This work has been published on Real-Time Systems, Vol. 13, No. 1, pp. 50-55, July 1997

Copyright ©1997, Kluwer Academic Publishers, <URL:http://www.kluweronline.com/>

# gA TRAFFIC GENERATOR FOR TESTING COMMUNICATION SYSTEMS: PRESENTATION, IMPLEMENTATION AND PERFORMANCE

Nedo Celandroni, Erina Ferro, Francesco Potortì

CNUCE, Institute of National Research Council
Via S. Maria 36 - 56126 Pisa - Italy
Phone: +39-50-593207/593312/593203
Fax: +39-50-904052 - Telex: 500371
Email: E.Ferro@cnuce.cnr.it
N.Celandroni@cnuce.cnr.it
F.Potorti@cnuce.cnr.it

## Abstract

This paper presents a multi-application traffic generator (MTG), aimed at the generation of packets over a LAN. The generated traffic simulates the one produced by a number of both isochronous and anisochronous applications, thus allowing the measurement of a number of parameters relevant to the communication network. From a test point of view, data generated by the MTG system is equivalent to data generated by real applications spread over a LAN.
The MTG system is presented, its implementation is described, some figures relevant to the MTG performance are shown, and the statistical analysis which can be performed on the recorded data is briefly introduced. The user manual of the MTG system is referenced in [20].

## 1. PRESENTATION OF MTG

The Multi-application Traffic Generator (MTG) presented in this paper was originally conceived as a tool for the testing and the performance evaluation of a satellite access scheme able to support the requirements of different types of traffic even in bad atmospheric conditions. On the other hand, the modular architecture of MTG also makes the evaluation of other communication systems possible, provided that some parts of the system are replaced.
The satellite access scheme that was tested by means of MTG is FODA/IBEA-TDMA(1) [5]. It allows the simultaneous transmission of real-time data (or stream) and non real-time data (or datagram) sharing the use in TDMA of a satellite channel. In addition, FODA/IBEA is able to counter signal fade due to bad atmospheric conditions, by dynamically adapting the data bit and coding rates to the channel state. FODA/IBEA runs on two separate Motorola 68030 CPUs (Up and Down TDMA controllers, respectively), with a low level of interaction between them. To access the FODA/IBEA system a protocol, named GAFO(2), was developed [6]. Thanks also to the tune-up made by means of MTG, the FODA/IBEA system

---

(1)FIFO Ordered Demand Assignment-Time Division Multiple Access

(2) GAteway-FOda/ibea

was successfully used in a real environment between three Italian stations connected via the Italsat satellite [17]. An example of the real environment at each earth station where the system is used is shown in Fig. 1.

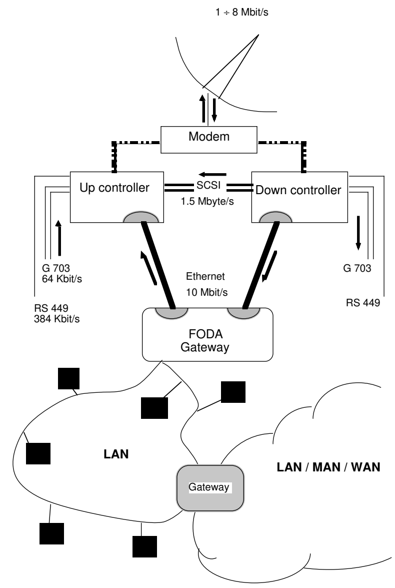


Fig. 1. Example of a real environment at an earth station

The testing and tune-up of complex communication systems in the real environment entails the problem of controlling the characteristics and allowing the reproducibility of the traffic that feeds the system during the performance evaluation process. Generally, a simulation approach is adopted, which requires a detailed formal description of the communication system being tested along with the traffic pattern that feeds the system. However, even if in-depth detail is used in the simulation of the system, it is practically impossible to take into account all the factors present in a real environment.

The environment simulation technique is the most realistic test technique for telecommunication systems. In fact, there is no approximation with regard to system behaviour. Moreover, the traffic generator can be used during the debugging phase of the communication system realisation.

A number of traffic simulation approaches can be found in the literature [1, 9, 10, 11], but most of them are dedicated simulators that can only be applied to a particular system. Other approaches deal with more system-independent concepts, but they do not allow the simultaneous generation of stream and datagram traffic. UNES [9], for example, is a versatile environment simulator for load tests of switching system software, but it is designed for telephone switching systems only [10]. ATGA [1] is one of the newest traffic generator analyzers available on the market but, unfortunately, it only supports tests and measurements in an ATM environment.

The aim of MTG is to develop a modular, flexible and very cheap test system. In fact, the tool is only software-based and uses a general purpose UNIX machine. Figure 2 shows the environment of the simulation where it was used in a loop-back configuration. Data generated by MTG enters the FODA/IBEA system via a dedicated Ethernet LAN, it is transmitted to the satellite, received back and addressed to the same MTG which originated it. After it has crossed the entire network, a selected part of the data returned is recorded onto disk, so that it can be used to produce statistics related to the FODA/IBEA system performance. Both MTG and the FODA/IBEA software implement the GAFO protocol.

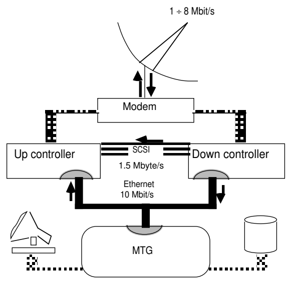


Fig. 2. The simulation environment

In very general terms, MTG can be described as a real-time system capable of generating traffic according to user specifications and detecting it back in order to evaluate the performance of a specific communication system, in our case the satellite network. The effects that the satellite access scheme used has on the data delays, the throughput and the congestion can be analysed by using MTG.

## 2. CHARACTERISTICS OF THE TRAFFIC GENERATED

Datagram traffic is a connectionless type of traffic without any particular delay requirement. It can tolerate out-of-order delivery of packets and a high jitter, but it usually accepts a low bit error rate. The delay caused by the network(s) crossing is not a critical constraint. However, especially on a high delay network, such as a satellite network, the end-to-end throughput of such traffic may be heavily impaired by bit errors or packet losses. Two priority levels of the datagram traffic are envisaged. The high priority datagram is called *interactive*, because its natural application is conceived for interactive computer sessions. The low priority datagram, called *bulk*, can be used for applications such as file transfer. Datagram packet delivery is not guaranteed since the datagram transmissions may be momentarily suspended (for example, when a congestion is detected), and packets exceeding the system buffering capacity may be dropped. Moreover, the delay a datagram packet experiences may occasionally be very high, in the order of seconds.

By *stream* we mean connection-oriented applications characterised by a constant packet arrival rate. These applications typically require short and fairly constant delays, they cannot tolerate out-of-order delivery of packets, but can tolerate occasional bit errors and dropped packets. In practice, stream traffic needs a fixed amount of bandwidth and a low and constant delay in the arrival of the information should be maintained. In addition, we subdivide stream traffic into Constant Bit Rate (*CBR*) traffic and Variable Bit Rate (*VBR*) traffic. CBR is a traffic generated by applications that have a fairly constant throughput (voice, slow-scan TV, fixed rate video conference, measurement data and so on); it needs a guaranteed bandwidth and a fairly constant transmission delay. A sub-class of CBR is the Reducible Bit Rate (*RBR*) traffic. An RBR application generates a fixed bit rate traffic, but can accept a request to temporarily reduce its throughput. Most of the H.261/H.263 video codecs are an example of this since their throughput is fixed but changeable on command.

An uncompressed video source may generate bits at rates as high as hundreds of Mbps. Data compression techniques are therefore used to reduce the video source bit rate which is transmitted over the network. The two major compression standards for VBR video used today are JPEG and MPEG-2. JPEG is an intraframe coder, which means that only redundancies within a frame are removed. An interframe coder, such as MPEG-2, removes redundancies both within a frame and between adjacent frames.

## 3. IMPLEMENTATION

MTG produces a mixed traffic that stems from the simultaneous activity of a number of different user-defined independent sources called *Traffic Generators* (TGs). Thanks to the combination of the various types of traffic simultaneously generated by the TGs, it is possible to simulate arbitrary complex traffic patterns, using a comparatively small set of parameters. Each TG has its unique set of characteristics, such as the type of traffic generated (voice, Poisson, fixed rate, fractal, 2-state Markov modulated Poisson, etc.), destination address, generation time distribution, generation time jitter, packet length distribution and contents pattern. A set of characteristics, described in an *input descriptor file* (IDF), defines the behaviour of each TG. The number of simultaneously active TGs is only limited by the performance of the machine where MTG runs.

The MTG system has been implemented in C language on a Motorola Delta 3300 single-board microcomputer running the UNIX System V 3.6. This machine has the following features:

- 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.

The choice of this machine was dictated by practical reasons only. On the TDMA controller of each earth station, an additional CPU board, equipped with a small disk, was installed during the test and tune-up phase, in order to have the development tool kit of the FODA/IBEA system on hand. MTG runs on the same board, thus requiring no additional hardware.

The UNIX system provides a convenient environment for the programmer, but poses fundamental obstacles because of its non real-time behaviour. Since System V currently provides no standard means to overcome this problem, a brute force approach was used. The devices related to MTG (the LAN manager and the timer) were forced to generate non-maskable interrupts. While leaving the operating system free to carry out its work, this technique allows a full use of the processor's power. This is important because MTG requests operating system services when it writes on disk (for off-line statistical analysis) the header part of the data packets that are returned after transmission to the satellite. There is practically no disk overhead; in fact, the MTG performance is independent of the disk speed, thanks to the UNIX I/O buffering, whose parameters were accurately chosen.

Real-time behaviour is vital to MTG because of the precision required by the packet dispatching times and the timestamps contained in the MTG header. These timestamps allow an accurate estimate of parameters of the system being tested.

MTG consists of two distinct sections: the *MTG driver* and the *MTG controller*.

The MTG driver is embedded in the UNIX kernel, running in supervisor mode, while the MTG controller is a super-user process. The two sections interact via the standard UNIX driver interface and a shared memory area.

The driver part depends on the LAN used. It is related to the management of the low-level interface with the hardware. It is responsible for the precision of the delivery times and the timestamps of the packets. Communication between the controller and the driver is implemented via standard *ioctl* system calls. The function *ioctl* performs a wide variety of control functions on the devices. The meaning of the argument is driver-specific. Six different types of commands are provided for starting or stopping a TG, sending a control message, suspending or resuming a TG, and changing the throughput of a TG.

The controller part is related to both the user interface (since MTG is an interactive program) and the operating system interface. It is responsible for: a) the start-up procedures, where the input descriptor file is read and the TG tables, containing the transmission times of the packets, are computed, b) the real-time display of the events, and c) the writing of the packet headers on disk.

### 3.1. The MTG controller.

After reading the input data file, the controller gives an *allocate* command to the driver, which allocates inside the kernel the specified amount of contiguous physical memory and gives back the starting physical address to the controller. The controller maps that physical space onto its own virtual space as a shared memory segment, which will be used as a receive buffer: both the data and control message queues are located here. Then, for each TG, the controller allocates a *Traffic Generator table Descriptor* (TGD) which points to the table containing the dispatching times and the lengths of the packets associated with that TG. Both the time intervals between two consecutive packet dispatches and the packet lengths are calculated by the controller, during the starting phase, according to the selected distribution of each TG. For each TG, after building the table and the TGD, the controller sets a start and a stop Unix timer. When a *TG-start* timer expires, the relevant TGD is handed to the driver using the *start* command. In order to allow the interrupt handlers inside the driver to access the TGD using physical addresses, the controller allocates a TGD so that it does not cross a virtual memory page boundary. The driver then builds its own table of the dispatching times and packet lengths, and inserts the TGD into the TGD linked list (see Fig. 3), which is scanned once per tick (1ms) in order to send the packets on the LAN at the times scheduled. When the *TG-stop* timer expires, the controller kills the TG by commanding the driver (*kill* command) to cancel the relevant TGD and its associated table.

The controller is also responsible for handling the GAFO protocol. It builds the GAFO control messages and, using the *ctrlmsg* command, commands the driver to send them to the net. The GAFO control messages received from the net are organised by the driver into a specialised *control-and-error queue* (which is different from the *data packet queue* that it puts other packets in) to achieve fast response times. The controller continuously monitors both these queues, which it does not have write permission for, by polling their current pointers. This method allows the fastest possible response time to the reception of a packet, because no system call nor task switch is ever required in order to process a received packet. However it does not waste resources, because the controller is the only active process in the machine. In fact, the other subsystems working concurrently with the controller are all interrupt-driven, and run inside the kernel.

The control-and-error queue contains the occasional GAFO protocol messages received from the net and the error messages internally generated by the driver: it is generally sized for a capacity of 20 packets. The data packet queue contains all the data packets arriving from the net. Its size can be chosen at the initialisation time, and can reasonably vary from a few hundred KB up to the system memory limit. The size of this memory area affects the performance of MTG on the receive side. The operations the controller makes on each packet received involve at least classifying the packet and checking its sequence number, and may also include various optional tasks. Such tasks are: registration on disk of the MTG header of the packet or of the whole packet; real time checking of the integrity of the data received with a count of the number of wrong bits; or recording on disk the list of wrong bits. Depending on the operations requested, the CPU might not be able to fulfil this task during traffic peaks without a big receive buffer.

MTG can write to disk the headers of the packet received for later statistical analysis. It is interesting to note that disk speed does not affect system performance, because the rate at which the writing happens is generally lower than the disk's physical data throughput. The only significant overhead is caused by the Unix buffering subsystem. In order to get the maximum performance from it, we use a dedicated disk slice, which eliminates all the file system access overheads, and uses the normal *fopen*, *fwrite* and *fclose* calls to the library, after setting the buffer size with *setvbuf* to an operator-selectable value (32KB seems to perform best). A character device is used to access the disk slice. While this may seem counterproductive (as opposed to using a block device), it neverthless improves performance if the buffer size is chosen to be a multiple of the Unix disk buffer size and is aligned on a virtual page boundary. This happens because many Unix kernels don't copy data from the process buffer to system buffers if the above conditions are satisfied, thus obtaining a higher processing speed.

The controller also polls the keyboard for operator actions. In fact, the controller gives a real-time overview of the active TGs and their parameters, their current status, the missing sequence numbers on reception, the current bit error rate of each TG, and to start or stop any TG from a terminal, where the numbers are constantly updated using the standard *curses* library.

### 3.2. The MTG driver.

The driver is very specialised: no *read* or *write* operations are provided for general use. The controller gives commands to the driver via the UNIX *ioctl(2)* system call and receives packets in a memory area mapped onto its virtual space. The driver's task is to read the tables created by the controller and to send packets to the network accordingly, while respecting the scheduling times as much as possible.

The scheduling times are currently specified with a 1 ms (one *tick*) resolution, while the timestamps have a 100 µs resolution (one *microtick*). Timestamp resolution can be reduced at will by increasing the CPU power.

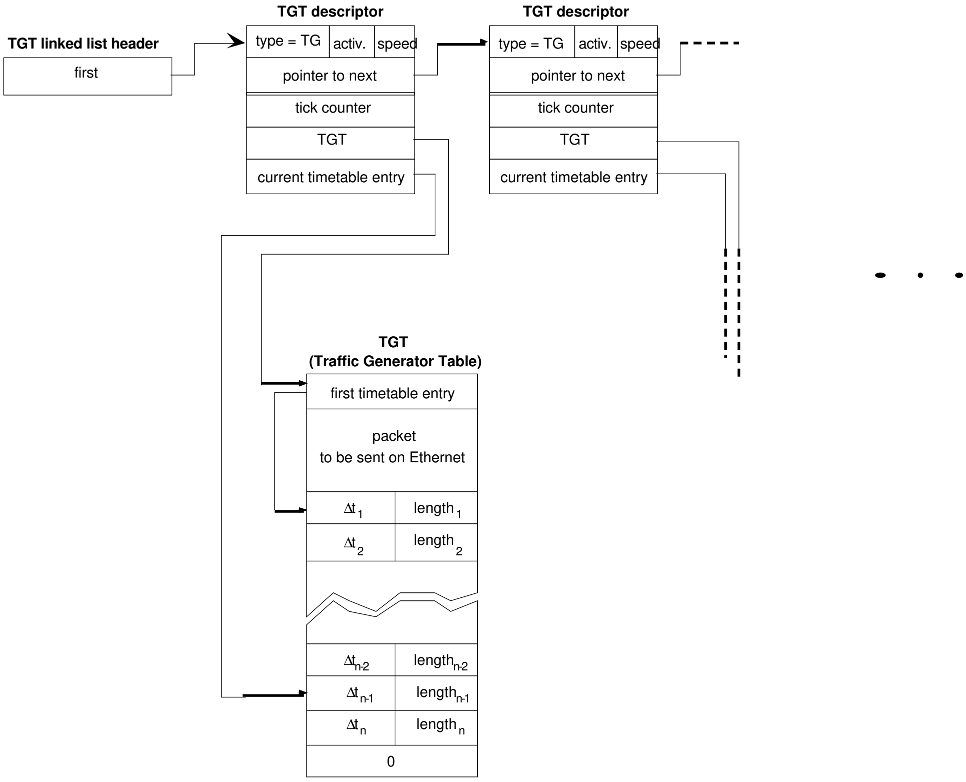


Figure 3. The organisation of the tables

Figure 3 depicts the organisation of the tables inside the driver. The driver is mainly composed from a system interface that manages the *ioctl* processing, and two interrupt handlers, one for the LANCE (the chip that manages the Ethernet) and one for the clock. Both devices generate level 7 (non-maskable) interrupts, in order to achieve predictable response times independently of what the Unix kernel is doing at the time. For this structure to work, none of the interrupt handlers must access any of the kernel areas, nor request any kernel service. Moreover, they must be reentrant, since the interrupts are non-maskable. The latter requirement has been fulfilled through the use of a private semaphore. The whole driver is written in carefully optimised C, mixed with some assembly lines for the semaphore handling and the hardware interrupt interfaces. The microtick interrupt handler is also written in assembly language, because it is called every 100µs, efficiency was thus deemed important. The microtick handler increases the microtick counter at each invocation and, once every ten times (that is, every ms), it increments the semaphore value by one and, if the semaphore is null after the increment, it calls the scan routine (which is written in C language).

If the semaphore is non negative, the scan routine walks through the TGD list, looking for packets to dispatch. For each active TG in the list, the *tick counter* in its TGD is decreased by one. If the result is a non-null value, the next TGD is examined. At the end of each scan, when all the tick counters have been decreased by one, the semaphore is also decreased by one; if it is non negative, another scan is initiated.

The semaphore thus acts as a counter of how many scans of backlog the driver has. Its value in any instant (unless negative) is the delay in ticks the packets to be sent are experiencing with respect to the scheduled sending times. If the semaphore value is too high, the scheduling of the packets transmitted may differ significantly from what is written in the tables. That's why the driver sends an error message to the controller when the semaphore exceeds a threshold set during the initialisation phase. The controller takes the threshold value from the init file.

When the tick counter in one of the TGDs reaches zero, the pointer to the current timetable entry is advanced (the timetable is circular) and the $\Delta t$ field is copied into the tick counter field. The first *length* bytes of the current packet of that timetable are sent to the LAN. This is accomplished simply by setting a bit in one of the structures used by the LANCE chip, and by issuing a hasten command to the chip, which would otherwise poll that bit only every 1.6ms. The scan is then suspended until the occurrence of the *end of transmission* hardware interrupt. When the end of transmission interrupt is received, the scan routine continues from where it had left off.
Although the LANCE chip would allow the packets to be queued for transmission, the driver handles the packets one at a time. This leads to an overhead of about 60 µs (with the present CPU) on each packet sent, which prevents back-to-back packet transmission, but allows MTG to set accurate transmission timestamps. Indeed, the microtick counter is read and written on the packet to be sent just prior to giving the buffer to the LANCE. Unless the LANCE has to make various attempts at transmitting because of collisions, the timestamp so set is perfectly accurate. A counter permits to monitor whether there are excessive delays due to collisions, that is, if the timestamps are really accurate (which normally happens).
In order to make the scan routine as efficient as possible, it has been coded so that it never reads any memory areas written either by the LANCE chip or by a user process. This avoids cache consistency problems, and allows the use of physical addresses, which bypass the MMU and are therefore faster. The efficiency of the scan routine is important, because the overhead per packet sent depends solely on it.

The LANCE interrupt driver also takes care of the reception of packets from the net. Each packet received is timestamped and put in the data packets queue, and the pointer to the current packet is advanced. No explicit communication is given to the controller, thus avoiding any system overhead. The controller polls the current pointer to know when a packet has arrived. As already noted, since the driver's work is driven by interrupts, this method does not slow down the system.

### 3.3. The TG parameters

In the same run, different types of traffic can be generated simultaneously. Each TG is described by a number of parameters (Fig. 4) , some of which depend on the network being tested, in our case the FODA/IBEA satellite network.

| | | | | |
|---|---|---|---|---|
| tgid n | data t | dist t | pkts t n1 n2 | thro n1 n2 |
| patt t n | time t n1 n2 | qlen n | jitt n | cos n |
| addr t n | seed n | inta n1 n2 | nowb nrpol n | |

Fig. 4. Definition of each TG

The mnemonic name of each variable used is followed by one or more values.

tgid *n* is the traffic generator number.

data *t* is the data type. It can be *stream, bulk* or *interactive*. Interactive traffic gets higher transmission priority than bulk traffic.

dist *t* is the traffic distribution type of this TG. It can be *fixed-rate, random* (uniform distribution), *Poisson* (negative exponential distribution), *2-state Markov-modulated Poisson*, *fractal* (fractional Gaussian noise), *voice* (the talk-silent periods are sorted according to a statistical distribution relative to the English language) or *file* (packet transmission times are read from a file on disk).

pkts *t* is the type of the generated packets. It can be *fixed-size*, *variable-size* or *computed* (i.e. the packet length is computed by MTG using the *inta* and the *thro* parameters). *n1* is the maximum packet size and *n2* the minimum packet size (in bytes).

thro is the throughput. *n1* is the requested throughput and *n2* is the minimum throughput this TG can deliver (to simulate variable-rate applications).

patt *t* is the data pattern type. It can be *fixed* (equal to the chosen value expressed by *n*), *random* [a random number in the range $0 \div (2^{16}-1)$] or *incremental* (the pattern is incremented one byte at a time, starting from an initial value expressed by *n*).

time *t* is the time type. It can be *relative* to the MTG start time, *absolute* (local time), *user* (on user command) or *Poisson*. The latter is an endless sequence of Poissonian starts with Poissonian duration. It is used to simulate phone-call traffic. *n1* is the start time or the mean call time; *n2* is the duration or the mean call duration time.

qlen *n* is the maximum number of packets the system being tested is allowed to maintain in the queue of the packets waiting for transmission.

| | |
|---|---|
| jitt | packets are generated with an initial jitter. A further delay between 0 and *n* ms, sampled from a uniform distribution, is added to each packet's dispatching time; *n* equal to zero means no jitter. |
| cos | *n* is the number of the class of service associated to the data generated by this TG. This parameter depends on the network under test. In our case, classes 1-4 specify that the data bit error rate (BER) must be kept within specified limits:<br>n =1 $\quad BER \le 10^{-8}$<br>n = 2 $\quad 10^{-8} \le BER \le 3 \cdot 10^{-7}$<br>n = 3 $\quad 3 \cdot 10^{-7} \le BER \le 3 \cdot 10^{-5}$<br>n = 4 $\quad 3 \cdot 10^{-5} \le BER \le 3 \cdot 10^{-3}$<br>Higher classes (5-20) are used to force the bit and coding rates which FODA/IBEA uses to send the data in faded conditions. |
| addr | *t* is the address type (to *myself, broadcast* or to a *selected station*); *n* is the destination address. |
| seed | *n* is the value of the seed for the random number generation to build the tables for this TG. |
| inta | *n1* is the minimum and *n2* the maximum packet inter-arrival time (in ms), respectively. If both the *pkts* and *thro* fields are specified, this field is ignored. |
| nowb | *n* is the number of wrong bits that the relevant TG must collect before displaying (in real time) the computed BER at the terminal. |
| rpol | is the policy adopted by the TG to compute the bandwidth request. For stream applications the request is computed on the *minimum* value (of *thro*) plus a percentage (expressed by *n*) of the difference between the maximum and the minimum value of the throughput. |

An *MTG header* is put on top of each data packet generated, and contains a number of measurement parameters. In this header four timestamps are reserved for the network being tested. In our case they allow the measurement of:

- the overall satellite network delay,
- the queueing times of the packets waiting for transmission to the satellite (on the Tx side),
- the queueing times of the packets waiting for transmission to Ethernet (on the Rx side),
- the packet jitter before entering and after exiting the FODA/IBEA system.

Two MTG timestamps allow the measurement of the MTG end-to-end delay and of the overall jitter (including the back and forth Ethernet crossing). The MTG end-to-end delay is the delay experienced by a packet since it leaves the generator until it is received back for recording. When many MTGs work together on LANs located at different FODA/IBEA stations, the FODA/IBEA timestamps are still significant, because they are coherent with each other thanks to the master station synchronisation.

The measurement of the jitter is significant for fixed rate traffic where the packet inter-generation time is constant. It is possible to retrace the history of the delay and of the jitter, for each packet, at several different test points of the system on trial, for example the generation of the packet by MTG, the arrival of the packet to MTG after traversing the network being tested, and four more network-specific points. For the FODA/IBEA system we tested four other testpoints: the arrival of the packet at the transmit controller, the time of transmission to the satellite, the reception from satellite, and the transmission on the LAN by the receive controller.
Six bytes are reserved for the network being tested. We used four of them to contain information on the transmit signal, i.e. to check whether, during a fade, the satellite access scheme adequately adapts the bit and coding rates to the computed signal-to-noise ratio. The other two bytes allow the estimation of the BER.

## 4. THE RANDOM NUMBER GENERATOR

Any simulation tool needs samples of quantities defined by their probability distribution. The samples are usually the output of deterministic mathematical algorithms known as pseudo-random number generators.
All the random numbers used within MTG are generated off-line. This means that a table for each TG is created at MTG start-up (before the simulation run) which contains entries filled with the starting time and length of each packet.
Several approaches can be found in the literature. Ramshaw and Amer [10] built a test system based on eight traffic generators for the NBSNET at the National Bureau of Standards. Each of the traffic generators — based on an 8-bit microcomputer — has a maximum throughput of one packet per millisecond. This constraint leaves 400µs for the uniform-distributed random number generation and for the mapping to the desired distribution. Unfortunately, this approach is not possible for MTG. Indeed, MTG can run a number of concurrent TGs, each able to generate packets as fast as one per millisecond, resulting in a total throughput of up to several packets per millisecond. At the same time, MTG receives the looped-back traffic and writes the packet headers onto disk. The all-software architecture of MTG thus makes on-line generation of the random numbers impossible, due to the extra overhead that would be imposed on the CPU.
With our table-based approach it is possible to choose a good generator and to transform the uniform distributed random number sequence into an arbitrary distribution without worrying about computing times. The main advantage is that at run time only a number in a table has to be read rather than an on-line generation. However, a significant disadvantage is that the period of the generated sequence is equal to the length of the table.
The uniform random number generator that we have adopted is a Lehmer generator with modulus $2^{31}$-1 and multiplier 950706376 [8].

The choice of the table lengths is not a secondary problem, since the goodness of the generator can be impaired significantly by the table-based approach. In any case, no time constant in the FODA/IBEA system exceeds a double round-trip delay, which is less than two seconds. Since each table entry requires 8 bytes and the maximum TG throughput is one packet per millisecond, the overall memory requirement for the tables is less than 8 KB per TG per second. To give an idea of the memory requirements, for each MB of space dedicated to the tables, 8 TGs at full throughput are able to generate a traffic whose pattern repeats every 16 s, which is much longer than the maximum expected time constant within the FODA /IBEA system.

The UNIX system, that MTG is currently running on, has an 8 MB real memory space. Less than 3 MB are required by the operating system, the disk buffers and the executable MTG image in memory. Depending on the MTG working mode, 1 to 4 MB are required for the receive buffer, the rest (1 to 4 MB) are available for the tables.

In practice, when more than one TG is active, the period for the overall traffic is generally much longer than that of the individual generators. In fact, the overall period is equal to the least common multiple of the periods (in ms) of all the active TGs. Therefore, we decided to use one table for each TG rather than a unique table.

## 5. PACKET GENERATION TIME DISTRIBUTIONS

MTG provides a number of different packet generation time distributions, plus the possibility to read the transmission times of the packets from a file on disk. The distributions currently available are: Fixed Rate, Uniform, Poisson, Two-state Markov-modulated Poisson, Fractional Gaussian noise, and Voice. We considered fractional Gaussian noise since there seems to be evidence about the self similar nature of LAN traffic, whose behaviour is not captured by any of the commonly used traffic generators, such as Poisson or Markov-modulated Poisson [12]. Even the argument, once generally accepted, that the aggregate LAN traffic becomes less bursty as the number of traffic sources increases now seems to be far from reality. In fact, measures seem to show that the burstiness of LAN traffic does not decrease as the number of active traffic sources increases (the self-similarity discussed in [12]).

Generators with different packet generation time distributions can be mixed in any possible way. The packets generated are individually tagged, so focused performance analysis of the network being tested is possible under very broad patterns of traffic. A brief discussion about the properties of some of the available distributions follows.

*Poisson* traffic is simple and has attractive theoretical properties, which is why it was widely used until recently. Interarrival times in Poisson traffic have negative exponential distribution and are independent, so the number of packets in any time interval follows a Poisson distribution. The only parameter needed for characterising Poisson traffic is its mean throughput. For wide area traffic, Poisson traffic models well the arrival of user-initiated sessions [13], but it is not good for capturing the features of the data traffic within sessions. However, Poisson traffic was used to load the FODA/IBEA system for measurement purposes on the Italsat satellite, because at that time other traffic patterns were not so widely used. Poisson is a particularly well-behaved traffic, from a burstiness point of view. Moreover, it does not have long range dependence, as its autocorrelation function decreases exponentially. It can be considered as a sort of best case loading, and indeed our measures show that the protocols manage to make the best use of the channel under this kind of load [19].

The *Two-state Markov-modulated Poisson* traffic implemented still does not exhibit long range dependence, but it does exhibit high burstiness on average. It consists of two independent Poisson generators with different mean throughputs. A two-state Markov chain is used to choose which of the two generators is active at any given time; the two states are called the *high traffic* state and the *low traffic* state. This generator is defined by four parameters. We chose mean throughput, mean time of permanence in high and low traffic states, and the ratio of mean throughputs in the two states. For example, the parameters representing a worst-case load for satellite protocols from a burstiness point of view can be defined as follows: the high traffic state lasts for a mean of 0.5 seconds and the low traffic state for a mean of 2.5 seconds. The traffic generation mean rates are set to a 17 to 1 ratio, resulting in a burstiness (peak to mean value ratio) of about 5. High burstiness is observed in LAN-to-LAN communications, as can be seen in the Bellcore traces [14], shown for example in [15]. In fact, the length of the bursts are chosen so as to maintain the station request algorithm in a “continuously transient” state, where the input traffic jumps to a different value as soon as the station's request and its assignment have stabilised.

The fractal generator is an approximated *Fractional Gaussian Noise* generator implemented with a simple Random Midpoint Displacement algorithm. In [16] there is an initial analysis of the statistical properties of this algorithm, with results that we considered to be good enough for our purposes. This generator exhibits relatively low burstiness and a long-term correlation, which we truncated to about 10 minute of simulation time. Our input traffic traces are thus made of consecutive 10 minutes long independent batches. Fractional Gaussian noise is defined by three parameters, namely mean, peakedness, and a Hurst parameter. We used a Hurst parameter equal to 0.85, in line with the findings published in [12]. The peakedness, defined as the ratio between the variance and the traffic distribution mean values, was set equal for all stations. Based on the measurements reported in [14], many authors believe that fractional Gaussian noise is good for modelling the aggregate output of a great number of sources whose traffics exhibit long range dependency [12, 15, 16]. If the satellite network is used as a bridge through many LANs with high traffic between them, this modelling is probably a good fit for the real traffic.

MPEG-2 coded variable bit rate data cannot be generated in real time without the use of dedicated hardware, but MTG can generate fixed pattern data with inter-arrival times read from a *file on disk*. If pattern-dependent effects are neglected, this arrangement faithfully replicates a real VBR traffic—or any other kind of traffic for which a trace of the generation times and lengths of the packets is available. We used packet generation times derived from the MPEG coding of the movie "Il té nel deserto", that the CSTV(4) Institute, Torino, Italy provided us in the framework of a joint Italian project(5) on VBR traffic.

## 6. THE PERFORMANCE OF MTG

This section presents the performance of the MTG system independently of the communication system being tested. The FODA/IBEA system was thus replaced with an Ethernet responder which looped the generated data back to MTG. Table 1 presents the MTG performance under different test conditions. Each test was run for 300 seconds. Ethernet was used both in simplex and in full-duplex mode. Neither the maximum throughput (10 Mbit/s) nor the maximum number of packets per second (14,880) allowed by Ethernet, when used in simplex mode, can be reached, due to the architecture of the MTG transmit side described above.

An Ethernet cable connected MTG to the responder during all the tests in order to measure both the generation and recording capabilities of MTG. The traffic looped back by the responder was collected by MTG and the recording of the MTG header of each received packet was active. The responder was not used in the tests where only the performance of the traffic generation was investigated.

Tests 1-3 show the maximum number of packets per second that MTG can handle. Transmission only and transmission/reception with and without disk recording were the test conditions, respectively. The packet length does not influence the MTG performance, provided that Ethernet can be viewed as an infinite bandwidth medium; thus a very small packet length was used to reduce collisions. Tests 4 and 5 give the maximum number of supportable 64 Kbit/s applications with and without the packet header recording, respectively. The packet length was set to 256 bytes plus a 22-byte header containing the Ethernet and the GAFO headers.

In the last five tests the maximum total throughput reachable was investigated with a different number of running generators (see Fig. 5). The packet length was chosen as the maximum possible, in order to reduce the weight of the 60µs overhead that MTG imposes on each packet transmitted. Figure 5 clearly shows how the total throughput increases when the number of generators decreases, due to the overhead required by MTG for each traffic

---

(4) Centro Studi Televisivi , del Consiglio Nazionale delle Ricerche

(5) Gestione del traffico VBR in Ambiente Interconnesso

generator. The throughput with only one generator is smaller because this test was performed with a shorter packet.
The ability of the system to generate packets according to a chosen statistical distribution is shown in Figs. 6 and 7. The sending times of the packets generated by a Poisson generator with an average inter-arrival time of 32 ms (64 Kbit/s throughput) were collected and the probability density function was approximated using 10,000 samples. The generator ran alone (Fig. 6) or together with 15 other generators (Fig. 7) of the same distribution type and with throughputs ranging from 16 to 256 Kbit/s. The offsets from the theoretical curve (a straight line in a logarithmic scale) are limited in the significant portion of the distribution. Furthermore, it is evident that the presence of many generators does not significantly worsen the accuracy of the traffic pattern of each generator. The X-square goodness-of-fit test, performed for the cases shown in Figs. 6 and 7, indicated that the Poisson distribution can be accepted at a 10% level of significance.

In the current version, where the LAN used is Ethernet, there are two distinct minor causes of inaccuracy in the generation of traffic: architectural and implementative.
The first is that packet collision and arbitration on Ethernet is not simulated by MTG. This is an architectural issue, i.e. it stems from the overall design of MTG, and is the result of a precision versus simplicity trade-off, where the need for simplicity comes from the limited CPU power of the hardware used. To evaluate the impact of this inaccuracy, we take some results from Molle [18], where an in-depth analysis of the delay characteristics of the Ethernet protocol is made. Molle shows that for an offered Ethernet load of 5 Mb/s the mean packet delay is nearly constant and less than 1 ms, and that it remains lower than 10 ms for an offered load of less than 7 Mb/s. His considerations also suggest that the effects of the truncated binary exponential backoff algorithm used on Ethernet is very much reduced for offered loads in the region of 0-5 Mb/s. Since in this region the granularity of the scheduling times of MTG (1 ms) is as big as the mean delay introduced by Ethernet, it would be virtually impossible to take this effect into account in any case. By comparison with the results presented in [4] for the FODA/IBEA satellite access protocol, the delay introduced by Ethernet is two orders of magnitude smaller than the delay introduced by the satellite network for all workloads. This difference is primarily due to the delays involved in the arbitration algorithms, which are in the order of hundreds of milliseconds for the satellite network and tenths of microseconds for Ethernet. Thus, for our purposes, this kind of error is negligible for all workloads, and should be considered negligible for any purposes whenever the offered load does not exceed 5 Mbit/s.

| number of TGs | Inter-packet generation time for each TG | packet length (bytes) | traffic volume in TX | traffic volume in Rx | header per number of packets | test run number |
|---|---|---|---|---|---|---|
| | ms<br>512 Kbit/s per TG | | Mbit/s | Mbit/s | | |
| | ms<br>512 Kbit/s per TG | | Mbit/s | Mbit/s | | |
| | ms<br>512 Kbit/s per TG | | Mbit/s | simplex | | |
| | ms<br>69.5 Kbit/s per TG | 22+256 | Mbit/s | Mbit/s | | |
| | ms<br>69.5 Kbit/s per TG | 22+256 | Mbit/s | simplex | | |
| | ms<br>195 Kbit/s per TG | | Mbit/s | simplex | | |
| | ms<br>404 Kbit/s per TG | | Mbit/s | simplex | | |
| | ms<br>757 Kbit/s per TG | | Mbit/s | simplex | | |
| | ms<br>2345 Kbit/s per TG | | Mbit/s | simplex | | |

Table 1. Performance evaluation tests scenario

The second cause of inaccuracy is due to Ethernet collisions on the cable used by MTG both to transmit and receive data. The error introduced by this effect can be evaluated by examining the timestamps on the packets and the computation of the jitter of the time needed for a packet to reach the satellite transmitter since being generated by MTG. This error is observable only when Ethernet becomes significantly loaded. When such a high throughput is required and a high precision is needed, an extra Ethernet board must be added to the MTG hardware, in order to use two LAN stubs—one for transmission and one for reception.

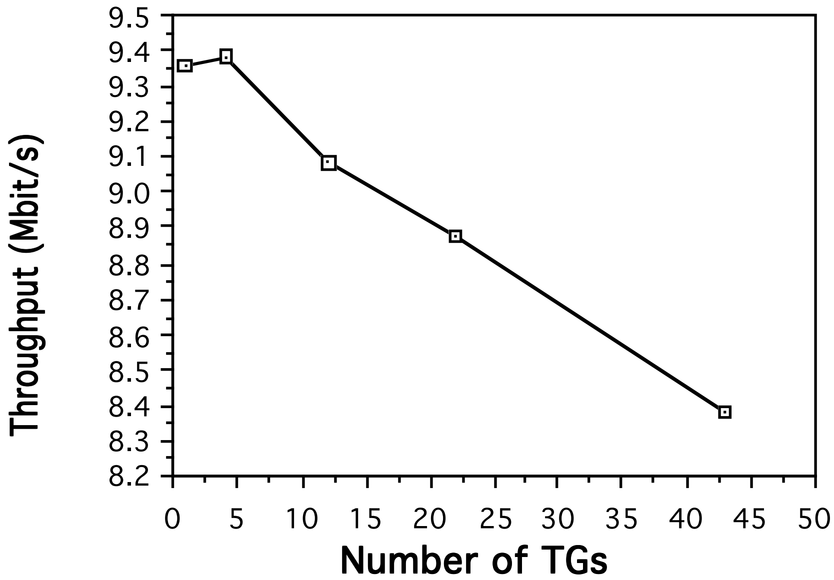


Fig. 5. Maximum throughput as a function of the number of TGs

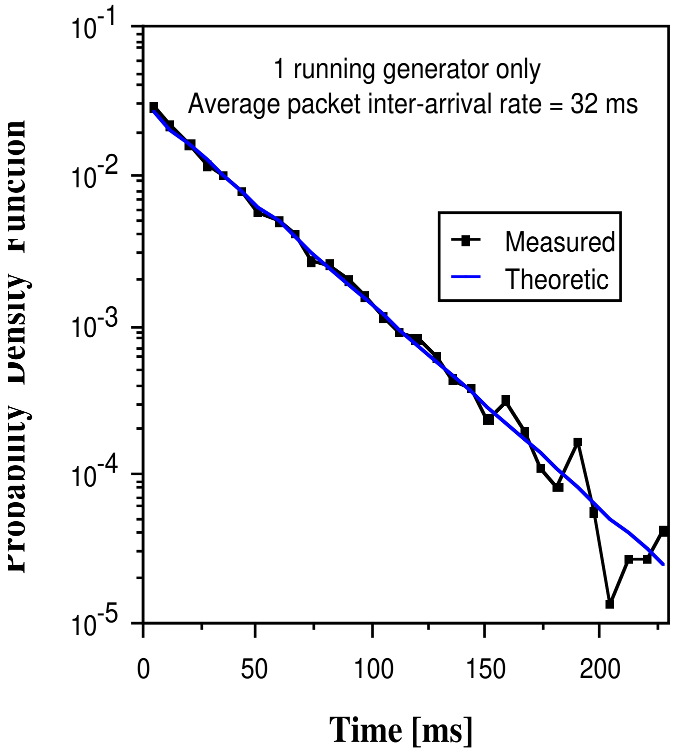


Fig. 6. Packet sending times distribution measured for one Poisson generator

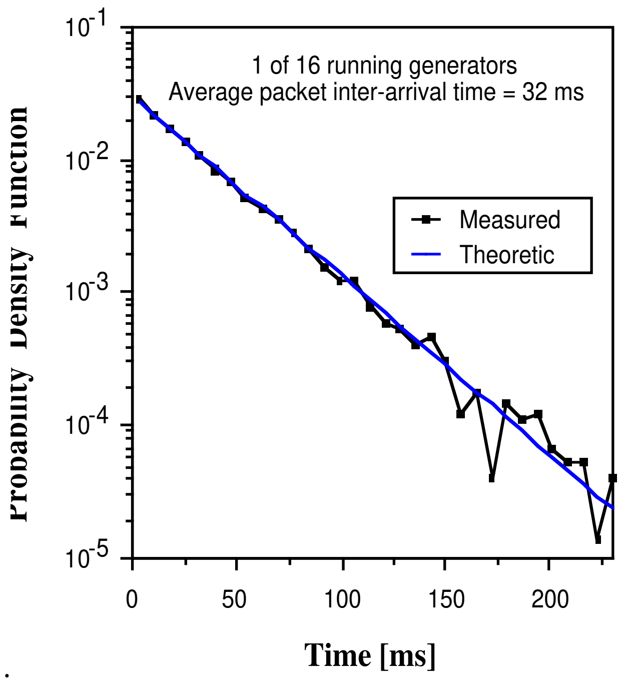


*Fig. 7. Packet sending times distribution measured for 1 of 16 Poisson generators*

## 7. THE ANALYSIS OF THE DATA COLLECTED BY MTG

Statistics on the data recorded onto disk by MTG are computed by means of a specialised statistical analysis program (MTGSAP), running under the UNIX operating system, which has an interactive user interface.

The following are the main functions of MTGSAP:

- Making a copy of the data recorded by MTG. This is mandatory before re-running MTG, because the output file *MTGout*, produced by MTG, is overwritten at each run.
- Filtering the data according to different parameters or making logical connections among the parameters themselves. An ASCII file is created containing the data relevant to the packets whose parameters match the specified conditions.
- After an interval of time has been selected, it is possible to get the maximum, minimum and average values, the variance, the probability mass function and the cumulative distribution function for any significant parameter specified in the MTG header or for the jitter of any time parameter.
- Computing the BER for each TG and the relevant standard deviation.
- Performing the CCITT G.821 analysis [3]. The following values are computed:
  - the interval of time where the analysis is made (in sec),
  - the total available time (in sec),
  - the percentages of severely errored seconds, errored seconds, and degraded minutes (with reference to the available time).

## 8. CONCLUSIONS

A sophisticated multi-application traffic generator has been presented, aimed at the test, tune up and performance evaluation of the FODA/IBEA-TDMA system. The flexibility of MTG in supporting different communication media (by changing the driver section) or different protocols (by changing the controller section) makes MTG a powerful traffic generator to test other types of communication systems. Moreover, many parameters of the MTG system, such as the accuracy of the time stamps and the granularity of the packet dispatch times, can be easily changed.

The UNIX system was chosen as a convenient environment for the programmer. The non real-time characteristics of UNIX do not affect MTG performance whatsoever, as shown in the figures. The performance can be improved by using hardware with a higher performance. In addition, MTG is cheap, simple to use from a user point of view, and has all the advantages of an interactive program.

Some examples of the possible statistics which can be performed on the data recorded by MTG have also been presented.

## REFERENCES


[1] Alcatel 8643
"ATM Traffic Generator. Analyzer ATGA", 1992.

[2] Carrapatoso E., Aguiar C., Ricardo M.
"Traffic generator/detector system concept", proceedings of the EFOC/LAN 90, pp. 119-123, Munich 27-29 June 1990.

[3] CCITT Red Book
"Digital Networks Transmission Systems and Multiplexing Equipment. Recommendations G700-G956", Volume III - Facicle III.3, pp. 310-317, Malaga-Torremolinos, 8-10 October 1984.

[4] Celandroni N., Ferro E.
"The FODA-TDMA satellite access scheme: presentation, study of the system and results", IEEE Transactions on Communications, Vol. 39, No. 12, pp. 1823-1831, December 1991.

[5] Celandroni N., Ferro E., James N., Potortì F.
"FODA/IBEA: a flexible fade countermeasure system in user oriented networks", International Journal of Satellite Communications, Vol. 10, No. 6, pp. 309-323, November-December 1992.

[6] Celandroni N., Ferro E.
"Protocol between the FODA system and the outside environment. The GA-FO protocol", CNUCE Report C91-21, November 1991.

[7] Consorzio Pisa Ricerche
"Multi-application Traffic Generator. Statistical Data Analysis Software Package MTGSAP, September 1993.

[8] Fishman G.S., Moore L.R.
"An exhaustive analysis of multiplicative congruential random number generators with modulus $2^{31}$-1", SIAM Vol. 7, No. 1, January 1986.

[9] Lemppenau W., Tran-Gia P.
"A Universal Environment Simulator for SPC Switching System Testing", proceedings of the 11th International Teletraffic Congress, Kyoto, 1985.

[10] Lemppenau W., Tran-Gia P.
"UNES: a versatile environment simulator for load tests of switching system software", proceedings of the ISS '87, 1987.

[11] Ramshaw L.A., Amer P.D.
"Generating artificial traffic over a local area network using random number generators", Computer Networks, Vol.7, No. 4, pp. 233-251, August 1983.

[12] Will E. Leland, Murad S. Taqqu, Walter Willinger, Daniel V. Wilson
"On the self-similar nature of Ethernet Traffic (extended version)", IEEE/ACM Transactions on Networking, Vol. 2, No. 1, February 1994.

[13] Vern Paxon, Sally Floyd
"Wide Area Traffic: the failure of Poisson Modelling", IEEE/ACM Transactions on Networking, Vol. 3, No. 3, June 1995.

[14] Samples of external Ethernet traffic measured at Bellcore during October 1989 are available with anonymous FTP from flash.bellcore.com, directory pub/lan_traffic/OctExt.TL.

[15] Ikka Norros
“The management of Large Flows of Connectionless Traffic on the Basis of Self-Similar Modelling”, ICC ’95, Seattle, Washington USA, June 18-22 1995, pp. 451-455.

[16] Wing-Cheong Lau, Ashok Erramilli, Jonathan L.Wang, Walter Willinger
“Self-Similar Traffic Generation: The Random Midpoint displacement Algorithm and Its Properties”, ICC ’95, Seattle, Washington USA, June 18-22 1995, pp. 466-472.

[17] N. Celandroni, E. Ferro, F. Potortì, A. Bellini and F. Pirri
“Practical experiences in interconnecting LANs via satellite”, ACG SIGCOMM Computer Communication Review, Vol. 25, No. 5, pp. 56-68, October 1995.

[18] M.L. Molle
“A new binary logaritmic arbitration method for Ethernet”, Computer Systems Research Institute, University of Toronto, Technical Report CSRI-298, available at ftp://ftp.cs.toronto.edu/pub/reports/csri/298.

[19] N. Celandroni, E. Ferro, F. Potortì
“Experimental results of a demand-assignment thin route TDMA system”, International Journal on Satellite Communications, Vol.14, N.2, pp.113-126, March-April 1996.

[20] N. Celandroni, E. Ferro, F. Potortì
“MTG. Multiapplication Traffic Generator. Presentation and Use”, CNUCE Report C95-29/Rel. 4.0, September 1995.