A Packet-Switching Satellite Emulator: A Proposal about Architecture and Implementation

Mario Marchese, Marco Perrando
CNIT - Italian National Consortium for Telecommunications, University of Genoa Research Unit
Via Opera Pia 13, 16145, Genova (Italy).

Abstract - **The paper contains a proposal about the design and implementation of an on-board satellite packet-switching emulator. The paper lists the requirements of the emulator that have been followed for the design and that will represent the guidelines for the implementation of the features not yet concluded. The characteristics of the emulator architecture are described, including motivations, advantages and drawbacks of the architectural choices. Particular attention is given to the need of a having a real-time behavior and an efficient transport of the information. Implementation details are reported. The support to the real-time over the Linux operating system, on which the implementation is performed, and the utilization of virtual tools to model the ACE emulator interfaces and to map the address at the data link layer are of particular interest. Neither the design or the implementation of the emulator is fully concluded, but some interesting results, essentially about computation for now, are already available and are reported.**

I. INTRODUCTION

Satellite communications have many advantages with respect to terrestrial communications. On the other hand, they amplify also many problems already existing in terrestrial networks. Within this framework, there is, on one hand, the opportunity of extending the use of satellite networks and the need of developing new instruments and schemes to improve the efficiency of the communications; on the other hand, it is important to observe the difficulty to test the solutions. A satellite system may be hardly studied on the field. It is expensive and it often concerns only software components for earth stations. Alternatives are necessary to investigate new systems and to evaluate the performance. The first one is the analytical study. It is very attractive but complex. It often requires simplifications and approximations. The second alternative is simulation. The behavior of a system is simulated via software. It is possible to by-pass the complexity of real systems and solution not yet technically feasible may be tested in view of future evolutions. The drawback is the need of modeling. A model is not accurate enough to consider all the aspects of a real system. A third alternative, which seems to summarize most of the advantages of the solutions mentioned, is emulation. Emulation is composed of hardware and software components that behave as a real system. An emulator allows using real traffic and it is similar to the real system also from the point of view of the physical architecture.

The paper presents the design and implementation of an emulator concerning on-board satellite packet switching. The work is part of a wider Project called "Emulation of on-board satellite switching systems" (ACE - ASI CNIT Emulator) [2], funded by the Italian Space Agency (ASI) and carried on by the Italian National Consortium for Telecommunications (CNIT). The project is aimed at testing the efficiency of on-board circuit and packet switching. The project will design and implement a circuit-switching and a packet-switching emulator, of which this work is a part, to obtain a good representation of a real system and to test solutions at any layer [1].

The paper is structured as follows. Section II contains the requirements of the emulator. Section III shows the proposal concerning the architecture and specifies the problems related to the real-time issue and the transport of information. Some more details about the implementation are reported in Section IV. Section V contains the results and Section VI the conclusions.

II. REQUIREMENTS

Aim. The aim of the work is the emulation of a real satellite network composed of earth stations and satellite devices including the satellite itself.

Scope. The emulation should range from Geostationary (GEO) to Low Earth Orbit (LEO) satellite systems. Earth stations may be connected to external PCs implementing algorithms oriented to the QoS (Quality of Service) at the network layer and new implementations at the transport layer. It should be possible to test different types of data link layers and different packet encapsulation formats at the data link layer.

Modeling. The statistics about losses and delays should take into account the real system and the status of the channels.

Transparency. The emulator should result as more transparent as possible towards the external world; it means that it should be seen as a real satellite device (e.g. a modem or a hardware card) by the external users (e.g. Personal Computers (PCs), routers, switches).

Scalability. The complexity of the overall system should be, as much as possible, independent of the enlargement of the emulated network. Adding a new component to the system (e.g. a new earth station) may increase the traffic and the computational load; but it should not affect the architectural structure of the emulator.

Traffic class support. The emulator, in an extended future version, should support different traffic classes at the data link layer.

Reliability. The results obtained by the emulator should be as close as possible to the results obtained by a real satellite system that implements the same packet switching strategy.

Architecture. It is not necessary that the internal architecture of the emulator is a mirror of the real system from a physical and topological level (e.g. not necessarily each hardware component of the emulator should correspond to each device of the real system).

0-7803-7400-2/02/$17.00 © 2002 IEEE

Simplicity. The emulator should result very simple from the point of view of the computational load.

Real time. The tool should work under stringent time constraints; in particular, at the interface with the external world.

Implementation. The implementation should use, as much as possible, hardware and software material already implemented in other projects.

Interface. The hardware interface towards the external PCs should be represented by devices (actually PCs, properly configured for this), called Gateways (GTW). The communication between the layers should be guaranteed by the use of exchanging Protocol Data Units (PDUs), where the information coming from the external PCs is encapsulated.

Core. The core of the emulator should be represented by a single tool, which imposes losses, delay and jitters, following a statistics, to each single PDU entering the emulator.

Transport of information. Each single PDU should be transported from the input to the output gateway.

QoS (Quality of Service). The PCs that utilize the emulator should be able to implement bandwidth reservation schemes and allocation algorithms to guarantee QoS to the users.

III. ARCHITECTURE

A. Revision of a Real System

A satellite system is constituted by a certain number of ground stations (each composed of a satellite modem that acts both at the physical and at the data link layer) and a satellite that communicates with the ground station over the satellite channel. The modem may be an independent hardware entity connected to other units by means of a cable or also a network adapter card plugged into a unit (e.g. the router itself or a PC), as shown in Fig. 1. In practice, it can be though as a data link layer of an overall protocol stack. For example, if an IP router is directly connected to the modem, the IP layer of the router interacts with the modem by sending and receiving traffic PDUs. Whenever a satellite modem receives a PDU from the upper layers, its main function is to send it towards the desired destination. On the other hand, when a modem receives a PDU from the satellite network, it must deliver it to the upper layers. The emulator should allow testing various kind of protocols, switching systems, and whatever else, in order to evaluate suitable solutions to be adopted. It is possible to identify, in a real satellite system, the following main parts: a modem with an interface towards the upper layers (namely the network layer); a channel characterized by its own peculiarities; a data link protocol over the satellite channel and a satellite with its on-board switching capabilities.

B. General Architecture

The reference architecture of the emulator is shown in Fig. 2, along with one possible system to be emulated enclosed in the cloud (a GEO satellite system has been depicted in this case). Different units called Gateways (GTW) operate as interface among the emulator and the external PCs. Each GTW is composed of a PC with two network interfaces: one towards the external world (a 10/100 Mbit/s Ethernet card), the other towards the emulator. An Elaboration Unit (EU), which has a powerful elaboration capacity, carries out most of the emulation, as the decisions about each PDU. The interface towards the external world concerns the GTWs; the loss, delay and any statistics of each PDU regards the EU; the real transport of the information PDU through the network concerns the input GTW and the output GTW. The various components are connected via a 100 Mbits/s network, completely isolated, by a full-duplex switch. In such way, the emulator has an available bandwidth much wider than the real system to be emulated, which should not overcome a maximum overall bandwidth of 10/20 Mbits/s. In more detail, Fig. 3 shows how the different parts of the real system (modem, data link protocol, channel and switching system, as mentioned in the previous sub-section) are mapped onto the different components of the emulator. As indicated in the requirements, the architecture of the emulator is not exactly correspondent to the real system. The earth station, identified by the grey rectangle, is divided, in the emulator, into two parts (GTW and EU). The network layer, the network interface towards the external world and the interface between the network layer and the satellite modem are contained in the Gateway (GTW). The other parts of the modem (i.e. the data link layer, protocol and encapsulation), the overall transmission characteristics (e.g. bit error ratio, channel fading, lost and delayed packets), the on-board switching architecture as well as the queuing strategies are contained in the Elaboration Unit (EU).

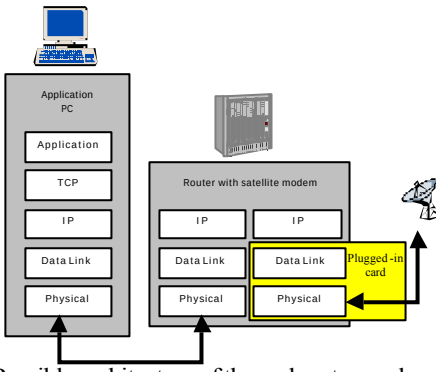

Fig. 1. Possible architecture of the real system - plugged-in card.

C. The interface

The interface between the modem at the ground station and the protocols of the upper network layers has been implemented in the emulator by creating a virtual device. It must be created on each of the GTWs and appears to the user and to the operative system as a network adapter (as a new Ethernet or Token Ring adapter). The Linux kernel, where the emulator is implemented, gives the possibility of creating such device by means of the tun/tap device [3] (see the Section IV for details). It may be used in two possible ways: a point-to-point device (namely the tun device, suitable for a direct link between two PCs) and a broadcast, Ethernet-like device (namely the tap device, useful to connect many PCs as if they were connected to the same Local Area Network - LAN). The latter choice has been adopted in the emulator because it is more similar to the real system structure. By the use of the tap device, the GTWs are actually connected by means of a virtual network that takes the PDUs and transports them as if they were transmitted over the real satellite system. After running the emulation software, a new network adapter

(the tap device) is ready to be used as an Ethernet adapter, with its new 48-bit address. The new network adapter can be linked to the any network protocol (such as IP, IPX, Novell). For example, if IP is used, an IP address shall be configured over the tap device and static routes or a routing daemon (e.g. routed) shall be started. Moreover, other IP configuration tasks may be performed, exactly as the virtual device was the real satellite modem, which acts at the data link layer. Transparency is the real advantage of this solution. The approach allows the emulator to behave exactly as a broadcast link that connects more stations. It is not mandatory to use only one network layer protocol; every layer 3 protocol suitable to work with an Ethernet network adapter may be adopted.

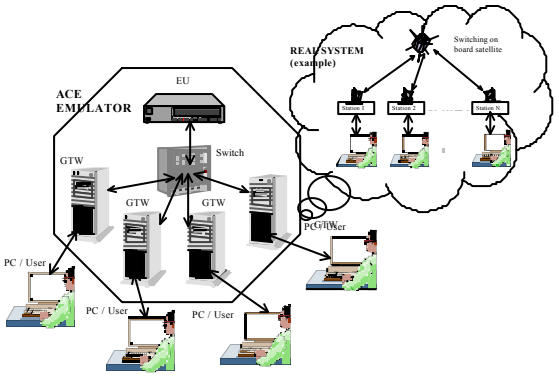

Fig. 2. Overall Emulator Architecture and Real System.

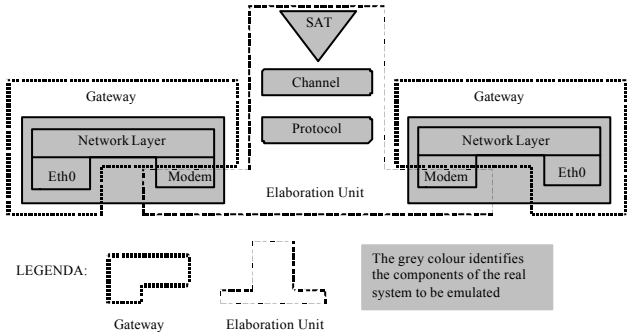

Fig. 3. Emulator versus Real System.

D. Transport

The topic concerns the transport of information between the different components that constitute the emulator (i.e. GTW and EU). Two different kinds of information have been identified: the real traffic (i.e. a packet containing the PDU), which the emulator transmits between the Gateways, and the control information (namely a control packet), which is related to the computation of the emulation results (i.e. the "destiny" of each PDU). A traffic packet (e.g. pkt_X) is directly sent to the proper output Gateway. A control packet (e.g. ctrl_pkt_X), which contains concise information about the traffic packet pkt_X, is sent to the EU. The EU performs most of the emulation and, on the base of statistics, takes decision about the traffic packet pkt_X. The EU reports them in ctrl_pkt_X and sends it the output Gateway. The latter, once received pkt_X, must store it until the arrival of ctrl_pkt_X. When the output GTW gets such information, it can properly act on the PDU by discarding it, by delivering it to the upper network layer at an exact time instant, by corrupting some bits or by performing other actions as indicated in the control packet. The same communication technique has been adopted for both types of information. A new protocol (called ACE) has been created in order to transport information. Due to the structure of the emulator (a set of directly connected PCs), the ACE protocol is encapsulated into the layer 2 PDU of the GTWs and of the EU. No routing is needed because all PCs are directly connected among them. The approach allows saving the amount of memory dedicated to contain the header bytes concerning the upper protocol address (20 bytes for an IP encapsulation). At the moment, Ethernet is the layer 2 protocol, but it may be changed by a small adaptation of the source code. Another advantage is the following. Once the traffic packet enters the ACE emulator (through the virtual interface implemented by the tap device), it does not pass through other protocol layers (e.g. IP) until it exits from the emulator at the output Gateway (through the tap device). It means that the ACE protocol is a layer 2 protocol. Traversing more layers (i.e. encapsulating ACE at higher layers) could overload the gateways because the traffic should pass twice through the network layer. The first time as it would do in the real system (e.g. through an external router or through the source PC) and the second time when it is sent by the emulator software to the output GTW. In the working hypothesis made, the emulator must perform only a simple mapping between the layer 2 address of the virtual device (the tap device) and the address of the physical device (e.g. the Ethernet card connecting the units among them). The structure of an ACE packet is very simple: the first byte carries the identification code of the packet, while the other 1499 bytes carry the ACE packet header and payload. Only two kinds of packets have been created for now: the traffic and control packets, identified, respectively, with code identifier 1 and 2. If a PDU cannot be entirely contained into an ACE traffic packet, part of it may be sent to the destination through the control packet. In such a way only two packets are necessary to emulate a single PDU. If the payload size of the virtual device packet is larger than the physical device packet, then more than one physical packet should be used to send the PDU to the destination. Both the interfaces are Ethernet in the present configuration of the emulator, so a virtual PDU of 1500 bytes can be divided into two parts: the biggest one (nearly 1450 bytes) is sent directly to the destination, while the remaining part flows through the control packet. A comparison between the sequence of operations necessary to deliver a PDU from the input and the output GTW in real system and in the emulator is reported in Table I.

E. The Real-Time Issue

The emulation system should act under stringent time constraints at the external interfaces. The interfaces should be real-time devices. The emulator interfaces the upper layers of the protocol architecture in two different ways: collection and deliver of a PDU. The software component that carries out these tasks shall operate with precise timing. Otherwise, the results obtained by the emulator could be unreliable. For these reasons a real-time support has been inserted in the GTWs only for the operations of collecting and delivering of a PDU. Synchronization among all the emulator components is also necessary. A tool called Network Time Protocol (NTP) is used in the current implementation. On the other hand, it is not necessary that the operations performed inside the EU act under strict time constraints

but it is needed they provide the results in time to be applied to the real traffic in transit. The aim may be reached by optimizing the emulator code or by using more computing power in the Elaboration Unit.

TABLE I
Comparison between Real System and Emulator operations

Real System	Emulator
The network layer sends the PDU to the input modem specifying a destination address	The network layer sends the PDU to the virtual device specifying a destination address
The input modem sends the PDU to the output modem through the satellite link	the virtual device (contained in the input GTW) collects the PDU the input GTW resolves the physical address of the output GTW and sends the PDU to the output GTW by using the ACE protocol the output GTW receives the PDU, stores it in a buffer and waits for an ACE control packet from the EU the input GTW sends the ACE control packet to the EU the EU receives the control packet, performs the necessary operations and sends another ACE control packet containing the "destiny" of the PDU to the output GTW
The out put modem delivers the PDU (how and if the modem receives it) to the upper network layer	The output GTW receives the ACE control packet and delivers the PDU to the upper network layer or drops it, in dependence of the indications contained in the control packet

IV. IMPLEMENTATION

A. Virtual Devices

Concerning the virtual interface of each GTW to the upper layer, the *tun/tap* tool [3] is used within the framework of the Linux operating system. The tool is a file, from the point of view of a programmer. The PDUs flow through the protocol layers via a simple read/write operation over a file. Writing in the file corresponds to the delivery of the PDUs to the upper layer; reading in the file means to collect the PDUs that the upper layer sends to the virtual device. The file is treated asynchronously. Delivering a PDU is an asynchronous *write* operation; if the device is not ready, the real-time system tries writing again at the next cycle. Also concerning the collecting operation an asynchronous *read* operation is used, because, when the emulator is ready to receive data, it is not necessarily true that the upper layer has data to send. If the operation were not asynchronous the overall program would stay blocked until the arrival of a packet from the upper layer.

B. Real Time

The Linux operating system is not a real-time system. Two distinct methods have been found to obtain a support to real-time in this environment: use of the real time support by patching the Linux kernel, which, originally, is a non-real time operating system (see [4], for an example) and use of the device Real Time Clock (RTC) [5], which is available in the mainboards of the PCs and usable through a proper device of the Linux kernel. The former is probably the most efficient and precise but it is more complex. The current version of ACE is based on RTC. The real-time portion of the source code has been isolated from the rest and a change towards a kernel-based solution should be simple and quick. The RTC limit is a maximum work frequency (to be set) of 8192 Hz. Such value has a period of 122 s. It could create a problem with small and frequent packets.

C. Transport

To complete the discussion reported previously, Table II shows a C-like structure of the traffic and control ACE packet headers.

TABLE II
Traffic and Control ACE Packet Headers

```
struct ACEptdhdr /*
TRAFFIC_PKT */
{
 ace_type_t pkt_type;
 unsigned int
serial_number;
 unsigned short
payload_size;
 unsigned short
station_id;
};
```

```
struct ACEcphdr /*
CONTROL_PKT */
{
 ace_type_t pkt_type;
 unsigned int
serial_number;
 unsigned short
station_id;
 unsigned short
pdu_total_size;
 unsigned short
payload_size;
 __time_t
enter_time_sec;
 __suseconds_t
enter_time_usec;
 __time_t exit_time_sec;
 __suseconds_t
exit_time_usec;
 ace_pdu_stat_t
pdu_flags;
};
```

V. RESULTS

This section shows measurements performed on the Gateway part of the emulator that interacts with the upper network layer. This measure is dedicated to test the accuracy of the developed code in delivering the PDUs to the upper network layer at a given time instant. The operation analyzed concerns the emulation phase when the output GTW receives the ACE control packet and schedules the deliver of the PDU to the upper layer. No meaningful performance difference has been measured between the delivery operation and the collecting operation. Only the former is considered in the following. The difference between the instant of the real delivery and the scheduled instant is the performance index. The eight different tests have been composed as follows. The overall number of packets delivered in each test is 1000. The dimension of the packets delivered to the upper network layer is fixed at 1500 bytes (the Ethernet MTU). The scheduled time interval between the delivery of two consecutive packets is not time variant within the same test. The following values for the time interval have been tested: 20 s, 200 s, 2 ms, 20 ms. Two different values of RTC have been used: a frequency of 4096 Hz and a frequency of 8192 Hz (the maximum frequency settable on the RTC).

A frequency of 4096 Hz corresponds to an intervention of the CPU (Central Processor Unit), which performs the real delivery, approximately each 244 s; a frequency of 8192 Hz, each 122 s. The

delay between the instant of the real delivery operation and the scheduled instant and should range from 0 to 122 s (or 244 s). The mean value of the delay should be about 60 s. A delay constantly much higher than the upper bound means that the system is saturated and the real time behavior cannot be controlled. Table III contains the mean value of the mentioned delay for the different scheduled delivery interval and RTC. The results obtained respect the expected values. Only the case apparently more critical (20 s) behaves better than expected. The behavior is explained in Fig. 4 and in Fig. 5, where the events occurring in the emulator for the 20 s and the 200 s case, respectively, are reported over the time. In the first case, when the CPU time needed to process and deliver a packet is over, it is probable there is another packet scheduled to be served before the CPU returns to the *sleep* status, waiting for the next RTC signal. On the contrary, in the second case, when the service time is over, the process returns immediately to the *sleep* status because there is no packet scheduled to be sent. The overall effect, in the first case, is a drastic reduction of the delay value. Fig. 6 contains the mentioned delay versus time in the case of a RTC frequency of 8192 Hz and a scheduled delivery of 20 s. The figure allows checking the real behavior over time and to verify that not only the mean value is under the upper bound but that each single packet has been delivered with a low delay. Similar considerations may be done concerning the other tests (200 s, 2 ms and 20 ms), not reported. It is important to observe that the results provided by adopting a RTC frequency of 4096 Hz are also satisfying when they are compared with the mean delay of satellite systems. The advantage of a lower RTC frequency is a CPU less loaded. Anyway, the measures about the CPU load performed during the tests have shown an acceptable load level even in the case of a RTC frequency of 8192 Hz.

TABLE III
Mean value of the delay

Delivery interval	Mean delivery delay [ms]	
	4096 Hz	8192 Hz
20 s	57.656	25.559
200 s	112.840	70.024
2 ms	130.460	69.850
20 ms	129.870	70.284

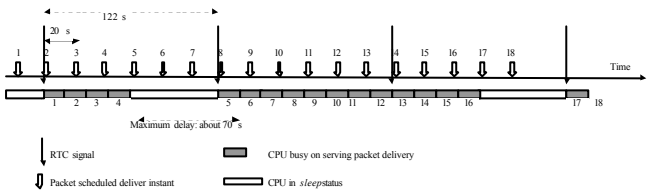

Fig. 4. Emulator events - 20 s.

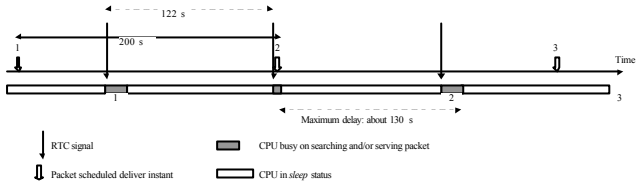

Fig. 5. Emulator events - 200 s.

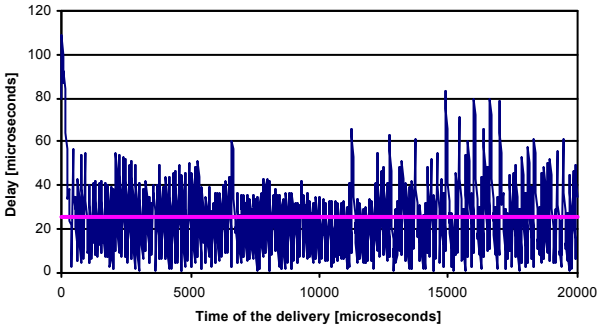

Fig. 6. Delay versus time, 8192 Hz, 20 s.

VI. CONCLUSIONS

The paper has described the requirements and the architecture of an emulator for packet-switching satellite systems. The emulator is composed of units called Gateways (GTW). An Elaboration Unit (EU), which has a powerful elaboration capacity, carries out most of the emulation, as the decision about loss and delay of data. Different parts of the real system are mapped onto the different components of the emulator

The results have been dedicated to test the accuracy of the developed code in exchanging data between the emulator and the upper network layer. The operation concerns the emulation phase when the output GTW receives the ACE control packet and schedules the deliver of the PDU to the upper layer. The difference between the real delivery and the scheduled instant is the performance index. The delay is constantly under the upper bound expected and the behavior of the overall system very satisfying.

ACKNOWLEDGEMENTS

This work has been supported by the Italian Space Agency (ASI) under the contract "Emulation of on-board satellite switching systems".

REFERENCES

[1] M. Allman, D. Glover, L. Sanchez, "Enhancing TCP Over Satellite Channels using Standard Mechanism", IETF, RFC 2488, January 1999.

[2] G. Albertengo, T. Pecorella, M. Marchese, "The ACE project: a Real Time Simulator for Satellite Telecommunication Systems", Proc. Sixth Ka-Band Utilization Conference, June 2000, Cleveland, Ohio, pp. 571-576.

[3] Universal TUN/TAP Device Driver, http://www.linuxhq.com/kernel/v2.4/doc/networking/tuntap.txt.html.

[4] Linux Real Time Application Interface Information, http://opensource.lineo.com/rtai.html.

[5] Real Time Clock Driver for Linux, http://www.linuxhq.com/kernel/v2.0/doc/rtc.txt.html.