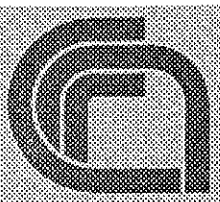
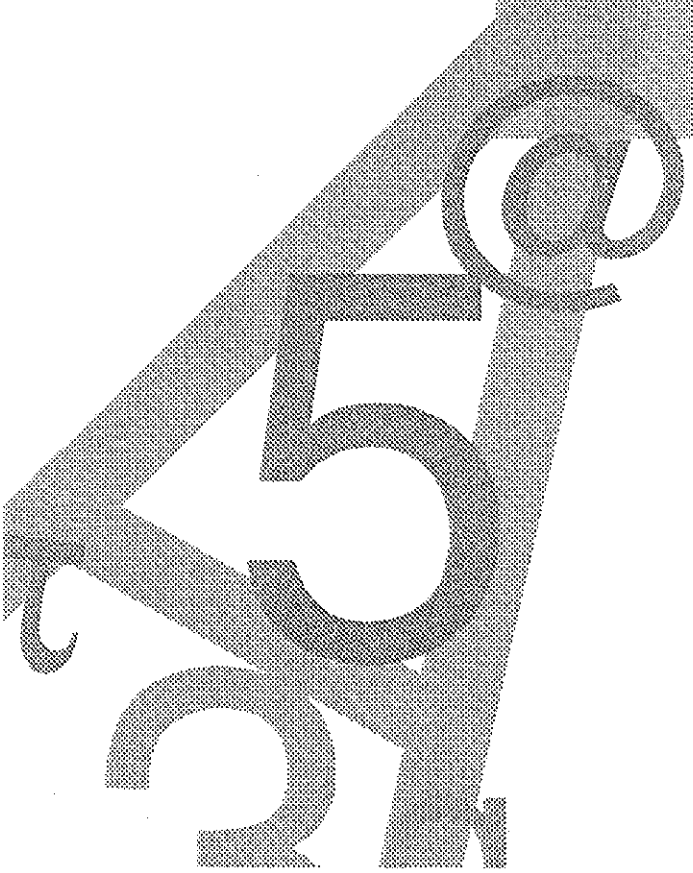


Consiglio Nazionale delle Ricerche

The VBR Send/Receive Facility in UNIX

N. Celandroni, E. Ferro, F. Potortì e I. Delic'

CNUCE: C96 - 06



THE VBR SEND/RECEIVE FACILITY IN UNIX

Nedo Celandroni
Erina Ferro
Francesco Potorti'
Izeta Delic' [*]

CNUCE/CNR Institute
Via S.Maria 36 - 56126 Pisa (Italy)

Tel. : +39-50-593207/593312/593203
Telex: 500371 CNUCE
Fax : +39-(0)50-904052
e: mail: n.celandroni@cnuce.cnr.it
e.ferro@cnuce.cnr.it
pot@cnuce.cnr.it

[*]Visiting Professor at CNUCE

CNUCE Report C96- 06

February 1996

Work carried out in the framework of the Italian Co-ordinated Project "Variable Bit Rate Traffic In An Interconnected Environment"

TABLE OF CONTENTS

1. INTRODUCTION 1

1.1 The VBR Traffic 1

1.2 The co-ordinated project 2

2. THE UNIX ENVIRONMENT 3

3. BASIC COMMUNICATION MECHANISM 3

4. DATA TRANSFER 4

5. HOW TO START PROGRAMS 5

REFERENCES 8

1. INTRODUCTION

1.1 The VBR Traffic

Variable Bit Rate (*VBR*) video is currently by far the most interesting and challenging real-time application. A VBR encoder attempts to keep the quality of video output constant and at the same time reduces bandwidth requirements since only a minimum amount of information has to be transferred. On the other hand, as VBR video traffic is both highly variable and delay sensitive, high-speed networks (e.g., ATM) are generally implemented by assigning peak rate bandwidths to VBR video applications, and by using the residual bandwidth for non-real-time traffic.

A variety of new applications, such as the transport of images, teleconferencing, moving images and a large volume of interactive computer data must be supported in an integrated fashion by today's high speed networks for long distances (i.e., several hundred miles). Networks that can guarantee a requested bandwidth must be adopted for real-time applications such as VBR video, due to their stringent quality of service constraints (i.e. a limited transfer delay and information loss sensibility) and the highly variable bit rate. This means that such applications must be assigned high priority. Data services, on the other hand, tolerate delays and can compensate for loss by retransmission. Low priority can thus be assigned to these applications. These networks are called B-ISDN (Broadband-Integrated Services Digital Networks) and the ATM (Asynchronous Transfer Mode) is considered to be the most promising information transfer technique for the developing of such networks. The multimedia applications of a B-ISDN will have diversified quality of service (*QOS*) requirements, while the traffic speed may vary from a few Kbps (slow terminals) up to several hundred Mbps (high quality moving images). The traffic statistical description may range from the high traffic burstiness of the video applications to the smooth continuous traffic generated by large file transfer.

An uncompressed video signal may result in a bit rate as high as hundred of Mbit/s. And even though in the last few years we have seen an astronomical increase in bandwidth availability, a few such video sources could easily fill all line capacities. Data compression techniques are currently employed to transmit coded full motion video within digital lines of significantly lower capacity. Several coding algorithms have been proposed for VBR traffic. A source characterisation method that models the properties of a large variety of sources is ideally required for analysis. One approach is to obtain video source models for coders that utilise a standard algorithm

that can be applied to a multitude of video sources, as the Motion Pictures Expert Group (MPEG) coding standard.

1.2 The co-ordinated project

In the framework of the Italian co-ordinated project entitled "Variable Bit Rate Traffic in an Interconnected Environment" we were involved in two main activities:
a) The study of a satellite access scheme supporting the transmission of the VBR traffic together with the Constant Bit Rate real-time traffic (stream traffic) and the standard non real-time traffic (datagram) deriving from bulk and interactive applications.
b) The study of the performance relevant to the transmission of VBR video MPEG coded traffic in an environment which would simulate as much as possible the satellite link. The real VBR data were supplied from another group of researchers at the CSTV centre in Turin, who worked on the MPEG video source coding.
This report presents one of the two solutions adopted for point b), i.e. the solution called "pattern sensitive". This solution carried out by means the VBR Send/Receive programs, allows the production of an output generated by MPEG coded video data after crossing the satellite link affected by error, and the convolutional encoder/Viterbi decoder.
The VBR Send/Receive programs work in two different modes:

1) the MPEG coded data are read from a file and sent, via dedicated Ethernet, to the satellite link handler at the maximum speed allowed by the mini-system Motorola Delta where the program runs (SEND). After crossing the satellite link, possibly affected by error, the data is recorded on the output file (RECEIVE).

2) A fixed data pattern is transmitted to the erred satellite link. On the receive side, the comparison between the transmitted and the received bit streams creates a map of bit in errors. Such a map, applied on the input sequence, produces the sequence of data corrupted by the satellite link crossing.

Some considerations must be done. The use of convolutional coder/Viterbi decoder allows high gains of Eb/No, but, on the other hand, it produces trains of erred bits which can made unacceptable the reproduction of the MPEG sequence in such a way corrupted. In order to improve this situation, additional techniques, such as the data interleaving and/or block codes can be used in order to reach very low BERs.

2. THE UNIX ENVIRONMENT

The VBR Send/Receive programs have been implemented in C language on a Motorola Delta 3300 single-board microcomputer running the UNIX System V 3.6. This machine features the following characteristics:

- VME bus interface,
- Motorola 68030 µp and 68882 math co-processor running at 25 MHz,
- 8 Mbyte memory,
- two independent counter-timers with 6.25 µs time resolution,
- Local Area Network Controller for Ethernet AM7990(LANCE) - Serial Interface Adapter AM7992(SIA) chip set for Ethernet interface,
- SCSI interface,
- 1.5 Mbyte/s transfer rate, 16.5 ms av. access time disk.

Facilities for interprocess communication (IPC) and networking were a major addition to the UNIX system. Socket-based IPC has been superseded as the "standard" method for accessing network protocols. Processes may communicate through a UNIX file system-like name space, as well as through a network name space. For more information about interprocess communication on the UNIX system and the system calls used to manipulate socket, see [1].

Send and *receive* programs provide fast transmission of a large amount of data from one host to another one, in a most efficient manner. Therefore, *raw socket* mechanism is chosen, as implementation.
Program execution parameters are explained in the following sections.

3. BASIC COMMUNICATION MECHANISM

The basic building block used for communication is the ***socket ()***. Each socket in use has a type and one or more processes associated. Socket exists within *communications domain*. Domains are abstractions which imply both an addressing structure (address family) and a set of protocols (protocol family).
Processes are presumed to communicate among sockets of the same type only. There are several types of sockets currently available.

• *Stream socket (SOCK_STREAM)* supplies the bi-directional, reliable, sequenced and unduplicated flow of data without record boundaries.

• *Datagram socket (SOCK_DGRAM)* supports bi-directional flow of data that is not promised to be sequenced, reliable or unduplicated.
• *Raw socket (SOCK_RAW)* provides user access to the underlying communication protocols which support socket abstraction.

We consider *raw sockets* in order to communicate directly with lower-level protocols. Raw sockets are connectionless interactions which are typical of the datagram socket.
Sockets exist within *communications domains*. The domain is specified as one of the manifest constants defined in the <sys/socket.h> include file. For the UNIX domain the constant is AF_UNIX; for the Internet domain it is AF_INET. The socket system call is used as follows:

s = socket (communication domain, socket type, protocol)

To create a *raw socket in the Internet domain* the following call is used:

s = socket (AF_INET, SOCK_RAW, protocol).

The characteristics of the raw socket depend on the interface provided by the protocol. For the default protocol (a value 0), the system selects an appropriate protocol. This is usually correct; however, the protocol argument may be important when using raw sockets to communicate directly with lower-level protocols or hardware interfaces.

4. DATA TRANSFER

There are a lot of possible calls to send and receive data. The client-server model (using in developing applications), which is commonly used in applications, is chosen here as well.
We first consider the properties of *server* processes. A socket is created without name. Until a name is associated to a socket, processes have no way to reference it and, consequently, no message may be received on it. Communicating processes are bound by an association. These associations are composed of local and foreign address, and local and foreign ports. Through the **bind()** system call a process may specify half of an association.
The bind() system call is used as follows:

bind (s, name, namelen)

The bound name is a variable length byte string that is interpreted by the supporting protocol(s). Its interpretation may vary with the communication domains. In our case, Internet domain names contain an Internet address (local address) and port number (local port). The bind() call is required to insure that the server listens at its expected location.
After the name is bound, normal **read()** system call is issued to receive messages.

We will consider now the *client* processes. While processes are still likely to be client and server, there is no requirement for connection establishment. Instead, each message includes the destination address. Our application deals directly with addresses. This allows services to be developed as much as possible in a network independent fashion. To send data, the **sendto()** primitive is used:

*sendto (s, buf, buflen, flags, (struct sockaddr *)&to, tolen)*

The *s, buf, buflen*, and *flags* are user-defined parameters. The *to* and *tolen* values are used to indicate the address of the recipient of the message.

5. HOW TO START PROGRAMS

In order to enable communications, firstly we must run the receive side, as a server side. Notice that as we are using raw type of the socket, user should have super-user privileges. Only privileged sockets may bind addresses in privileged portions of an address space (lower levels of the network).

receive [options] hostname [outputfile] [errfile]

where:

hostname	is the IP source address, expressed by name.
[outputfile]	output file where to copy the data received.
[errfile]	output file where to list the bits in error.
[options]	are:
	- <pattern type [f/i/r]> <pattern value>
	means that if no [datafile] is specified in the send command of the client side, the packet must be filled with pattern. The

pattern is specified in <value> and the use of this parameter depends on the <type> option:

- f packet is filled with word length (4 bytes) value.
- i packet is filled with a value incremented word by word.
- r packet is filled with random words (value is the seed).

-b compute wrong bit
-s save all records received
-l record wrong bits
-h help (print this page).

After this command is properly executed, the server side is ready for reception. Note that the receive side is activated in the background.
If the receive program execution is interrupted with ctrl<d>, the following information is displayed:

- Total run time [sec]
- Total number of packets
- Total number of delayed packets
- Maximum delay [ms]

When the client side is activated by executing the send command, messages from <datafile> or <pattern> are sent.
Notice that as we are using raw type of the socket, the user must have super-user privileges.

send [options] hostname timefile [datafile]

where:

hostname is the IP source address, expressed by name.
timefile is the input file containing the timetable.
[datafile] is the input file containing the data to be sent.
[options] are:
- s <packet size> is the packet size in bytes.
- m <multiplier> multiplies the timetable entry intervals. One (1) means that times are expressed in milliseconds.
- <pattern type [f/i/r]> <pattern value>

means that if no <datafile> is specified, the packet must be filled with pattern. The pattern is specified in <value> and the use of this parameter depends on the <type> option:

- f packet is filled with word length (4 bytes) value.
- i packet is filled with a value incremented word by word.
- r packet is filled with random words (value is seed).

- t error generator.
- h help (prints this page).

If the BER (bit error rate) has to be calculated, the options ***b*** (compute wrong bit) and *l* (list/record wrong bit) must be specified in the command line on the receive side. Specifying the *b* option only, the number of wrong bits calculated is printed at the end of the transmission. Alternatively, they can be displayed immediately by issuing of SIGINT signal (press ctrl<d>). If *l* option is specified, wrong bits are recorded. In addition, the type of the pattern MUST be specified as well on both the sides, choosing among three options:

- f fixed pattern
- i incremental pattern
- r random pattern

When optional <errfile> is specified, the user MUST define *pattern <type>* and *<value>*, *b* and *l* options to generate valid traffic trace statistics. If one of these parameters is missing, the resulting statistics are meaningless.

Notice that <type> and <value> parameters of the pattern chosen on the receiving side MUST match exactly the <type> and <value> parameters of pattern on the sending side.

Any error in the command line is flagged with one of the following error messages:

•*Host name must be specified.*

The <hostname> MUST be specified.

•*Unknown host.*

The <hostname> must be recognised in the UNIX */etc./hosts* list.

•*Output file must be specified.*

s option is chosen but <outputfile> is not specified.

•*Error file must be specified.*

l option is chosen but <errorfile> is not specified.

•*Input timetable file must be specified.*

<Timetable> file MUST be specified in a command line.

•*One only of <pattern> and <datafile> can be specified.*

If <datafile> is not specified, then the pattern <type> and <value> MUST be specified.

•*Input data file must be specified.*

<datafile> or <pattern> must be specified.

Other messages are self-explanatory; therefore they are not specified here.
It is important to note that each error message appearance indicates the program termination as well.

REFERENCES

1. System V/68, Release 3, System Administrator's Guide (TCP/IP Transport provider procedures; Basic network procedures).

2. Nedo Celandroni, Marco Conti, Erina Ferro, Enrico Gregori, Francesco Potortì *"A Bandwidth Assignment Algorithm On A Satellite Channel For VBR Traffic"*, to appear on the International Journal on Satellite Communications.