)$$

The condition (10) must be upheld, because otherwise the channel as a whole will be different. If one divides a task into separate smaller tasks, it should follow that if the original task had a deadline, then the sum of the subtasks' deadlines must equal this larger deadline. The condition (11) should be upheld because otherwise the partitioning will automatically yield a non-EDF-feasible situation. The deadline cannot be allowed to be shorter than the capacity because the capacity is the WCET of the supposed tasks. We can also assure ourselves that if $d_i < 2C_i$ then the channel cannot, by definition, be EDF-feasible.

4 Deadline Partitioning Schemes (DPS)

We make the following definition:

- A **Deadline-Partitioning Scheme (DPS)** is a function that maps the deadlines d_i of all the channels in the system into two deadlines d_{iu}, d_{id} such that the condition (Equation 10) is upheld for each RT channel.

The Domain of the function DPS is thus all the possible system states (Equation 1). The presence of a DPS gives us the freedom to create d_{iu} and d_{id} from every channel i . In fact, the availability of a DPS is not optional, but the system cannot operate without a DPS. There are different ways of looking at DPSs, but the most mathematically satisfying one is as a multi-dimensional function. The dimension of the function is then

$$\dim = \text{size}(K) \quad (12)$$

where K is the set of channels in the system state.

We can make the DPS more agreeable as a function, by turning it into a vector field, with the range of its elements fixed between 0 and 1. To start out with, the function would not generate scalars, but it would be $\dim$ number of pairs of deadlines, $\{d_{iu}, d_{id}\}$. We now take steps to change this function. First, we normalize with the original deadline, d_i for each corresponding pair. Because of (10) this would mean that we would have pairs, ranging from 0 to 1. The output would look like:

$$\begin{aligned} Upart_i &= d_{iu} / d_i \\ Dpart_i &= d_{id} / d_i \end{aligned} \quad (13)$$

where $Upart_i$ and $Dpart_i$ are the factors of d_i to get d_{iu} and d_{id} respectively. But because of (10) we conclude that:

$$Upart_i = 1 - Dpart_i \quad (14)$$

This means that both $Upart_i$ and $Dpart_i$ contain all the information by themselves. Only one is sufficient, and thus we have:

$$Upart = \text{DPS}(SS) \quad (15)$$

$Upart$ is a $\dim$ dimensional vector, of elements ($0 < Upart < 1$). The number of different possible DPSs is infinite, and in this paper two different ones will be examined: Symmetric (SDPS) and Asymmetric (ADPS).

4.1 DPS Performance Measure

A channel that is requested by the system can fail any of the two constraints posed by the feasibility test. Firstly, if the first constraint test fails, we note that whatever generous deadline the task-set has, it will not affect the outcome. The first constraint is a test of utilization only, and the utilization is invariant of the deadline, which we note from (Equation 4). On the other hand, if it is the second constraint that fails, the choice of another DPS could have made the channel feasible. It is clear that we can note a difference of performance, in the different DPSs we can choose to divide the deadlines. We define an optimal DPS as follows:

- A DPS is **optimal** iff it allows for a maximum number of channels to be accepted.

When a task set fails the second constraint, it does so because one or more values of the workload function $h(n,t)$ exceeds the value of t ($h(n,t) > t$). The test $h(n,t) \leq t$ tells us if the task set is EDF schedulable but it lacks any levels of magnitude other than the Boolean true or false. In an effort to get a significance measure, that is, a measure of how well the task set passed the second constraint, we do the following definition:

- The "**workload margin**" (w_m) for a link is defined as the lowest value of the expression:

$$t - h(n,t), \quad t \geq \min(d_i) \quad (16)$$

where $\min(d_i)$ is the minimum of the deadlines of the tasks running on the link. The "workload margin" is negative for the links that are not feasible. For links that are feasible it is positive, and gives us the minimum differences between $h(n,t)$ and t .

4.2 Symmetric Deadline Partitioning

In the original paper [6], it was proposed to partition the deadline of the channels into two equal parts, i.e. to split it in half. Following the notation introduced above, this would imply that

$$d_{iu} = d_{id} = d_i / 2 \quad (17)$$

$$Upart_i = Dpart_i = 1/2 \quad (18)$$

We define this approach as a Symmetric DPS (SDPS). It is easily seen that condition (10) is upheld under this function. We can also note that the SDPS only depends on the $\text{size}(K)$ of the system state SS . In the view of DPSs as vector fields this means that the SDPS is a vector of $\text{size}(K)$ dimension, with each element constant, equal to 0.5. Obviously, as the SDPS doesn't take into consideration what the system looks like, we should be able to propose a better DPS.

4.3 Asymmetric Deadline Partitioning Scheme

With bottlenecks we mean links with a greater number of channels traversing them than other links. We say that bottlenecks have a higher link-load, which we define in the following manner.

- The **LinkLoad (LL)** of a link is the number of channels traversing it which is the same as the number of tasks running on the link.

The parameters of the channels can also be taken into account when calculating the LL but this is not treated here.

A logical approach in the case of a bottleneck is for the system to partition deadlines of the channels that traverse the bottleneck, so that as much of the deadlines of the channels can be found in the tasks of the bottleneck. Obviously, the SDPS does not do anything to relieve bottlenecks, as the SDPS, as stated above, is invariant of the System State.

The ADPS is a DPS devised to distribute, when possible, the deadline of channels, to where it is most needed, i.e. where the LL is greatest. We define $ADPS(SystemState) (ss)$ as:

$$Up_{part_i} = LL(Source_i) / (LL(Source_i) + LL(Destination_i)) \quad (19)$$

$$D_{part_i} = LL(Destination_i) / (LL(Source_i) + LL(Destination_i)) \quad (20)$$

5 Practical Considerations

So far, we have only discussed DPSs in terms of their performance. There are other things to consider besides how well the DPS performs, however. We will address three issues: (a) memory and time complexity, (b) the need for node-feedback, (c) the need for synchronous deadline updating.

The most obvious consideration is the first item (a), computational speed and memory requirements. As we go from SDPS to ADPS we gain in DPS performance, but we lose time for calculating the new elements of the algorithms and we need more memory to keep track of data structures required implementing them (The calculations are, however, only made when a new channel is requested).

The second and third considerations deal with whether the DPS is actually implementable or not. A DPS, as we recall, has full freedom to divide the deadlines of the entire system - in reality the switch may not have this freedom. After the DPS has operated on the deadlines, the feasibility test is performed on the tasks we create from the partitioned deadlines. It is the switch that does the actual scheduling of packets for the download links, but for the upload parts, it is the various nodes in the system that performs scheduling. For the SDPS we can note a definite advantage, and that is the following. For the actual scheduling of packets, the nodes need only to look at the channels relative deadline. They do not need an updated version of a d_{iu} for each channel, because under SDPS, the system always generates the same Up_{part_i} , namely 0.5. A node can get relative deadlines of all its tasks by multiplying the d_i of the corresponding channel with 0.5. For ADPS, we need to be certain that all nodes actually have the partitioned deadline for each channel that we assume when doing the feasibility test (node-feedback). We also need to have guarantees that the nodes update their relative deadlines at exactly the same timeslot as the switch

(synchronous deadline updating). We cannot have a system where condition (10) does not hold at all times. The following steps describe the operation of adding a channel to the system:

1. Create a system state, with the new channel added.
2. Apply DPS to the system state to get deadlines.
3. Test feasibility of all task-sets of the links.
4. Make sure that all nodes are updated with the new deadlines that the DPS provided.
5. Make sure that all nodes update their relative deadlines synchronous with the switch
6. Permit traffic over the new channel

No other channel can be tested while the switch software is occupied with any of these steps. The items we want to put emphasis on are step 4 and 5. How do we let the nodes know of the update that they need to do, and how do we synchronize it with the Switch? Because the nodes are assumed to be independent of each other, save for the connection to the switch, the only way of providing the nodes of the update information would be to somehow send it over the links.

Unfortunately we cannot simply send a frame over the links as soon as we want to update the nodes, as this would destroy the guarantees of all of the other channels. However, we do have the sync frames that need to be sent every T_{cycle} (3) anyway. Depending on how much free space we have in the synchronization frame, a finite number of tasks in the nodes can be updated each T_{cycle} . To assure that the nodes actually update their upload tasks at the same time as the switch updates its download tasks, a pre-determined time lag of a few time slots should be implemented for both the node and the switch, counting from when the node receives the synchronization frame with the updating information.

6 Simulation Analysis

The simulation analysis presented here shows the performance difference between using the EDF scheduling with Symmetric and Asymmetric deadline partitioning in the network. We simulated a network with a single 100 Mbit/s full-duplex Ethernet switch, M master nodes, S slave nodes, and sets of RT channels. Each RT channel is randomly generated with uniformly distributed source (master node) and destination (slave node). As described in Section 3, the feasibility test of a link is done in two steps, the first constrain and the second constrain.

In Figure 4, the number of accepted channel is showed for a network configuration of 10 master node and 50 slave nodes. To compare the result between symmetric and asymmetric deadline partitioning, in this simulation, every requested channel have the same parameters: $C=3$, $P=100$, $d=40$. The result proved that we get much better result with asymmetric deadline partitioning. With many requested channels the number of accepted channels is about 110, while there are only 60 accepted channel with symmetric deadline partitioning. It means that with master-slave

Figure 4: Number of accepted channels when all requested channels have the same characteristics $C=3$, $P=100$, $d=40$

traffic communication, EDF scheduling with asymmetric deadline partitioning is much better than symmetric deadline partitioning.

Figure 5 (for symmetric deadline partitioning) and Figure 6 (for asymmetric deadline partitioning) show the different results when the number of master nodes is varied between 12, 15 and 20, the number of slave nodes is 60. Periods are randomly generated in a range from 80 to 120, while deadlines range from 30 to 50. We can see the different result when each master node has responsibility for 3, 4 or 5 slave node. In these cases, all the deadline d_i and period P_i are generated randomly. The acceptance ratio is still better when we use ADPS than DPS.

7 Conclusions

In this paper, we have presented an Ethernet network with support for real-time traffic by EDF scheduling. Two Deadline Partitioning Schemes have been proposed, and an effort to mathematically model the workings of DPSs has been described. Two DPSS, SDPS and ADPS have been compared. Clearly, As we go from SDPS to ADPS, we gain in DPS performance. The simulation shows that ADPS is especially suitable for a network with a traffic pattern that generates bottlenecks, of which the Master-Slave pattern is the most relevant.

Figure 6: Number of accepted channels with ADPS. Number of master nodes is variable while number of slave nodes is constant.

Figure 5: Number of accepted channels with SDPS. Number master nodes are variable while number of slave nodes is constant.

References

- [1] C. Venkatramani and T. S. Chiueh, "Supporting real-time traffic on Ethernet," *Proc. 15th IEEE Real-Time Systems Symposium (RTSS'94)*, pp. 282-286, 1994.
- [2] D. W. Pritty, J. R. Malone, D. N. Smeed, S. K. Banerjee, and N. L. Lawrie, "A real-time upgrade for Ethernet based factory networking," *Proc. IEEE IECON'95*, vol. 2, 1995.
- [3] S.-K. Kweon, K. G. Shin, and G. Workman, "Achieving real-time communication over Ethernet with adaptive traffic smoothing," *Proc. 6th IEEE Real-Time Technology and Applications Symposium (RTAS'2000)*, Washington, D.C., USA, 31 May - 2 June 2000, pp. 90-100.
- [4] S. Varadarajan and T. Chiueh, "EtheReal: A host-transparent real-time Fast Ethernet switch," *Proc. ICNP*, Oct. 1998.
- [5] H. Zhang, "Service disciplines for guaranteed performance service in packet switching network," *Proc. of the IEEE*, vol. 83, no. 10, Oct. 1995.
- [6] H. Hoang, M. Jonsson, U. Hagström, and A. Kallerdahl, "Switched real-time Ethernet with earliest deadline first scheduling - protocols and traffic handling," *Proc. Workshop on Parallel and Distributed Real-Time Systems (WPDRTS'2002) in conjunction with International Parallel and Distributed Processing Symposium (IPDPS'02)*, Fort Lauderdale, FL, USA, April 15-16, 2002.
- [7] C. L. Liu and J. W. Layland, "Scheduling algorithms for multiprogramming in hard real-time traffic environment," *Journal of the Association for Computing Machinery*, vol. 20, no. 1, Jan. 1973.
- [8] D. Ferrari and D. C. Verma, "A scheme for real-time channel establishment in wide-area networks," *IEEE Journal of Selected Areas in Communications*, vol. 8, no. 3, pp. 368-379, Apr. 1990.
- [9] Q. Zheng and K. G. Shin, "On the ability of establishing real-time channels in point-to-point packet-switched networks," *IEEE Transactions on Communications*, vol. 42, no. 2/3/4, pp. 1096-1105, Feb./Mar./Apr. 1994.
- [10] C.M. Krishna and K. G. Shin, *Real-Time Systems*. McGraw-hill international edition, 1997.
- [11] J. A. Stankovic, M. Spuri, K. Ramamritham, Giorgio C. Buttazzo, *Deadline Scheduling for Real-Time Systems - EDF and Related Algorithms*. Kluwer Academic Publishers, 1998.