A Multipoint Communication Protocol based on Ethernet for Analyzable Distributed Real-Time Applications

José María Martínez, Michael González Harbour, and J. Javier Gutiérrez

Departamento de Electrónica y Computadores
Universidad de Cantabria
39005-Santander, SPAIN
chema@gmx.net, mgh@unican.es, gutierjj@unican.es

Abstract

This paper presents a work-in-progress design and implementation of a software-based token-passing Ethernet protocol for multipoint communications in real-time applications, that does not require any modification to existing Ethernet hardware. Because the protocol is based on fixed priorities, applications using it can be easily modeled using common techniques for fixed priority systems, and well-known schedulability analysis techniques can be applied. We call this protocol RT-EP (Real-Time Ethernet Protocol).

1. Introduction[1]

Ethernet is by far the most widely used local area networking (LAN) technology in the world today. Unfortunately, Ethernet uses a non-deterministic arbitration mechanism (CSMA/CD) which makes it unsuitable for real-time communications. Several approaches and techniques have been used to make Ethernet deterministic in order to take advantage of its low cost and higher speeds than those of real-time field buses available today (like the CAN bus [9], for example). Some of these approaches are the modification of the Medium Access Control [6], the addition of transmission control [5], a protocol using time-triggered traffic [3], or the usage of a switched Ethernet [7].

Our research group has been working in the last few years on the development of MaRTE OS [1], a real-time kernel for embedded applications. The objective of this work is to add a real-time communication network to MaRTE. We want to achieve a relatively high speed mechanism for real-time communications at a low cost, while keeping the predictable timing behavior required in distributed hard real-time applications. The communications protocol proposed in this work can be classified as an addition of a transmission control layer over Ethernet, since it is basically a token-passing protocol in a bus [8].

1. This work has been funded by the *Comisión Interministerial de Ciencia y Tecnología* of the Spanish Government under grant TIC99-1043-C03-03

The paper is organized as follows: Section 2 describes how the communication protocol works. Section 3 gives details about the implementation and the model describing the timing behavior of its implementation. In Section 4 we provide some results on the overhead introduced by this protocol. Finally, Section 5 gives our conclusions.

2. Description of the Communication Protocol

RT-EP has been design to avoid collisions in the Ethernet media by the use of a token. Each station (processing node or CPU) has a transmission queue, which is a priority queue where all the packets to be transmitted are stored in priority order. Each station also has a set of reception queues that are also priority queues. Packets with the same priority are stored in FIFO order. The number of reception queues can be configured depending on the number of application threads (or tasks) running in the system and requiring reception of messages. Each application thread should have its own reception queue attached. The application has to assign a number, the channel ID, to each application thread that requires communication through the protocol. This channel ID is used for the purpose of identifying communication endpoints in a given station.

The network is logically organized as a ring. Each station knows which other station is its predecessor and its successor, so the logical ring can be built. The protocol works by rotating a token in this logical ring. The token holds information about the station having the highest priority packet to be transmitted and its priority value. The network operates in two phases. The first phase corresponds to the priority arbitration, and the second phase to the transmission of an application message.

For the transmission of one message, an arbitrary station is designated as the *token_master*. During the priority-arbitration phase the token travels through the whole ring, visiting all the nodes. Each station checks the information in the token to determine if one of its own packets has a priority higher than the priority carried by the token. In that case, it

changes the highest priority station and associated priority in the token information; otherwise the token is left unchanged. Then, the token is sent to the successor station. This process is followed until the token arrives at the *token_master* station, finishing the arbitration phase.

In the message-transmission phase the *token_master* station sends a message to the station with the highest priority message, which then sends the message. The receiving station becomes the new *token_master* station.

The maximum information size held by a packet is limited by the information field in an Ethernet frame, and it can vary from 0 to 1492 bytes) [2]. Fragmentation of messages at this layer is not allowed.

3. Implementation and Modeling of RT-EP

The functionality and architecture of the multipoint communication protocol are shown in Figure 1. This protocol offers three functions to any application using the network: *send_info* (to send a message), *recv_info* (to receive a message), and *init_comm* (to initialize the network). The application threads encapsulate the information in a message type, which is used both for transmission and reception. This message type contains the destination station address, the destination channel ID, and the priority of the message. When a message is sent, it is stored in the priority queue on the transmitting station. There is only one thread, the *Main Communication Thread*, that is responsible of reading the packets from the transmission queue and of writing the received packets into the reception queues. The highest priority packet to be transmitted determines the priority used for the priority arbitration token, as we described in the previous section.

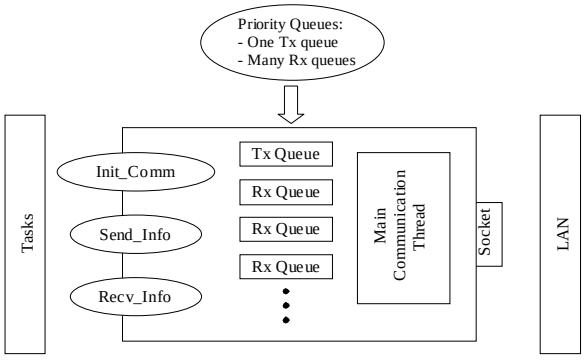

Figure 1. Functionality and details of RT-EP

The protocol has been implemented in GNU/Linux, directly over the network link layer, to test the design and to have a quick estimation of the overheads. In the near future the protocol will be implemented in MaRTE OS. We use RT-EP-token-packets to send the tokens and RT-EP-info-packets to send the information. To identify the stations, RT-EP uses the ethernet MAC addresses. There is no need for routing the information since we are working in a local area network.

In order for this protocol to work, the maximum number of communicating threads running in the system must be known at configuration time. This is the usual case in this kind of real-time system.

3.1. RT-EP Frame Formats

The frame formats used in our protocol goes into an Ethernet frame, which for Ethernet II has the following structure [2]:

8 bytes	6 bytes	6 bytes	2 bytes	46-1500 bytes	4 bytes
Preamble	*Destination Address*	*Source Address*	*Type*	*Data*	*Frame Check Sequence*

The *Type* field identifies what type of high-level network protocol is being carried in the data field. We use a value of 0x1000 for the *Type* field, which represents an unused number protocol, which could be changed if the protocol is registered in the future.

RT-EP packets are carried into the *Data* field of the Ethernet frame, that must be at least 46 bytes long. Due to this restriction, even though our packets can be less than 46 bytes long, a 46 bytes data field will be built. Our protocol has two types of packets:

- *Token Packet*: it is used to transmit the token and has the following structure:

1 byte	1 byte	2 bytes	6 bytes	34 bytes
Packet Identifier	*Token Identifier*	*Priority*	*Station Address*	*Extra*

  The *Packet Identifier* field is present also in the *Info Packet* and is used to identify the type of the packet. It can hold two different values for this type of packet: *Token* (used in the arbitration phase to get the highest priority packet) or *Transmit Token* (it grants the destination station permission to transmit a message). The *Token Identifier* will be used in the future to handle the loss of tokens. The *Priority* indicates the highest priority element on the LAN at the rotation time. The *Station Address* stores the address of the station with the highest priority packet. Finally, the 34 *Extra* bytes are needed to be compliant with the Ethernet protocol.

- *Info Packet*: it is used to transmit data and has the following structure:

1 byte	1 byte	2 bytes	2 bytes	2 bytes	0-1492 bytes
Packet Identifier	*Reserved*	*Priority*	*Channel ID*	*Info Length*	*Info*

The *Packet Identifier* has a value corresponding to an *Info Packet*. There is one *Reserved* byte for further use. The *Priority* field holds the priority of the packet being transmitted. The *Channel ID* is used to identify the destination queue in the destination station. The *Info Length* is the size of the data stored on the *Info* field. If the information to be transmitted is less than 38 bytes long, padding is performed in order to get the 46 data bytes required in an Ethernet frame.

3.2. RT_EP as a State Machine

RT-EP can be described as a state machine for each station in order to understand its functionality and obtain the relevant parameters for the different operations involved in the timing model. Figure 2 shows the states and the transitions between them, which are shortly described next:

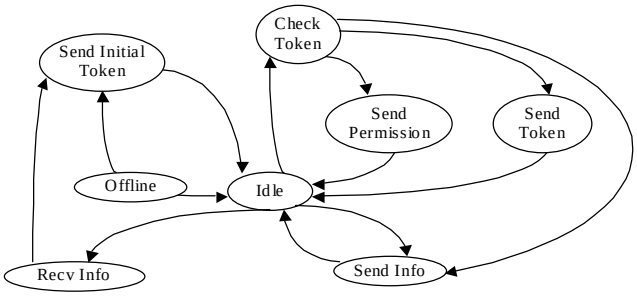

Figure 2. RT-EP state machine in each station

- *Offline*. It is the starting state reached during configuration time. Each station reads a configuration file describing the token ring and gets configured as one of its stations. The station configured as the initial *token_master* is set to the *Send_Initial_Token* state and the others are set to the *Idle* state.
- *Idle*. The station listens for the arrival of any packet. When a packet is received, a check is made to determine it's type: if it is an *Info Packet* the station switches to the *Recv_Info* state; if it is a *Token Packet*, two different states can be reached: *Send_Info* (if a *Transmit Token* is received), or *Check_Token* (when a regular *Token* is received).
- *Send_Initial_Token*. The station reaching this state becomes the *token_master.* A token is sent to the successor station, and the current station switches to the *Idle* state.
- *Check_Token*. If the station isn't the *token_master* the *Send_Token* state is reached; if it is, it switches to the *Send_Info* state if the *Station Address* is the current station, or to the *Send_Permission* state when not.
- *Send_Token*. The station compares the priority of the token with the highest priority element on its transmission queue, updates the token if its own priority is higher, and sends the token to next station. Then it switches to the *Idle* state.
- *Send_Permission*. The *token_master* role is lost and a the *Transmit Token* is built and sent to the highest priority station.
- *Send_Info*. This is the state in which a station has the highest priority packet on the ring and it is allowed to transmit it.
- *Recv_Info*. The information is written into the appropriate reception queue and the station switches to the *Send_Initial_Token* state, becoming the *token_master.*

3.3. MAST Model of RT-EP

This subsection draws out the modeling information of RT-EP according to MAST (Modeling and Analysis Suite for Real-Time Applications) [4]. This methodology provides an open source set of tools that enables engineers developing real-time applications to check the timing behavior of their application, including schedulability analysis for checking hard timing requirements. MAST includes the model of a fixed priority network as a specialized class of a processing resource. The model of the network encapsulates the relevant information to ensure that the schedulability analysis can be performed. In addition, MAST defines the network drivers, with parameters that represent the overheads of the activities executed by the processors to manage the communication packets.

We can use the MAST model to characterize RT-EP by obtaining the specific values for the network parameters. In order to have a complete description of the RT-EP model we must extend MAST by adding a new network driver (based on the existing *packet_driver*) which includes the operations to send and receive packets performed by the Main Communication Thread, the thread itself, and the protocol operations to manage and pass the tokens. The complete information to model RT-EP is described next using the MAST notation.

RT-EP Packet Driver. It is a specialization of a packet driver in which there is an additional overhead associated to passing the token. Its main attributes are:

- *Packet Send Operation*. It corresponds to the code executed in the *Idle* state followed by the *Send_Info* state.
- *Packet Receive Operation*. It corresponds to the code executed in the *Idle* state followed by the *Recv_Info* state and by the *Send_Initial_Token* state.
- *Number of Stations*.
- *Token Manage Operation*. Time required to send the token in the *Send_Token* or the *Send_Permission* states.

- *Token Check Operation*. Code executed in the *Idle* state followed by the *Check_Token* state.

The following attributes are used to characterize the *Fixed_Priority_Network* resource for RT-EP:

- *Max Packet Transmission Time* and *Min Packet Transmission Time*. They include only the time spent to send information bytes (1500 or 46 bytes).
- *Packet Overhead*. This is the overhead caused by the protocol information that needs to be sent before or after each packet. It is calculated as:

  (N+1)*(Min_PTT+EPB+TCO+TMO)

  which is the time spent to send a number of tokens equal to the number of stations, *N*, performing a complete circulation of the *Token*, plus one *Transmit Token*. The time to send a token is calculated as the sum of the *Min Packet Transmission Time*, Min_PTT, the time to send the Ethernet Protocol Bytes, EPB, the time of the *Token Check Operation*, and the time of the *Token Manage Operation*.
- *Max Blocking*. The maximum blocking caused by the non preemptability of message packets. In RT-EP, it is calculated as:

  N*(Min_PTT+EPB+TCO+TMO) + PWO+Max_PTT

  that represents a complete circulation of the token (*N* tokens sent), plus the blocking effect caused by the transmission of a lower priority packet (the *Packet Worst Overhead* and the *Maximum Packet Transmission Time*).

4. Overhead estimation

We have measured the CPU overheads of the protocol under GNU/Linux. Since this isn't a real-time OS, we have taken average values of our measurements. The times have been measured with a platform composed of two PCs (a Pentium 200 MMX and a Pentium 233 MMX) running GNU/Linux and connected by means of a null cable at 10 Mbps. The following table shows the average execution times of the operations involved in each state of the state machine description:

Operation	Time (µs)
Idle State	11.00
Send_Initial_Token	120.78
Check_Token	5.97
Send_Permission	107.61
Send_Token	99.42
Send_Info	149.70
Recv_Info	134.82

5. Conclusions

We have presented an implementation of a software-based token-passing Ethernet protocol for multipoint communications in real-time applications, that does not require any modification to existing Ethernet hardware. The protocol is based on fixed priorities and thus common tools for fixed priority schedulability analysis can be used. A precise timing model of the protocol has been obtained, which enables us to perform a schedulability analysis of a distributed application using this protocol.

Future work plans for this protocol are to extend it to take into account three kinds of failures: failure of a station, loss of a packet, or delay in handling a packet that could result in the duplication of a token. In addition, we have to port it to MaRTE OS.

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] Charles E. Spurgeon *Ethernet: The definitive Guide*. O’Reilly Associates, Inc. 2000.

[3] Paulo Pedreiras, Luis Almeida, Paolo Gar. “The FTT-Ethernet protocol: Merging flexibility, timeliness and efficiency”. Proceedings of the 14th Euromicro Conference on Real-Time Systems, Vienna, Austria, June 2002.

[4] M. González Harbour, J.J. Gutiérrez, J.C. Palencia and J.M. Drake: “MAST: Modeling and Analysis Suite for Real-Time Applications”. Proceedings of the Euromicro Conference on Real-Time Systems, Delft, The Netherlands, June 2001

[5] Chiueh Tzi-Cker and C. Venkatramani. “Fault handling mechanisms in the RETHER protocol”. Symposium on Fault-Tolerant Systems, Pacific Rim International, pp. 153-159, 1997.

[6] Jae-Young Lee, Hong-ju Moon, Sang Yong Moon, Wook Hyun Kwon, Sung Woo Lee, and Ik Soo Park. “Token-Passing bus access method on the IEEE 802.3 physical layer for distributed control networks”. Distributed Computer Control Systems 1998 (DCCS'98), Proceedings volume from the 15th IFAC Workshop. Elsevier Science, Kidlington, UK, pp. 31-36, 1999.

[7] Choi Baek-Young, Song Sejun, N. Birch, and Huang Jim. “Probabilistic approach to switched Ethernet for real-time control applications”. Proceedings of Seventh International Conference on Real-Time Computing Systems and Applications, pp. 384-388, 2000.

[8] ANSI/IEEE Std 802.4-1990. “IEEE Standard for Information technology--Telecommunications and information exchange between systems--Local and metropolitan area networks--Common specifications--Part 4: Token-Passing Bus Access Method and Physical Layer Specifications”.

[9] K. Tindell, A. Burns, and A.J. Wellings, “Calculating Controller Area Network (CAN) Message Response Times”. Proceedings of the 1994 IFAC Workshop on Distributed Computer Control Systems (DCCS), Toledo, Spain, 1994.