Adding Contract-Based Reservation Services to a Hard Real-Time Ethernet Protocol

Michael González Harbour, José María Martínez, Juan López Campos,
J. Javier Gutiérrez, and Julio L. Medina

Departamento de Electrónica y Computadores, Universidad de Cantabria, 39005-Santander, SPAIN
{mgh, martinjm, lopezju, gutierjj, medinajl}@unican.es

Abstract

This paper presents the implementation of DFSF, a contract-based framework for flexible scheduling in real-time networks. The initial implementation of DFSF provides the ability to make dynamic bandwidth reservations using a distributed acceptance test that ensures the overall network schedulability. It has been implemented using the Real-Time Ethernet Protocol (RT-EP), which is a software-based token-passing Ethernet protocol for real-time applications, that does not require any modification to existing Ethernet hardware.

1. Introduction[1]

FIRST (Flexible Integrated Real-Time Systems Technology) [6] is an EU project in which a flexible scheduling framework (FSF) has been developed with the aim of combining the hard real-time guarantees that are required in many real-time systems, with more flexible timing requirements that avoid the large pessimism of the hard real-time scheduling techniques and let the system resources be fully utilized to achieve the highest possible quality of service. FSF is based on establishing service contracts that represent the complex and flexible requirements of the application, and which are managed by the underlying system.

Together with other goals, FSF is designed to support applications with requirements for distribution. The first step towards distribution in this context is the ability to support service contracts for the networks used to interconnect the different processing nodes in the system. Similar to the FSF implementation in the processors, the contracts on the network allow the application to specify its minimum utilization (bandwidth) requirements, so that the implementation can make guarantees or reservations for that minimum utilization. The distributed part of FSF is called DFSF.

1. This work has been funded in part by the *Ministerio de Educación y Ciencia* of the Spanish Government under grant number TIC2002-04123-C03-02 (TRECOM), and by the IST Programme of the European Commission under project IST-2001-34820 (FIRST).

This paper presents an overview of the DFSF services and their application program interface (API), and describes the implementation that has been made using the Real-Time Ethernet Protocol (RT-EP) [4] which is a software-based token-passing protocol in a bus that uses fixed-priority scheduling, and does not require any modification to existing Ethernet hardware. The implementation has been added to an FSF system running on top of MaRTE OS [1], which is a real-time kernel on which our research group has been working in the last few years.

2. Distributed Flexible Scheduling

One of the key elements of the FSF API is the *contract*, which is an abstract data structure used by the application to specify its timing and flexible scheduling requirements. The contract has many attributes such as those required to specify how the application can make use of any spare capacity that the system may give to it, or the notification mechanisms for deadline misses or execution budget overruns, for instance. Table 1 shows the main attributes that are relevant to the network contracts. The minimum budget and maximum period allow specifying a guaranteed bandwidth reservation.

Table 1. Main contract attributes

Name	Description
minimum budget	Minimum execution capacity per server period
maximum period	Maximum server period
network id	Identifies the network for which the contract is negotiated; if null, the contract is negotiated on a processing node
deadline	The deadline of the server
D=T	Whether the server's deadline is equal to the period or not

Once the contract has been prepared by the application, it has to be negotiated. During the negotiation the system checks that it has enough resources to guarantee the minimum requirements specified in the new contract, together with all the contracts currently accepted in the system. If so, it accepts the new contract creating a server for it and readjusts the distribution of any spare capacity that may have been assigned to the current contracts.

The server is a software entity that keeps track of the consumed resources and effectively limits the amount of budget consumed both for processors and networks. In the processors, one or more threads may be bound to a server; its execution budget is consumed whenever one of those threads is running. The deadline specified in the contract is used by the system to set the degree of urgency of each server, with the guarantee that the specified budget will be provided before the deadline in each server period, as long as the application requested execution for that budget at the beginning of the server period.

As an abstract framework, FSF can be implemented on top of different scheduling strategies. For instance, it has been implemented in SHaRK with an EDF-based scheduler and using Constant Bandwidth Servers as the underlying server mechanism [8], and also in MaRTE OS with a fixed priority scheduler and using sporadic servers for implementing the FSF servers [7]. Since these implementations share the same API, applications can use FSF in a totally platform-independent manner.

For the DFSF implementation to keep track of consumed network resources and to enforce the budget guarantees it is necessary that the information is sent and received through specific DFSF services. To provide communication in this context we need to create objects similar to the sockets used in most operating systems to provide message communication services. We call these objects *communication endpoints*, and we distinguish send and receive endpoints.

A *send endpoint* contains information about the network to use, the destination node, and the port that identifies a receive endpoint. It is bound to an *FSF network server* that contains the scheduling parameters of the messages sent through that endpoint, keeps track of the resources consumed, and limits the bandwidth to the amount reserved for it by the system. It provides message buffering for storing messages that need to be sent.

A *receive endpoint* contains information about the network and port number to use. It provides message buffering for storing the received messages until they are retrieved by the application, either with a blocking or non blocking message receive operation. A receive endpoint may get messages sent from different send endpoints, located in the same or in different processing nodes.

The main operations of the FSF API involved in the network contracts are the following:

- initialize a contract
- set contract attributes (see Table 1)
- negotiate a contract, obtaining a network server
- create a send endpoint
- bind a network server to a send endpoint
- send a message through a send endpoint
- create a receive endpoint
- receive a message from a receive endpoint

The connection between a send and a receive endpoint is established by specifying at the creation of both the network id and the port number to use, and by specifying the destination node in the sender's end. Figure 1 shows the main elements involved in the communication with DFSF. To establish a communication path the application has to select a network and choose a port to be used in it. At the sending end it creates a send endpoint with that network id and port and with the destination node, initializes a contract, and sets its attributes specifying its timing requirements related to the communication. It then negotiates the contract. If accepted, a server id is returned, and it is used to bind the associated server with the send endpoint. At the receiver's end, a receive endpoint is created specifying the same network id and port. The budget consumption management is implemented in the server, which is bound to the send endpoint.

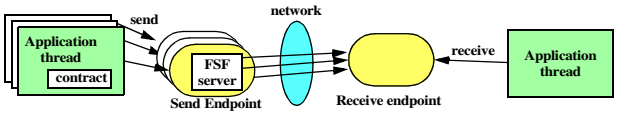

Figure 1. Communication elements in DFSF

Once the communication path has been established it may be used to send and receive messages though the associated operations in the API. The FSF network server will keep track of any consumed budget, in order to guarantee the bandwidth reservations.

3. Implementation

A major subset of the DFSF framework has been implemented on top of the RT-EP real-time ethernet protocol [4]. Two important functionalities had to be added to the protocol. On the one hand, the ability to manage budget-based servers, and on the other hand the distributed contract negotiations. For this initial implementation we have not implemented the spare capacity facilities defined in FSF.

To manage communication budgets, the underlying scheduling mechanism of RT-EP, which is fixed priorities,

was changed into the sporadic server scheduling policy [5]. Under this policy, each server is assigned a priority, a transmission capacity (measured in network packets) and a replenishment policy. The priority is set according to the server's deadline, the capacity is initialized to the minimum budget, and the replenishment period is set equal to the server's period. When the capacity is larger than zero, packets can be sent in the RT-EP network at the server's priority; each packet sent decreases the capacity by one and enqueues a replenishment operation to occur one replenishment period later. When the capacity is zero, packets can only be sent at a background priority level. When a replenishment operation is due, it consists of increasing the current server's capacity.

In order to make distributed contract negotiations, the information with the contract information is replicated in all the nodes. Negotiation operations are mutually exclusive so that the results of negotiation are consistent. This is accomplished with a special negotiation token that has been created. The results of each negotiation operation are propagated to all the other nodes so that they can update the contract information. The negotiation token is only released for a new negotiation once the operation has been completed and its results have been propagated to all the nodes.

The information related to the state of the negotiation, the negotiation token, and the results of a negotiation are transmitted with the RT-EP packets, in a way completely independent of the RT-EP behavior. In this way we do not affect the RT-EP operation, and we can take advantage of the error detection and recovery features in a transparent way.

The implementation of DFSF has been made in Ada, although both C and Ada interfaces are provided. Three software modules have been used:

- *DFSF.Shared_Info*: This package contains a protected object (that provides mutual exclusion) with the information about the contracts that is shared among all the nodes. The information allows a node to negotiate a new contract, or renegotiate a previous one.
- *DFSF.Server*s: This package contains a protected object that stores the information relative to a server that is local to the node where it is created, and therefore needs not be shared among the different nodes. The most important piece of this information is the current budget of each server.
- *DFSF.Negotiation*: This package contains a protected object that implements the state machine associated with the contract negotiations, and all the associated information. To negotiate a contract there is a negotiation token that must be acquired to ensure mutual exclusion. Once the negotiation is finished the information must be propagated to all the other nodes. The information associated with these operations is circulated in the token ring, and this package manages it as a function of the current negotiation state of each node.

Figure 2 shows the negotiation state diagram. The node starting the negotiation successively switches from the *Idle* state to *Acquiring_Negotiation_Token*, *Negotiating*, and *Waiting_To_Release_Token*. Meanwhile, the nodes not making the negotiation, switch to the *Waiting* state, where they await the information with the negotiation result.

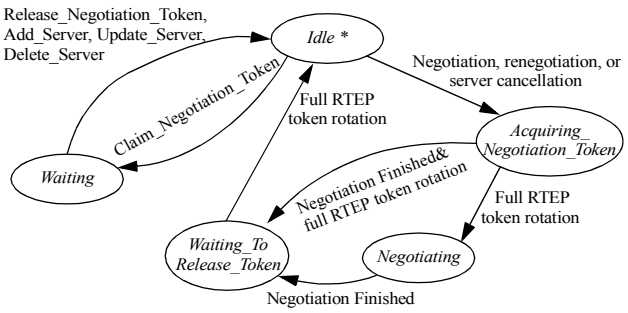

Figure 2. Negotiation state diagram

The token-passing behavior of RT-EP eases the propagation of the negotiation token and contract information to all the nodes. In a different network this propagation could be achieved by explicit broadcast or with regular protocol messages.

4. Evaluation

DFSF has been implemented on top of RT-EP, on MaRTE OS platforms, and tested in different configurations with a single 100 Mbits/s ethernet. In this section we show some of the evaluation results that were obtained.

Figure 3 shows the time it takes to make a negotiation in a system with four processing nodes, in which all the nego-

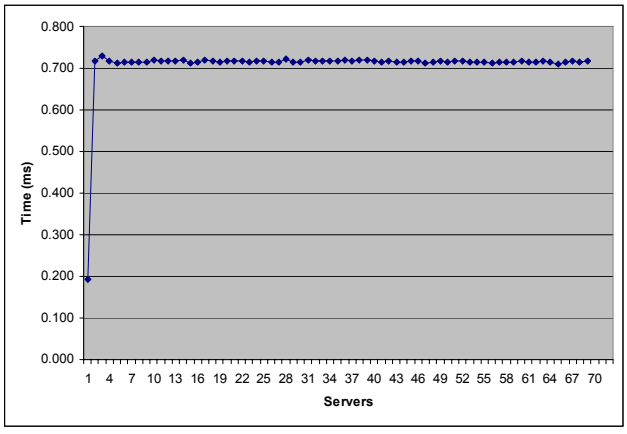

Figure 3. Network negotiation times

tiations are made by a single node, with delays between negotiations. The contracts specify a network utilization of 1%, and have deadlines equal to periods. The admission test in this case is a plain rate monotonic utilization test, which gives constant time in relation to the number of servers, and has a utilization bound of 69%. Of course, other more elaborate admission tests may be used. Other evaluation results show that the negotiation times increase when they are made within a small interval, because during the time it takes to propagate the results of the negotiations to the other nodes, the negotiation token is not available.

Table 2 contains the results regarding the transmission times in DFSF. Because we do not have synchronized clocks we measure the time it takes to send a message of maximum size to a remote node and get a similar message back. We can see that there is a difference in the worst case measured times depending on whether there is contention with lower priority messages or not. The reason is that in the latter case the non preemptability of network packets imposes a bounded blocking effect on the higher priority messages.

Table 2. Transmission times using DFSF

Operation	Worst (ms)	Avg (ms)	Best (ms)
Send a message to a remote node, execute a handler there, and receive a reply message, no contention	1.398	1.127	1.111
The same, with contention with other lower priority messages	2.517	1.211	1.157

The overhead for the new information sent in the network packets is relatively small: just 29 bytes. This makes the total packet overhead still less than the minimum ethernet packet size, and represents only 2.39 μs on a 100Mbit/s ethernet. It has to be compared with the 1492 bytes of a maximum packet size, or with transmission times in the range of 0.5 ms, to see that it represents a very small overhead.

The effort of implementing DFSF on top of RT-EP has been moderate, requiring 2093 lines of code for the actual implementation, plus adding 120 lines of code to RT-EP.

The implementation of DFSF has been successfully used to implement the distributed control system of an industrial robot. The original software based on fixed priorities was easily migrated to the new framework by just adding the contract information at the application layer, and modifying the middleware layer [2][3] to use the DFSF API.

5. Conclusions

In this paper we have presented DFSF, a flexible network scheduling framework. Applications can encapsulate their flexible timing requirements inside a contract that is negotiated with the system. The implementation of DFSF includes a distributed negotiation mechanism as well as a new scheduler that manages budget consumption and enforces the bandwidth reservations.

The implementation has proved to be feasible, efficient, and easy to use. It has been successfully used to implement a distributed controller for an industrial robot.

For the future, we plan to introduce the missing FSF facilities in the distributed implementation, starting with the ability to share any spare capacity that there may be in the network, using the same quality-of-service parameters specified in the contracts and already used by the processor FSF schedulers.

REFERENCES

[1] M. Aldea and M. González. “MaRTE OS: An Ada Kernel for Real-Time Embedded Applications”. Proceedings of the International Conference on Reliable Software Technologies, Ada-Europe-2001, Leuven, Belgium, Lecture Notes in Computer Science, LNCS 2043, May, 2001.

[2] Juan López Campos, J.Javier Gutiérrez and Michael González Harbour. “The Chance for Ada to Support Distribution and Real Time in Embedded Systems. Proceedings of the International Conference on Reliable Software Technologies, Palma de Mallorca, Spain, in Lecture Notes in Computer Science, Vol. 3063 , Springer, June 2004.

[3] L. Pautet and S. Tardieu. “GLADE: a Framework for Building Large Object-Oriented Real-Time Distributed Systems”. Proc. of the 3rd IEEE Intl. Symposium on Object-Oriented Real-Time Distributed Computing, (ISORC'00), Newport Beach, USA, March 2000.

[4] José María Martínez and Michael González Harbour. “RT-EP: A Fixed-Priority Real Time Communication Protocol over Standard Ethernet”. to appear in the Proceedings of the International Conference on Reliable Software Technologies, Ada-Europe-2005, York, UK, June 2005.

[5] Sprunt, B., Sha, L. and Lehoczky, J.P., “Aperiodic Task Scheduling for Hard-Real-Time Systems”. The Journal of Real-Time Systems, Kluwer Academic Publishers, 1, pp. 27-60, 1989.

[6] FIRST web page. IST Programme of the European Commission project IST-2001-34820.
`http://www.idt.mdh.se/salsart/first/`

[7] FSF/MaRTE OS home page.
`http://marte.unican.es/fsf/`

[8] FSF/SHaRK home page.
`http://shark.sssup.it/contrib/first/first.html`