Enhancing a Hard Real-Time Ethernet Protocol to support Distributed Contract-Based Bandwidth Reservation

Daniel Sangorrín, Michael González Harbour
Grupo de Computadores y Tiempo Real
Universidad de Cantabria
{daniel.sangorrin,mgh}@unican.es

Abstract

This paper describes the design, implementation and evaluation of a set of new services for a token-based hard real-time Ethernet protocol. The new services provide support to implement bandwidth reservations in the context of a contract-based scheduling framework.

1 Introduction

FRESCOR (Framework for Real-time Embedded Systems based on COntRacts) [1] is an EU project with the objective of providing engineers with a scheduling framework that represents a high-level abstraction that lets them concentrate on the specification of the application requirements. Key in FRESCOR is the concept of an integrated view of the different resources involved in a transaction, like the processor, network, memory or disk resources, that eases the deployment of complex distributed applications with a variety of real-time requirements, including hard real-time behavior as well as soft requirements.

The framework is based on the notion of *contracts*, which are negotiated between the application and the system.

In order to integrate a new network protocol into the framework, an adaptation layer in charge of negotiations of *contracts* and creating *virtual resources*, to enforce the contracted bandwidth parameters, must be implemented. This adaptation layer requires some services from the underlying protocol:

- A way to *control* and *analyze* the traffic
- A way to *reliably spread* the results of a negotiation
- A way to negotiate in *mutual exclusion*

The goal of this paper is to describe our solution for these requirements using the hard Real-Time Ethernet Protocol called RT-EP [2].

The rest of the paper is organized as follows. In Section 2 we provide details about related works on the matter. Section 3 describes the solution for the control and accounting of network traffic through network scheduling servers and fixed priorities. Section 4 shows the implementation of a reliable multicast mechanism that solves the requirement to reliably spread the results of a negotiation. Section 5 presents a solution for the requirement to negotiate in mutual exclusion, by adding distributed mutexes to the protocol. Finally, section 6 shows performance metrics and section 7 gives our conclusions and future work.

2 Related work

2.1 RT-EP

RT-EP is a software Ethernet protocol for hard real-time applications where a fixed pri-

[1]This work has been funded in part by the Plan Nacional de I+D+I of the Spanish Government under grant TIC2005-08665-C03 (THREAD project), and by the European Union's Sixth Framework Programme under contract FP6/2005/IST/5-034026 (FRESCOR project). This work reflects only the author's views; the EU is not liable for any use that may be made of the information contained herein.

ority can be assigned to each packet. RT-EP organizes stations in a logical ring and operates in two phases [2]. In the *arbitration phase* a RT-EP token packet is circulated in the ring to find out the station with the highest priority message. In the *transmission phase*, the winner is granted the right to transmit.

RT-EP provides the ability of recovering from some fault conditions [2]. The recovery method is based on simultaneous listening to the media, in a promiscuous mode (hubs) or using broadcast addresses (switches). Each station, after sending a packet, listens to the media for an acknowledge, which is implicit in the correct transmission of the next frame by the receiving station. If no acknowledge is received after a timeout, the station assumes that the packet is lost and retransmits it.

2.2 FTT-SE FNA

Another protocol that is being integrated into the FRESCOR framework is FTT-SE [3]. As FTT-SE is a *master-slave* protocol, the control and analysis of the traffic is done by the master using a table, and mutual exclusion is provided through ordinary mutexes.

Compared to that work, here we address the challenge of implementing the same bandwidth reservation capabilities using a fully *distributed* approach.

2.3 DFSF

In a previous project [4], we made a proof-of-concepts implementation of a distributed bandwidth reservation mechanism by embedding the negotiation process inside RT-EP [5]. In that work, negotiations were not required to have hard deadlines because they were supposed to happen only sporadicly.

Compared to that work, in this paper we implement a hard real-time solution. In addition, our solution is easier to mantain and extend because it is based on a layered architecture approach. Other enhancements over this previous work are the addition of real-time reliable multicast capabilities, real-time distributed mutexes, which are available from the application and the posibility of combining scheduling servers, presented previously in [4], with regular fixed-priority messages.

3 Server-based schedulers and fixed priorities for networks

3.1 Introduction

Server-based scheduling techniques have been used for a long time, typically associated with processor time scheduling, to limit the bandwidth assigned to a particular computation or set of computations while also guaranteeing some minimum level of service. Servers such as the periodic server [6], the sporadic server [7], or the constant bandwidth server are examples of such scheduling policies.

The concept of server is also applicable to networks. For example, the *leaky bucket* concept used in network traffic shaping [8] is similar to the sporadic server; EDF-based servers [9] have been successfully applied to the CAN bus.

When a server is used to schedule a network the concept of execution time must be mapped into *transmission time*. In most networks, information is sent in units called packets which are usually non preemptible and therefore constitute the minimum effective budget. Therefore, in a network the most natural unit for measuring budget is a number of packets. In RT-EP the server's budget will be consumed one packet at a time. The non preemptability property of a single packet must be taken into account as a bounded blocking effect on higher priority servers.

In [5], server-based schedulers for networks were implemented. Since RT-EP packets use fixed-priorities, a natural choice was to base the schedulers on the *sporadic server* policy [7]. Fig. 1 shows the architecture of the implementation of servers in RT-EP. Each server is assigned an initial transmission capacity (in network packets), a replenishment period (a time interval) and a priority. Internally, servers keep track of the current available budget and the count of packets pending to be sent. All this information is kept in a table at the sending node.

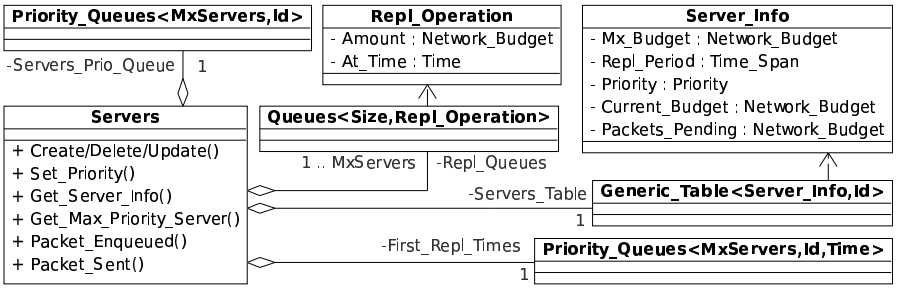

Figure 1: RT-EP Network Servers

3.2 Our enhancement

The extension that we have made to RT-EP, allows messages to be sent through a sporadic server or using plain fixed priorities in the same network. For the plain fixed priority messages, there is a priority queue (see Fig. 2), called the *FP transmission queue*, that stores the packets pending to be sent. Packets sent through a sporadic server are stored in a FIFO queue associated with each server, called the *server transmission queue* (see Fig. 2). This architecture allows engineers to design several priority bands for using fixed priorities and other bands for servers, and schedule them at the same time. It also eases the maintenability of the protocol by eliminating the need of branching the RT-EP implementation in two different versions.

When the RT-EP main communication task needs to find out which is the packet with the highest priority, it executes the possible pending replenishment operations and then it checks both priority queues, the servers priority queue and the FP transmission queue, to find out which is the pending packet with the highest priority. If the packet is from a server, it finds it in the corresponding server transmission queue. In Fig. 2 we see the queues involved and another one for mutexes that will be explained later.

In the receiving end, RT-EP stores the messages in priority reception queues, so that they can be retrieved in priority order, each associated with a channel number. Note that to control the usage of the network we only need control it in the sender endpoint.

4 Real-Time Reliable Multicast

4.1 Introduction

A reliable multicast service must ensure that packets are delivered to receivers from the multicast group in a bounded amount of time, free of errors and in the order they were sent by the source.

The most typical approach to reliable multicast is the *sender-initiated* approach, where the sender maintains the state of all the receivers from whom it has to receive acknowledgments (acks). But this technique has a scalability drawback, commonly known as the acknowledgement implosion problem. [10]

A *receiver-initiated* approach, where receivers request the sender the retransmission of packets that are missing, could be more scalable but requires infinite buffers to prevent deadlocks. [10]

In [10] two other solutions to the acknowledgment implosion problem that operate correctly with finite buffers are described. *Tree-based* protocols organize the receivers in a tree and send acks along the tree. And *ring-based* protocols where the basic premise is to have only one token site responsible for acknowledging packets back to the source. These solutions apply to generic protocols rather than to specific implementations so a direct implementation on RT-EP would be quite inefficient compared to the approach presented in this paper.

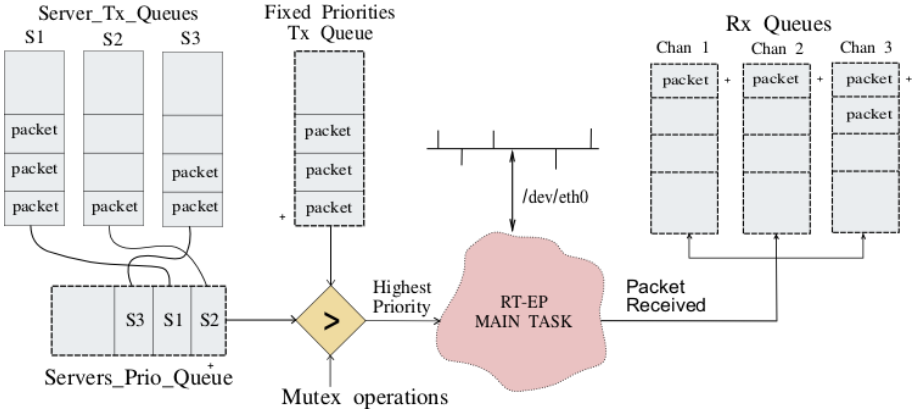

Figure 2: RT-EP Packet Buffers

4.2 Our approach

An option available in RT-EP to avoid the acknowledgment implosion effect of the *sender-initiated* approach is to take advantage of the RT-EP arbitration phase to receive from all the receiving nodes a negative acknowledgment (NACK) to a multicast message.

8	14	13	33	4
Pre	EthHd	TokenHd	Spare	FCS

Figure 3: RT-EP Token packet

In this work we have chosen another approach, that takes the premise of ring-based protocols [10], where only one token site is responsible for acknowledging packets. Our approach goes one step further by distributing the responsibility of retransmissions among all the nodes in the ring.

The key of our approach consists on sending multicast information in the spare bytes of the token packets which, due to the Ethernet minimum frame size have some spare transmission capacity (33 bytes, see Fig. 3). In addition, we can take advantage of the mentioned built-in fault handling mechanisms present in the protocol to ensure automatic retransmission of faulty packets. The main fields of a multicast packet (which would go in the spare bytes of the RT-EP Token) are:

- **MType:** Multicast Type selects the multicast operation: an ordinary multicast message, a mutex lock or a mutex unlock operation
- **MA:** Multicast Address: a static table, similar to the ring configuration table, states which nodes belong to a multicast group
- **Chan:** the destination channel
- **Prio:** the priority of the packet
- **Len:** the length of Info (in bytes)
- **Inf:** the information itself
- Also, the *source address* is implicit in the Token Master field of the RT-EP header

Compared to the first option the available capacity for information is smaller because we make use of the spare bytes in the token. But on the other hand, this approach provides better response times, because retransmissions are handled immediately, and it is simpler. Note that in the first option new functionality should be added to receivers so that they could discard multicast retransmissions that had already been received.

Summing up the multicast procedure, when a node wants to send a multicast packet it enqueues the message as a normal message with

the desired priority, but specifying as the destination address a multicast address. Then it competes in the arbitration phase of the protocol like a normal packet. When it is granted with the right to transmit, instead of sending a normal message, a new arbitration phase is initiated and the multicast message is sent through the token. Thanks to the reliability of the token transmission, which used implicit acknowledgements and a timeout-based fault handling mechanism, the protocol ensures that the multicast message is received by all the nodes and free of errors, up to the same reliability level as the normal packets have.

5 Real-Time Fault-Tolerant Distributed Mutexes

5.1 Introduction

Over the last decades there has been a lot of work in the field of distributed mutual exclusion. Some helpful works, [11] [12] [13], have tried to categorize and compare those algorithms. Probably, the most straightforward approach to achieve mutual exclusion is to use a centralized coordinator that serves requests to enter a critical section. The problem of this approach is that the coordinator represents a single point of failure and it can become a performance bottle-neck [14]. The distributed approach is certainly where most of the research has been done. Distributed algorithms are usually divided in Token-based and Non-Token-based algorithms.

In a Token-Based algorithm, the right to enter a critical section is equivalent to the possession of a unique abstract object, called a Token. How this Token is obtained differentiates the algorithms. One of the simplest Token-based algorithm uses a logical ring where a Token is passed from process to process in a single direction [15] [13]. Another interesting Token-based algorithm was proposed in [16] where in order to get the token, a process sends a request with a sequence number to all other processes and then waits for the arrival of the token message.

In most of the Non-Token-based algorithms, also called Permission-based algorithms, the right to enter a critical section is formalized by receiving permission from a set of nodes in the system. In a previous work [17], Sanders made an abstraction information structure that generalizes this approach. Most of these algorithms (i.e.: [18]) use logical clock timestamps [19] to order the requests.

Other aspects that differenciates these algorithms are: static vs dynamic, logical-structure-based vs broadcast-based. Some of these algorithms have been extended to be fault-tolerant and suitable for real-time systems.

Our requirements on the mutual exclusion algorithm that we are looking for are rather hard. It must provide bounded times on the lock and unlock operations, be fault-tolerant to node failures or lost packets, be simmetric in the sense that there is no node more important than the rest, simple and efficient.

On the other hand, we have an advantage that we may be able to exploit. In RT-EP, messages from different nodes cannot be transmited asynchronously; there is a Token that allows a single node to transmit at each time.

In [5], a token-based algorithm similar to the mentioned for logical rings [15] was used to implement mutual exclusion. Some extra fields were introduced in the header of every packet of the protocol to declare the status of a single distributed mutex. In this approach, when a node wants to lock the mutex, it waits for the token and checks the mutex status field. If the mutex is free, it locks it. The problem of this approach is that even when there is no process wanting to lock the mutex, the information is always circulated in every packet. Another problem is that it doesn't provide scalability because more mutexes would mean increasing the number of extra fields. Also, as RT-EP does not conform a perfect circular ring, the algorithm does not fulfil the non-starvation property.

5.2 Our approach

In this paper, we study, implement and compare two alternative algorithms:

The **first one** is a permission-based algorithm that presents similarities to the generalized mutual exclusion algorithm described in [17] but instead of using timestamps we exploit the implicit transmission token present in RT-EP to order mutex operations according to a priority. When a process wants to lock a mutex, it enqueues a REQUEST message with a priority in the RT-EP queues. Eventually, it will win the arbitration phase and the REQUEST will be sent in the spare bytes of the RT-EP token packet. The rest of the processes, receive the REQUEST in the Token. If one of them is holding the mutex, it will set a GRANT bit to FALSE. When the token packet comes back to the sender, it will look at the GRANT field. If it is True, it will become the new holder of the mutex. If not, it will wait for a RELEASE message that will be sent when the node holding the mutex executes an unlock. This approach solves the problem of scalability because a high number of mutexes can be used. It also helps in bounding the mutex locking time because locking operations can be priorized. It is simmetric, simple and efficient. Also, lost packets and node faults are handled by RT-EP. In particular, when a node is missing, all nodes are informed about it [2] and therefore we can wake up requests that were enqueued for a RELEASE message. On the other hand, there is an efficiency drawback in the algorithm. When there is a RELEASE message, all the enqueued REQUESTs are awaken to fight again through priorities for getting the Token. An alternative would be that the node holding the Mutex, keeps track of the received REQUESTS and, send them later with the RELEASE message that would only awake the highest priority petition. This solution is similar to the one proposed in [16] but it is not scalable because you need to send an array of data that depends linearly on the number of nodes.

The **second algorithm** overcomes all these problems and fulfils all the requirements. It is a token-based algorithm that exploits the RT-EP transmission token to be more efficient. In this algorithm, each node has local information of the distributed mutex: the id, whether it is locked or not, and the holder. When a process wants to lock a mutex it will add this command with a priority to the RT-EP queues. In the arbitration phase, when the node must check if it has a higher priority than the one received in the token, it checks all the queues that we can see in Fig. 2. The Mutex priorities are only taken into account if the Local mutex indicates that it is not locked by someone else. Eventually it will win the arbitration phase, lock the mutex and notify the rest of the processes (during the following arbitration phase), which will update their local variables accordingly. For unlocking the mutex, a similar operation will take place but with an Unlock message. As we can see, only TWO RT-EP messages are needed to lock and unlock a mutex. Furthermore, from the analysis point of view they behave just like normal messages so it simplifies the design a lot. Like in the previous alternative, we can make use of the built-in fault-handling mechanisms. When a node fails, all nodes will be notified and they will check if the failing node had any mutex locked, and will update their local variables accordingly.

In both algorithms, for the case in which two nodes compete for the mutex with the same priority, the order is not defined. This is typically addressed using logical clocks to timestamp [19] the requests. In our case, RT-EP sequence numbers could be used instead, but since for hard real-time analysis only the worst case scenario is relevant, this timestamping procedure has not been implemented in the protocol.

6 Performance and measurements

To measure the performance of these new services we have used a time measurement library that stores the worst, average and best measurements and sends them to a linux node periodically. The measurement platform is a ring configured with two AMD Duron 800Mhz stations connected through a 100 Mbps Ethernet switch. Since RT-EP does not provide a global time basis, in order to measure the amount of time to send a message we measure the time

Operations	Worst(ms)	Avg(ms)	Best(ms)
Send+Receive a FP message in previous RT-EP	0.98	0.85	0.68
Send+Receive a FP message in new RT-EP	0.98	0.87	0.69
Send+Receive a SERVER + Low Prio FP messages	1.35	1.1	0.7
Send+Receive a FP + Low Prio SERVER messages	1.33	1.1	0.7
Send+Receive a SERVER message	0.98	0.86	0.7
Lock 1st Algorithm	1	0.8	0.5
Lock 2nd Algorithm	0.36	0.2	0.09
Unlock 1st Algorithm	0.5	0.4	0.2
Unlock 2nd Algorithm	0.37	0.24	0.1
Send+Receive a MULTICAST message	1.79	1.6	1.4
Token round using FP in previous RT-EP	0.37	0.36	0.36
Token round using FP in new RT-EP	0.38	0.37	0.37

required to send a packet, execute a handler in the other node and receive an answer.

We see that the addition of servers to the protocol does not affect the efficiency of the previous implementation with fixed priorities (FP). Also, the arbitration phase remains with similar timings. When we mix servers and fixed priorities the timings are a bit higher (although part of that time is due to low priority blocking times). Servers timings have been measured without too many pending replenishment operations and that is the reason they are similar to FP timings. The good point is that, as the new architecture is modular we can remove the servers when we want to use a static fixed priority RT-EP.

Regarding the metrics of the two mutual exclusion algorithms proposed, we see in the table that the second algorithm is slightly better in performance. Both algorithms have good timing properties when compared to other algorithms found in the literature. This is achieved because in RT-EP we can assume that there cannot be asynchronous messages being transmitted at the same time. The difference between the algorithms is that in case of contention for locking the mutex, in the first algorithm more messages are sent through the network, with the associated overhead. On the other hand, the first algorithm has a good property when a node wants to join the network because it does not need to update any local variable.

The algorithm based on a token-ring used in [5] would have similar metrics on average but with the mentioned starvation issues.

7 Conclusion

The fixed-priority real-time ethernet protocol called RT-EP has been extended with the addition of three new services: server-based scheduling policy combined with fixed priorities, reliable multicasts and real-time distributed mutexes. The new services have been implemented and tested, and their performance metrics show that the transmission times are very efficient. These services will allow us to support bandwidth reservations in a distributed contract-based scheduling framework that is under development. This framework, called FRESCOR, defines a network adaptation layer in charge of negotiating contracts and managing virtual network resources that keep track of the network resources consumed, and provide the necessary quality of service guarantees for supporting both hard and soft real-time requirements. The new RT-EP protocol will be one of the implementations of this layer in FRESCOR.

References

[1] Frescor project home page. `http://www.frescor.org/`.

[2] José María Martínez and M. González Harbour. RT-EP: A Fixed-Priority Real

Time Communication Protocol over Standard Ethernet. In *10th International Conference on Reliable Software Technologies, Ada-Europe*, pages 180–195. Springer, June 2005.

[3] Ricardo Marau, Luís Almeida, Paulo Pedreiras, M. González Harbour, Daniel Sangorrín, and Julio M. Medina. Integration of a flexible network in a resource contracting framework. In *In Proc. of the WiP session of the 13th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'07)*. IEEE, April 2007.

[4] M. Aldea et al. FSF: A Real-Time Scheduling Architecture Framework. In *12th IEEE Real-Time and Embedded Technology and Applications Symposium (RTAS'06)*, pages 113–124, San Jose (CA, USA), April 2006. IEEE.

[5] José María Martínez, M. González Harbour, Juán López Campos, J.Javier Gutierrez, and Julio L. Medina. Adding contract-based reservation services to a Hard Real-Time Ethernet Protocol. In *4th International Workshop on Real-Time Networks (RTN'05)*, Palma de Mallorca (Spain), July 2005.

[6] Giorgio C. Buttazzo. *Hard Real-Time Computing Systems.* Kluwer Academic Publishers, 2002.

[7] B. Sprunt, L. Sha, and J.P. Lehoczky. Aperiodic task scheduling for hard-real-time systems. In *The Journal of Real-Time Systems*, pages 27–60, Palma de Mallorca (Spain), 1989. Kluwer Academic Publishers.

[8] Clarence Filsfils John Evans. Deploying ip and mpls qos for multiservice networks: Theory and practice. Morgan Kaufmann Publishers, 2007.

[9] Thomas Nolte, Mikael Nolin, and Hans Hansson. Real-time server-based communication for can. pages 353–372. 2006.

[10] Brian Neil Levine and J. J. Garcia-Luna-Aceves. A comparison of reliable multicast protocols. *Multimedia Systems*, 6(5):334–348, 1998.

[11] M. Velazquez. A survey of distributed mutual exclusion algorithms. In *Technical Report CS-93-116, Colorado State University*, 1993.

[12] P. C. Saxena and J. Rai. A survey of permission-based distributed mutual exclusion algorithms. In *Computer Standards and Interfaces Volume 25 Issue 2*, pages 159–181, 2003.

[13] George Coulouris et al. *Distributed Systems: concepts and design 4th Ed.* Addison Wesley, 2005.

[14] Andrew S. Tanenbaum et al. *Distributed Systems: principles and paradigms.* Pretince Hall, 2007.

[15] G. LeLann. Motivation, objective, and characteristics of distributed systems. In *Springer-Verlag, 1(1):1- 9*, 1978.

[16] G. Ricart and A.k. Agrawala. Author response to 'on mutual exclusion in computer networks' by carvalho and roucairol. In *Communications of the ACM, vol 26 no. 2*, pages 147–148, 1983.

[17] Beverly A. Sanders. The information structure of distributed mutual exclusion algorithms. In *ACM Trans. Comput. Syst. 5(3)*, pages 284–299, 1987.

[18] G. Ricart and A.k. Agrawala. An optimal algorithm for mutual exclusion in computer networks. In *Communications of the ACM, vol 24 no. 1*, pages 9–17, 1981.

[19] Leslie Lamport. Time, clocks, and the ordering of events in a distributed system. In *Communications of the ACM Volume 21, Issue 7*, pages 558–565. ACM Press, 1978.