MTG

Multi-application Traffic Generator

PRESENTATION AND USE

Rel. 4.0

Nedo Celandroni
Erina Ferro
Francesco Potorti'

CNUCE / C.N.R.
Via S.Maria 36 - 56126 Pisa (Italy)
Tel. : +39-50-593111
Telex: 500371 CNUCE
Fax : +39-50-589354

Email: n.celandroni@cnuce.cnr.it
Tel.: 050-593207 / 593203
Email: e.ferro@cnuce.cnr.it
Tel.: 050-593312 / 593203
Email: pot@cnuce.cnr.it
Tel.: 050-593203

CNUCE REPORT C95-29 / Rel. 4.0

September 1995

Table of contents

INTRODUCTION 1

CHAPTER 1. THE FODA/IBEA SYSTEM 2
1.1 Main features of the FODA/IBEA system 5

CHAPTER 2. THE MTG SYSTEM 6
2.1 MTG general structure 8
2.1.1 The MTG controller section 9
2.1.2 The MTG driver section 11
2.1.3 The driver interface 13
2.2 The memory structure for data exchange 16
2.3 The contiguous area allocation 17
2.4 The algorithm for data exchange 18
2.5 The random numbers generator 19

CHAPTER 3. THE MTG SOFTWARE ORGANISATION 21
3.1 INI.C (Initialisation. MTG_INI module) 22
3.2 MMD.C (Man-Machine Dialogue. MTG_MMD module) 24
3.3 SCH.C (Scheduler. MTG_SCH module) 25
3.4 RCV.C (Reception. MTG_RCV module) 26
3.5 LIST.C (Data base management. MTG_LST module) 26
3.6 MESS.C (Error Messages) 26
3.7 The MTG transmission 27
3.8 Data structures 28

CHAPTER 4. OPERATOR'S GUIDE. 33
4.1 The input file 33
4.1.1 The input file parameters 34
4.2 How to start MTG 41
4.3 The commands 42
4.3.1 R 42
4.3.2 L 43
4.3.2.1 D 43
4.3.2.2 T 45
4.3.3 E 46
4.3.4 Q 47
4.3.5 P 47
4.3.6 H 50

CHAPTER 5. THE PERFORMANCE OF MTG 51

CHAPTER 6. MEASURES WHICH CAN BE PERFORMED 55

REFERENCES 59

INTRODUCTION

The Multi-application Traffic Generator (MTG) system is aimed at testing and evaluating the performance of a Satellite Network Access System (SNAS) under test. The behavioural aspects of different LANs and MANs can be observed and explored with the MTG system.

The SNAS considered by the authors is based on the FODA/IBEA-TDMA satellite access scheme, described in detail in [1], [2] and [3].

This report is divided in 6 chapters.

Chapter 1 describes some features and gives some information regarding the FODA/IBEA system.

Chapter 2 gives the basic characteristics of the MTG system, including detailed information about organisation and purposes.

Chapter 3 describes the MTG organisation.

Chapter 4 contains a guide to the use of the system.

Chapter 5 presents the performance of the MTG system.

Chapter 6 shortly describes the measures which can be performed by means of MTG.

CHAPTER 1. THE FODA/IBEA SYSTEM

A rough description of the FODA/IBEA functioning is here given and some key terms and basic functional blocks are explained.

FODA/IBEA software - the software for receiving/transmitting data from/to the satellite, running on the RX-TDMA and the TX-TDMA controller units, respectively.

TDMA terminal - consists of:

a) a burst-mode modem able to handle different bit rates (from 1 Mbit/s up to 8 Mbit/s), variable on a sub-burst basis;

b) a codec which supports variable coding rates: 1/2, 2/3, 4/5 and uncoded PSK. Punctured codes, derived from the 1/2 convolutional encoder, are adopted. The decoder operates asynchronously at 8 Mbit/s information bit rate with 3 bit sign/magnitude soft decisions.
Guarantied error rates better than 10^{-4}, 10^{-6} and 10^{-8} for different services are offered respectively, by adapting the code rate to the signal/noise conditions.

c) a TDMA controller, implemented using two VMEbus based racks, one for the transmit controller and one for the receive controller. High speed communication between the two controllers is possible using a SCSI bus. This CPU communicates over the VMEbus with a number of other cards. Each unit is based on a MOTOROLA MVME147S-1 processor board and a transition module MVME712M. The processor board uses a 68030 processor card containing 4Mbyte of RAM. It runs at 25MHz and occupies one 4E wide VMEbus slot.

The *transmit serial interface board* is a wire wrapped VMEbus compatible double Eurocard board (identity Y-35-9042). It acts as a buffer between the VMEbus and the three terrestrial interface ports of the transmit controller. The three ports are: a single high speed RS 449/ RS 422 port (high speed serial interface, at 384 Kbit/s), and two CCITT G.703 ports (low speed serial interfaces, at 64Kbit/s).

The *transmit modem interface* is controlled by the transmit control processor via a number of memory mapped ports which allow both reading and writing of control data. The circuit also provides the interface to the modem for data transmission and fault monitoring. The interfacing function is performed by two boards: the transmit modem interface and the line interface.
The transmit modem interface board is a wire wrapped VMEbus compatible double Eurocard board (identity Y-35-9041) 8W wide. It contains all the channel coding, framing and symbol rate selection functions. It processes data presented to it from the 32-bit VME data bus and also provides status information for circuit and modem monitoring purposes.

The *line interface board* is a double Eurocard size printed circuit board (identity Y-35-9037). This contains the line drivers/receivers for interfacing to the modem and also provides the 16.384 MHz reference clock, derived from a 32.768 MHz oven controlled crystal oscillator situated on the line interface board.

The *receive serial interface board* is a wire wrapped VMEbus compatible double Eurocard board (identity Y-35-9053). It acts as interface between the VMEbus and the three terrestrial interface ports of the receive controller. The three ports are: a single high speed RS 449/ RS 422 port (high speed serial interface, at 384 Kbit/s), and two CCITT G.703 ports (low speed serial interfaces, at 64Kbit/s).
Data is transferred between the receive control processor and the receive serial interface board via the VME Data Transfer Bus (DTB). A data transfer process is initiated by the receive serial interface board when any of the three dual ported memories becomes half full.

The receive modem interface and the channel decoder boards are two wire wrapped VMEbus compatible double Eurocard board (identities Y-35-9051 and Y-35-9052 respectively).
The *receive modem interface* is a processor controlled via a number of memory mapped ports which allow both reading and writing of status, control and data information between the processor and the hardware. The circuit also provides the interface to the modem for data reception and fault monitoring. The interfacing function is performed by three boards: the receive modem interface, the channel decoder and a line interface board.

The receive modem interface and the channel decoder boards are wire wrapped VMEbus compatible double Eurocards boards, each 8E wide. These contain all the channel decoding, framing and symbol rate identification functions. The receive modem interface processes data presented to it from the 32-bit VME data bus and also provides status information for circuit monitoring purposes.

The line interface board is a double Eurocard size printed circuit (identity Y-35-9037). This contains the line drivers/receivers for interfacing to the modem and also provides the 16.384 MHz reference clock.

FODA/IBEA system **-** both the FODA/IBEA software and the TDMA terminal (as shown in Fig. 1).

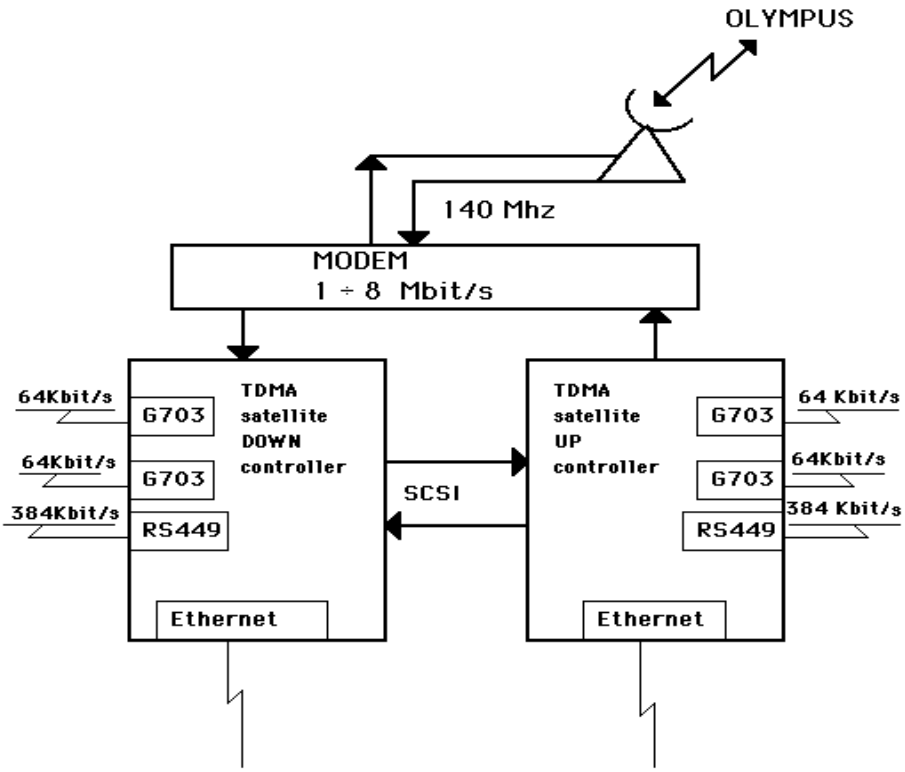

Fig 1. The FODA/IBEA system

1.1 Main features of the FODA/IBEA system

- The system can work both as a transit network and as a private network with directly attached users.

- The channel scheduling and the control are centralised.

- Two different basic types of traffic are simultaneously supported:

  - ***datagram*** *traffic.* It is generated by asynchronous applications, like bulk data transfers, interactive computer accesses, mailings and data base manipulation. The required channel characteristics are essentially: high throughput and low bit error rate, while the delay is not very critical. Corrupted packets are not accepted.

  - ***stream*** *traffic*. It is generated by synchronous applications, like the packetized voice and video in full or slow motion. The required channel characteristics are: fixed throughput, very low delay and regular inter-arrival time of the packets. Corrupted packets are accepted.

- The transmission time is divided into *time frames,* each one 20 ms long.

- Opening/closing of virtual circuits (for stream data) and retransmissions of corrupted packets (for datagram data) are demanded to higher level protocols.

- Fragmentation/reassembling techniques are implemented. Channel saturation control techniques are also foreseen.

- As far as the channel access strategy, the channel assignment and the burst structure are concerned, the reader is kindly requested to refer to [1].

- The FODA/IBEA system communicates with the rest of the world by means of an ad-hoc developed protocol, named GAFO. It is described in [5].

CHAPTER 2. THE MTG SYSTEM

MTG produces a traffic that stems from the simultaneous activity of a number of schematic user-defined independent sources called *Traffic Generators* (TGs). Thanks to the combination of the traffic generated by all the TGs it is possible to simulate arbitrary complex traffic patterns, using a comparatively small set of parameters (detailed in chapter 4). Volumes of traffic which rapidly increase or bursty sources are easy to configure, as they are devices which transmit at constant throughput. A small set of characteristics defines the behaviour of each TG. All the TGs are fully described in a common *input descriptor file* (IDF) by the listing of their characteristics. They act concurrently during the simulation: the traffic generated by MTG is the sum of the traffic individually generated by each TG. The number of simultaneously active TGs is only limited by the performance of the machine where MTG runs.
The MTG system (Fig. 2) has 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.

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

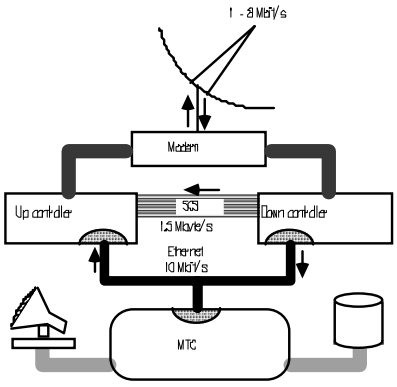

Fig 2. MTG system configuration

The UNIX system provides a comfortable environment for the programmer, but poses fundamental obstacles because of its non real-time behaviour. Real-time behaviour is vital to MTG because of the precision required by the packet dispatching scheduling times and the time stamps contained in the MTG header. Since System V currently provides non standard means to overcome this problem, a brute force approach has been taken. The devices related to MTG (i.e. the LAN manager and the timer) have been forced to generate non-maskable interrupts, while the operating system is free to carry out its work. This is necessary because MTG requests the operating system services when writing onto disk. This technique allows full use of processor power. As a consequence there is practically no disk writing overhead, if the UNIX I/O buffering parameters are suitably chosen.

Real-time behaviour is vital to MTG because of the precision required from the packet dispatching scheduling times and the timestamps contained in the MTG header. These timestamps allow an accurate estimate of the parameters under measure.
System calls are limited to cases such as the start or stop of a TG, the writing of a big disk buffer containing the headers of the received packets, or the check for operator interaction. The LAN packet sending/receiving operations do not relate with the operating system. Data to be sent is contained in only one buffer for each TG. This buffer is filled (during the pre-run phase) up to the maximum length and a fixed portion (according to the current length) is sent at each dispatch time.

The C language has been chosen as the most suitable and convenient implementation language.

2.1 MTG general structure

To provide the required amount of traffic, the MTG system has been organised in two separated, but closely interrelated, sections (Fig. 3):

a) *the MTG controller*

b) *the MTG driver.*

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

The controller is dependent on the communication protocol only, while the driver is tied to the LAN and it is independent of the protocol. The present implementation of MTG is based on Ethernet. The block structure organisation of MTG offers some flexibility. In fact, it will be possible to test the FODA/IBEA system over different LANs or MANs (Token Ring, FDDI, etc.) by changing the driver section only. On the other hand, in principle, any other type of network can be tested if the controller section is changed in order to support a different communication protocol.

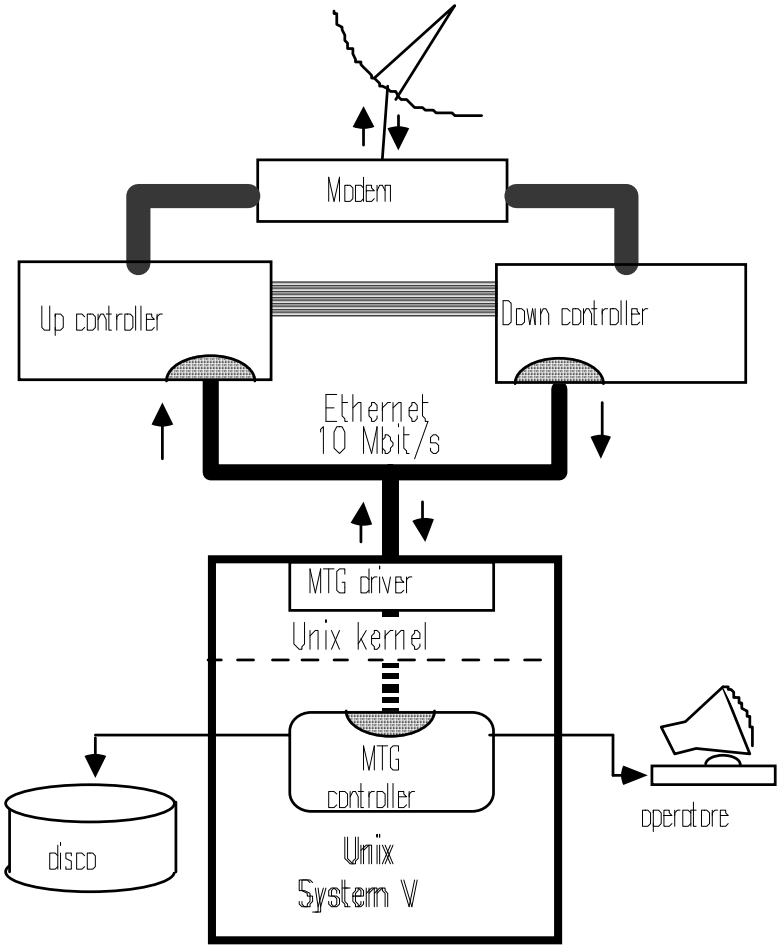

Fig. 3. MTG general structure

2.1.1 The MTG controller section

The controller section is related both to the interface with the user (being MTG an interactive program) and to the interface with the operating system. It is responsible for the start-up procedures (involving the reading of the IDF), the calculation of the *TG tables* (TGT), related to the transmission times of the packets, the real-time display of the events and the writing of the packet headers on disk.

The controller section starts by reading the IDF which contains the descriptions of all the TGs that will be active during the MTG run. For each TG, it allocates a data

structure and builds a *Traffic Generator table Descriptor* (TGD) which points to the TGT containing the dispatching times and the lengths of the packets which must be generated by the associated TG. The time intervals between two consecutive packet dispatches are calculated according to the relevant TG input parameters (contained in the IDF) and stored in the TGT entries. Once the TGTs are built by the controller, a start and a stop timer are set for each TG. When a *TG-start* timer expires, the relevant TGD is handed to the driver section of the MTG which, in turn, takes care of sending the packets on the LAN at the scheduling times specified in the pointed TGT. When the *TG-stop* timer expires, the controller kills the TG by commanding the driver to cancel the relevant TGD.

The controller part also handles the GAFO protocol. It builds the GAFO control messages and sends them to the driver, while the received GAFO control messages are organised in a specialised *control-and-error queue* (different from the *data packet queue*) to achieve fast response times.
The controller continuously monitors the received data packet queue, and writes on disk the MTG headers of the received packets (described in the following). It also monitors the keyboard for operator actions: in fact, it is possible to have a real-time overview of the active TGs and their parameters, and to start or stop any TG.

Each TGT is described by:

- the Ethernet header (as this implementation is based on the Ethernet LAN)
- the GAFO header
- the MTG header
- the data to be sent
- the TGT and the timetable addresses
- the size of the timetable
- the timetable entries (for the TG_CREATE system call).

Each entry in the timetable contains:

- the length of data to be sent with the current packet
- the number of ticks which must elapse before sending the next packet.

The timetable is terminated with a null entry.

The TGTs are prepared by the controller and delivered to the MTG driver section by issuing appropriate commands which provide all the needed services on the transmit side (TG_XMT). The interface between the MTG driver and the MTG controller sections is provided by the Unix *ioctl* system call.

2.1.2 The MTG driver section

The code of the driver section has been included as part of the UNIX Kernel and runs in *supervisor mode*. The MTG driver section implements the low-level interface to the satellite controller, managing both the transmit and the receive parts. In the current implementation it is based on the Ethernet LAN.

Two sets of interrupt routines are implemented:

- *The tick timer and the LANCE Tx-side handler.*
  They initiate the messages sending according to the indication contained in the TGT linked list. This list is scanned by the MTG driver which takes care of the physical transmission of the data packets on the LANCE.

- *The LANCE Rx-side handler*.
  It fills the reception buffers.

The driver works at the highest priority, in order to guarantee accurate timings. It 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 it receives packets in a memory area mapped into its virtual space. The task of the driver is to read the TGTs created by the controller section 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 (one *microtick*) resolution.
Fig. 4 depicts the organisation of the TGTs. The driver gets the TGDs from the controller and acts on them according to the commands received by the controller.

The driver scans the list once per tick, looking for packets to dispatch. When a TGD relevant to an active TG is found, the *tick counter* is decremented by one. If the result is a non-null value, the next TGD is examined. At the end of the scan, all the tick counters have been decremented. When a tick counter reaches zero, the first *length* bytes of the packet relevant to that TGT are sent on the LAN. The pointer to the current timetable entry is then advanced (the timetable is circular) and the Δt field is copied into the tick counter field. The scan is then suspended until the occurrence of the EOT(1) interrupt. It should be clear from this description that packets to be sent are handled one at a time. This behaviour imposes an overhead (CPU dependent) of about 60 μs on each packet sent — which denies the possibility of back-to-back packet transmission — but allows MTG to set accurate transmission timestamps.

It must be noticed that, in the current configuration, MTG uses the same Ethernet cable for both the outgoing and the incoming traffic. It is so impossible to reach the maximum throughput of a FODA/IBEA station while receiving back the traffic, because a FODA/IBEA station's throughput exceeds 7Mb/s and the Ethernet cable has a 10Mb/s capacity. A future version of MTG is foreseen that will use two separated Ethernet stubs. Such a configuration will allow a thorough test of the FODA/IBEA system at maximum speed, at the expense of increased MTG cost and complexity.

From the driver's point of view, each TGT is described by:

- the TGT starting address (in the controller space)
- the starting address of the timetable (for the TG_CREATE system call)
- the timetable size in a *mtg_ctl* area.

The real time functions of the system, such as the packet sending at precise time instants, are performed by the asynchronous part of the driver, which is interrupt driven.

(1) End Of Transmission on the LAN

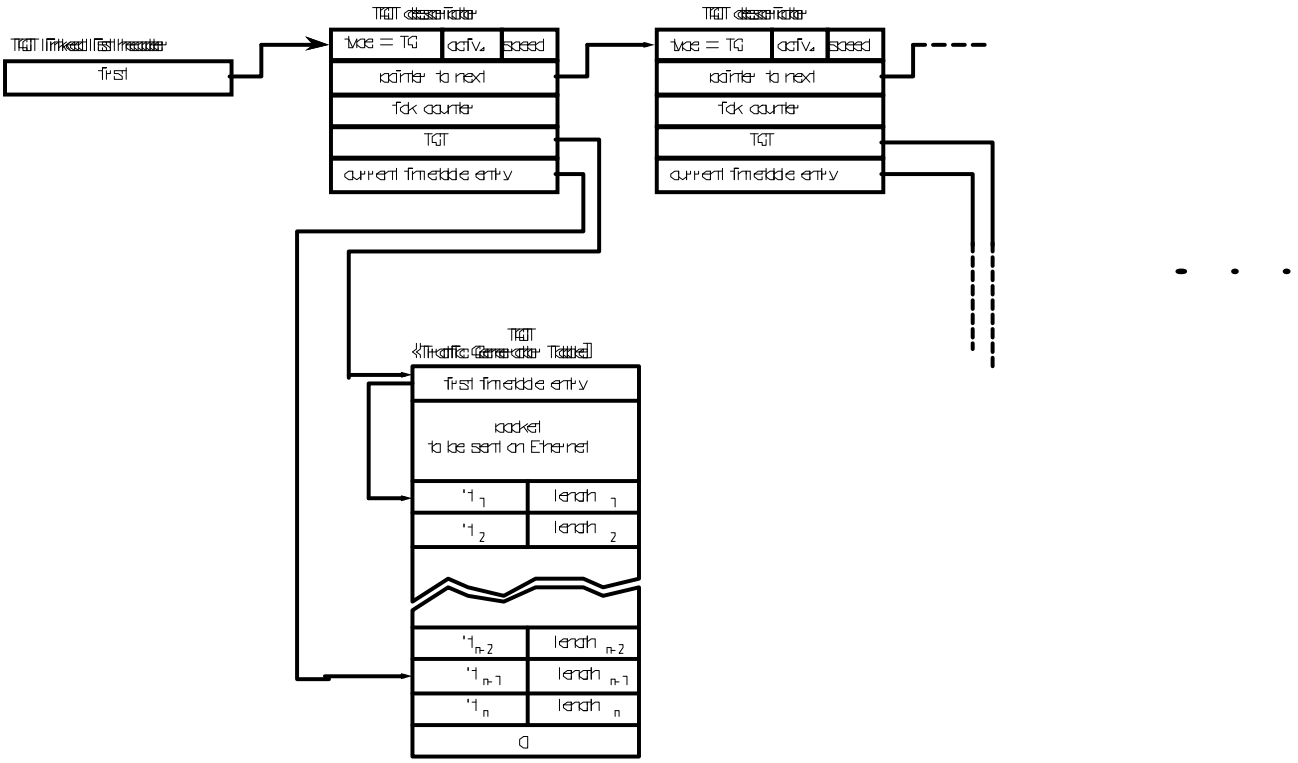

Fig. 4. The TGT linked list organisation

2.1.3 The driver interface

The communication between the control section and the driver section is implemented via standard *ioctl* system calls. The *mtg_ctl union* is filled by the control with the arguments needed by the involved command.
The function *ioctl* performs a wide variety of control functions on devices. The meaning of the argument is driver-specific.
The syntax for giving a command to the driver is:

```
ioctl (special, cmd, arg)
```

where:

special is an open file descriptor referring to a device (mtg_dev);

cmd is the command which selects the function to be performed;

arg controller address of the data required for the particular command.

If successful, the function *ioctl* returns a positive integer value (1). A negative value (-1) indicates an error.

Six different types of commands are provided:

ioctl (special, TG_CREAT, &TGT)
ioctl (special, TG_KILL, &TGT)
ioctl (special, TG_CTL, &CTL)
ioctl (special, TG_OFF, &TGT)
ioctl (special, TG_ON, &TGT)
ioctl (special, TG_RATE, &TGT).

The meaning of the commands is explained in the following.

TG_CREAT -> starts a TGT.
The traffic generator relative to each TGT is initiated by issuing the TG_CREAT command. The controller fills the TGT data packet and the TGT timetable, according to the parameters given by the operator. The address of the TGT, the address of the timetable and its size are passed to the driver via the *mtg_ctl* area.

TG_KILL -> cancels a TGT.
The controller gives to the driver the address of the TGT to be cancelled. The driver gives back to the controller the sequence number of the last sent packet. The controller then waits until the last data packet is received before erasing its own structures relative to the TGT.

TG_CTL -> sends a control message.
The controller supplies to the driver the address of the control packet to be sent, then erases the control packet.

TG_OFF -> suspends a TGT.
Upon reception of the control message STOP_DAT_PKT (congestion detected) from the FODA/IBEA system, the controller issues the TG_OFF command, giving to the driver the address of the interested TGT (datagram data type).

TG_ON -> resumes a TGT.
Upon reception of the control message RESUME_DAT (congestion overcome) from the FODA/IBEA system, the controller issues the TG_ON command, giving to the driver the address of the interested TGT (datagram data type).

TG_RATE -> changes the throughput of a TGT.
Upon reception of the control message STR_CH_MODIF (reduced or maximum throughput) from the FODA/IBEA system, the controller issues the TG_RATE command, giving to the driver the address of the interested TGT (stream data type) and two factors by which the time-table entry members (Δt's and packet lengths) must be multiplied.

When the *ioctl* fails, an error number is put by the system in the global variable *errno*. The meaning of the errors follows.

EINVAL occurs when:

• the controller-driver protocol is not respected;

• the receive buffer had been already allocated and the controller tries to allocate it again;

• no buffer is currently allocated and the controller issues the TG_GETRBUF command;

• the tick length is out of range.

ENOSPC the driver constant TGDPOOL_SZ is too small.

ENOMEM the driver constant TTPOOL_SZ is too small.

In the following table, the error codes passed by the driver to the controller receive areas are listed.

TG_SLOW	(code = 0x01)	TG too slow.
TE_LCOL	(code = 0x02)	Late collision on transmission.
TE_LCAR	(code = 0x03)	Loss of carrier on transmission.
TE_RTRY	(code = 0x04)	Retry error on transmission.
TE_SEMA	(code = 0x05)	Too much backlog.

2.2 The memory structure for data exchange

A contiguous buffer is allocated by the driver and is divided into two areas managed as circular queues. The queues are:

- *The control-error queue.*

The error messages are enqueued in the same queue used for the incoming FODA/IBEA control messages. The driver enqueues error messages as control messages, e. g. by using the same data structure with a different data type.

- *The data packet queue.*

The data packets are enqueued in a separate queue for incoming FODA/IBEA data messages. The driver is responsible for proper message identification and enqueueing.

2.3 The contiguous area allocation

The buffer manager handles the contiguous areas simply by using the TG_GETRBUF, TG_ASKRBUF, TG_FREERBUF commands of the driver.
Each queue consists of a fixed number of buffers. CMP_SIZE is the size of the control-error queue, whose buffers have fixed length CTL_LENGTH. The data packet queue size is DPP_SIZE, and the related buffers have fixed length TG_MAXDPL.

The contiguous area (physically contiguous space) for this set of buffers is allocated as a whole at the start-up time by using the buffer-related commands of the driver:

- TG_GETRBUF allocates contiguous space;
- TG_ASK_RBUF returns the physical address and the size of the allocated space;
- TG_FREERBUF releases the allocated space.

MTG uses the previous commands only in the routines to get a buffer a to release a buffer (*getrbuf()*)and *freerbuf()* routines, respectively). The sequence for allocating the needed buffer space is:

- the contiguous buffer is allocated and mapped in the controller space by using the *getrbuf()* routine;
- the control-error (CMP) array is allocated;
- the data (DPP) array is allocated;
- the controller address of the contiguous buffer and the controller addresses of the CMP and DPP arrays are passed to the driver by issuing the TG_INIT command.

2.4 The algorithm for data exchange

The control-error and the data queues are implemented as circular linked lists of buffers. During the initialisation, all the buffers are marked for *write* (i.e. they are free buffers).

When the driver receives a message, it takes the buffer from the beginning of the queue and sets its address in the control-error or in the data array, depending on

the packet type. The buffer then results marked as *readable* (i.e. busy buffer). The driver then takes the *next* buffer as *current* buffer (Fig 5).

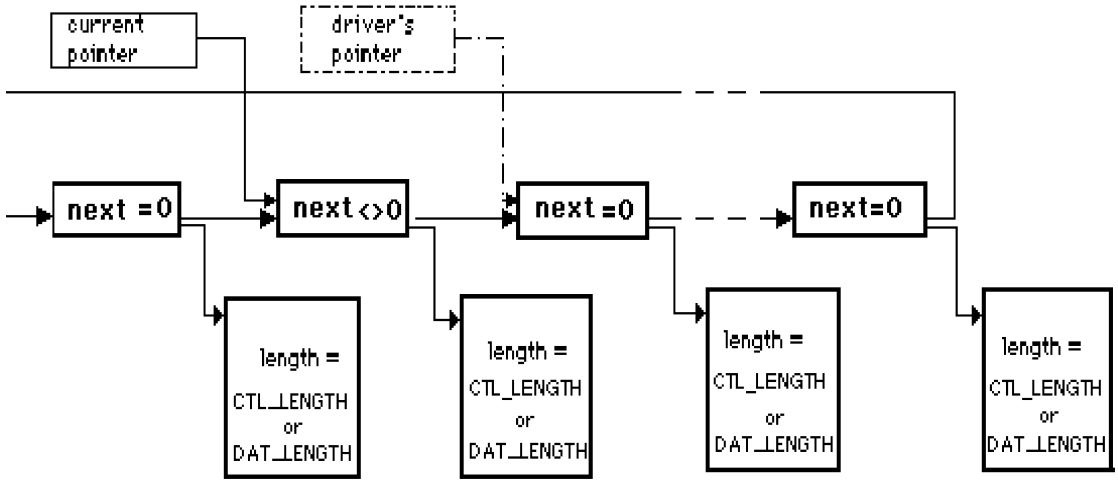

Fig. 5. The receive queue area

The MTG scans the readable addresses (non-zero values) in the control-error and in the data queues, starting from the *current* pointer. If the current pointer has a non-zero value, the buffer is read. After the processing the address is cleared, so the buffers results writable again.

Summarising:

- the driver marks the buffers as busy (non-zero value of the pointer);
- the controller marks the buffers as free (zero value of the pointer).

This mechanism is used for collecting all the incoming FODA/IBEA packets. The control packets, the error packets and the headers of the data packets are stored in an archive file (*mtgout*). At the end of the session the archive file contains the complete information about the session run. The collected data is used during the post-run analysis.

2.5 The random numbers 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. Particular care is required when choosing the algorithm for a particular simulation application. The issues and the constraints involved in the random number generation internal to MTG are here discussed.
All the random numbers used within MTG are generated off-line. This means that a table for each TG is created at the MTG start-up, prior to the simulation run, containing entries filled with the starting time and length of each packet. The parameters for the tables generation are taken from the IDF that lists the characteristics of the TGs.
The choice of table-based random number generation has been made because the MTG is a software tool, and it uses the entire processing power of the underlying hardware.
Different approaches can be found in the literature. Ramshaw and Amer [11] 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 a 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 one 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 makes thus impossible the on-line generation of the random numbers, due to the extra charge that would be imposed on the CPU.

The adopted table-based solution has well known advantages and disadvantages. Indeed, it is possible to choose a good generator without worrying about computing times, and it is possible to exactly transform the uniform distributed random number sequence into an arbitrary distribution (for example the exponential distribution). The major advantage is that at run time it is only required to read a number in a table, instead of requiring an on-line generation. On the other hand, the period of the generated sequence is equal to the length of the table, this being a major disadvantage.

The adopted uniform random number generator is a Lehmer's generator with modulus 2^{31}-1 and multiplier 950706376 [8]. The formula used to transform a uniform-distributed variable into an, let's say, exponential variable is a simple inversion. This is possible without time penalties because the computations are carried out before the start of the real simulation phase.

The choice of the length to be adopted for the tables is not a secondary problem, since the goodness of the generator can be impaired significantly by the table-based approach. Anyway, no time constant in the FODA/IBEA system exceeds a double round-trip delay, which is less than two seconds. Since every table entry requires 8 bytes and the maximum TG throughput is one packet per millisecond, the overall memory encumbrance of the tables is less than 8KB per TG per second. If a 1MB space is dedicated to the tables, 8 TGs at full throughput are able to generate a traffic whose pattern repeats every 16s, which is over ten times the maximum expected time constant internal to the FODA /IBEA system.

The depicted scenario is a worst-case in many respects. First, not all the TGs may be required to have maximum throughput, resulting in a longer repetition time. Second, more TGs could be employed at lower, differentiated throughput, so that the overall traffic pattern repetition time would be the least common multiple of the single TG's repetition times. Third, not all the TGs could be required to generate random traffic: a steady traffic just requires one entry in a table. As a last resort, one could even increase the memory space dedicated to the tables. The UNIX system MTG is currently running on has an 8MB real memory space. Less than 3MB are required by the operating system, the disk buffers and the executable MTG image in memory. Depending on the MTG working mode, 1 to 4MB are required for the receive buffer, the rest (1 to 4MB) being available for the tables.

CHAPTER 3. THE MTG SOFTWARE ORGANISATION

Fig. 6 shows the internal organisation of the MTG system.

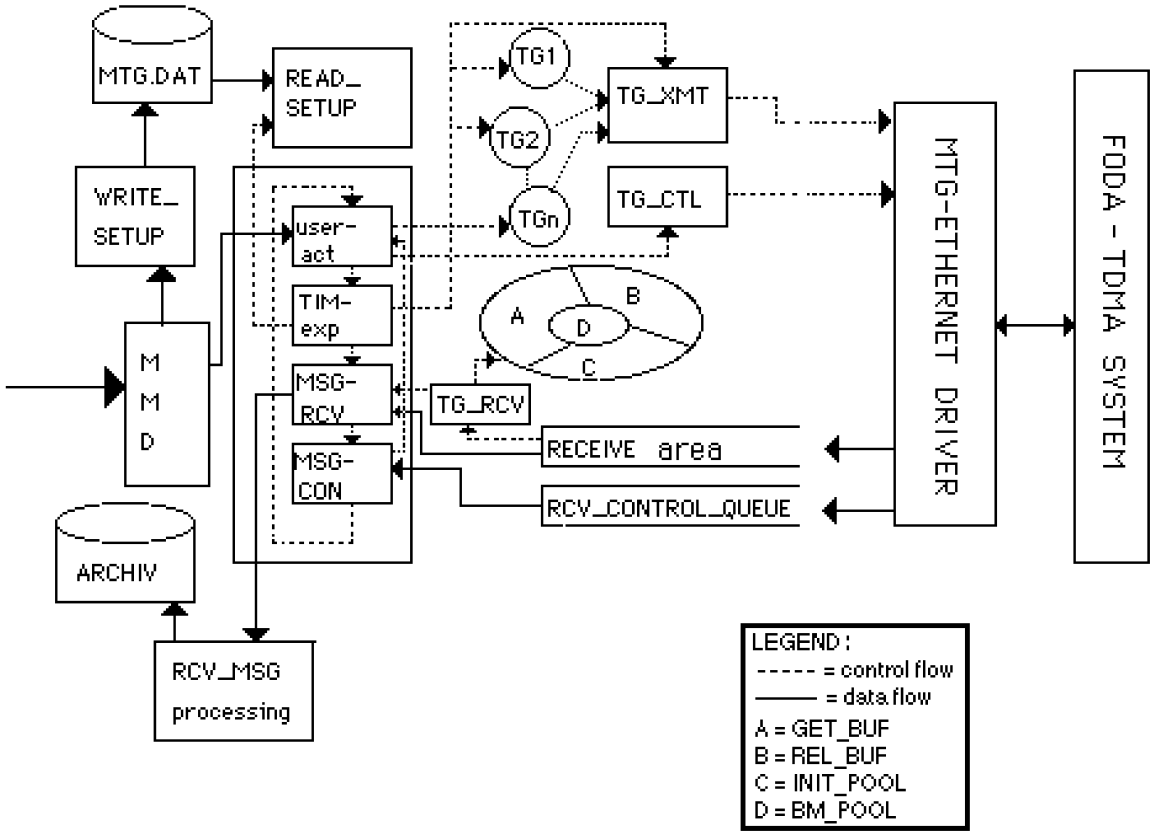

Fig. 6. MTG organisation

The MTG software consists of the following set of files:

mmd.c	man-machine dialogue
ini.c	initialisation routines
sch.c	scheduler routines
rcv.c	message processing routines
list.c	list manipulation routines
mess.c	available messages.

3.1 INI.C (Initialisation. MTG_INI module)

This program provides the basic information needed to define the traffic generation session and to run it. Each TG is defined in an IDF prepared by the user. The IDF describes the general characteristics of the MTG system and gives the detailed description of the traffic generators connected to each station.

The IDF must have the format shown in Fig. 7.

```
#Date
#general information (1 line). Valid for all the TGs.
#
msem n   tick n    rbuf n   ttsz n   dkbf n   mask n   refr n
eth1 0xn1 0xn2 0xn3 0xn4 0xn5 0xn6     eth2 0xn1 0xn2 0xn3 0xn4 0xn5 0xn6
eth3 0xn1 0xn2 0xn3 0xn4 0xn5 0xn6     eth4 0xn1 0xn2 0xn3 0xn4 0xn5 0xn6
#
#description of the traffic generators (1 TG per line). Example of one TG.
#
tgid n          tabl  n data n dist t npkts t n1 n2   thro n1 n2
patt  t n       time t n1 n2   qlen n jitt  t n cos n
opt  n1 n2      addr  t n       seed n          inta n1 n2      upcn n
nowb n          req c n1 n2    fnam   c filename
#
```

Fig. 7. The input descriptor file (IDF)

The meaning of the used fields is explained in 4.1.1.

When started, this program checks for validity the IDF parameters.
For each defined TG, this module creates two linked lists:

- the *Traffic Generator Table (TGT)*
- the *Traffic Generator table Descriptor (TGD),* according to the input file. It describes the general characteristics of the relevant traffic generator.

Each traffic generator can produce packets with one of the following inter-packet time distributions:

Fixed The intervals of time (Δt) between packets are constant.

$$\Delta t = 8 \cdot \frac{\text{pkts\_size}}{\text{thro\_req}} \quad \text{[ms]}$$

where:

pkts_size	is the packet size [bytes]
thro_req	is the requested throughput [Kbit/s].

Random The length of the packets (**L**) is a uniform random variable in a range between a minimum (pkts_min) and a maximum (pkts_max) value.

L = pkts_min+ (pkts_max - pkts_min) RAND

where RAND is a uniformly distributed random variable in the [0,1] interval. Since the produced numbers are not truly random at all, as each one depends in a definite way on its predecessor, it should be spoken of pseudo-random numbers. If the seed begins with same value each time the program is run, then the whole sequence of pseudo-random numbers is exactly the same. For that it is preferable to begin by setting the seed to a random value. So the system time is used as initial seed number and then the got random number is used as seed for the next random number production.

The interval of time Δt between packets is given by:

$$\Delta t = 8 \cdot \frac{\text{pkts\_max} + \text{pkts\_min}}{2 \cdot \text{thro\_req}} \quad \text{[ms]}$$

Poisson The intervals of time Δt between packets follows a negative exponential distribution.

The probability density function of the Poisson random variable X, with mean E[X] = t and variance VAR[X] = t^2 (t > 0), is:

$$f(x) = \frac{1}{t} \; e^{-\frac{x}{t}} \qquad \text{for } x \geq 0 \qquad (1)$$

and

$$f(x) = 0 \qquad \text{elsewhere.}$$

The cumulative distribution function is:

$$F(X) = \int_0^x \frac{1}{t} \; e^{-\frac{x}{t}} \; dx \; = \; 1 - \; e^{-\frac{x}{t}} \qquad (2)$$

Applying the inverse transformation to the (2) it is possible to generate exponential random variables [6]. Letting :

$$r = F(X) = 1 - e^{-\frac{x}{t}} \qquad \text{for } 0 \leq r < 1,$$

$$F^{-1}(r) = X = - \, t \log (1-r)$$

If the number r is uniformly distributed in the [0,1] interval, the argument (1 - r) has the same distribution.

Finally Δt can be computed by:

$$\Delta t \; = - \, Dt\_in \; \log RAND \qquad (3)$$

where:

$$\Delta t \_ \textbf{in} = 8 \; * \; \frac{pkts\_size}{thro\_req} \qquad [ms].$$

Voice The *talk* and the *silent* periods are sorted according to the statistical distribution relative to the English language contained in proper tables. Each entry in these tables is made using a computed random number as a displacement. During the *talk* periods, the interval of time Δt is computed exactly like in the fixed case.

External Distribution defined by user in a file containing Dt intervals.

3.2 MMD.C (Man-Machine Dialogue. MTG_MMD module)

The Man-Machine Dialogue (MMD) module provides the necessary interface with the operator.
The MTG is activated by an explicit operator request. The opening screen contains the basic information about the acceptable commands.

To this dialogue chapter 4 is devoted.

3.3 SCH.C (Scheduler. MTG_SCH module)

This module is the main scheduling loop. It :

- creates the archive file for the current MTG run
- handles the buffers
- serves (in an end-less loop) the:
  - terminal entry
  - message reception.

The scheduler handles all the above events. At the end of each routine the control is returned to the scheduler. The occurring of each event causes the MTG to switch into one of the following states:

INI_S	**0**	Initial state
WAIT_ALARM_START	**1**	Wait timer start
WAIT_ACK	**2**	Wait for ACK/REF
SEND_DATA	**3**	Send data without waiting timer start
WAIT_ALARM_STOP	**4**	Wait timer stop
CON_DET	**5**	Congestion detected
CON_OVER	**6**	Congestion overcome
CLOSED	**7**	TGT finished
WAIT_MACK	**8**	Wait for modification ACK
USR_GROUP	**9**	Build users group
WAIT_GACK	**10**	Build users reply group
USR_EXT	**11**	Build extended users group
USR_EACK	**12**	Build extended users group ACK

3.4 RCV.C (Reception. MTG_RCV module)

This module manipulates two separate buffer queues:

- the control-error queue
- the data packet queue.

The module scans the queues and reads the messages, calling the proper routines.

3.5 LIST.C (Data base management. MTG_LST module)

This module manipulates

- the TCB list (Time Control Block)

linked list in MTG. The TCBs are sorted in increasing order using the start and the stop times of the TGTs.

The arrays

- the TGT array (Traffic Generator Table)
- the TGD array (Traffic Generator table Descriptor)

are structured as two arrays (earlier structured as linked lists).

3.6 MESS.C (Error Messages)

During the MTG run, two types of errors may occur: fatal and non-fatal. The fatal errors cannot be handled by the MTG software and MTG terminates. When non-fatal errors occur, a warning message is printed on the terminal and the process goes on.
In both cases, a message describing the type of error or the cause of the forced finishing is printed.
Since multiple modules can generate same error message each message has an reserved position which indicates uniquely the type of the error.

3.7 The MTG transmission

The transmission side of the MTG is organised as an array of pointers to structures, which collects all the TGTs involved in the traffic generation.

Each structure is shown in Fig. 8 and contains:

- the LAN header (Ethernet, in the current implementation)
- the GAFO header
- the MTG header
- the pointer to the timetable relevant to the TG.

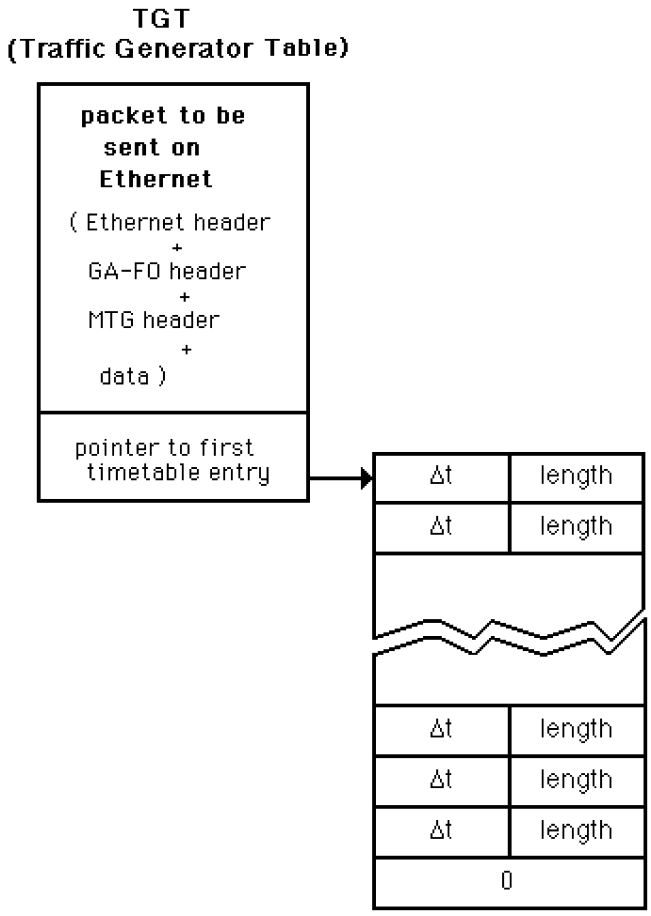

Fig. 8. TGT structure

The traffic generation relevant to each TGT is initiated by issuing the TG_CREAT command. The traffic generation is ended by issuing the TG_KILL command.

3.8 Data structures

Data structures are provided for different use:

a) data structures regarding the communications with the FODA/IBEA TDMA controller;

b) data structures regarding the MTG.

As far as the communications with the FODA/IBEA system (control messages and data) are concerned, they are regulated by the GAFO protocol, described in [5] where also the message's formats can be found.

As far as point b) is concerned, the MTG header is here described, which contains timing and other parameters relative to the MTG and to the FODA/IBEA systems. It is structured as shown in Fig 9, 10 and 11, according to the type of data to exchange.

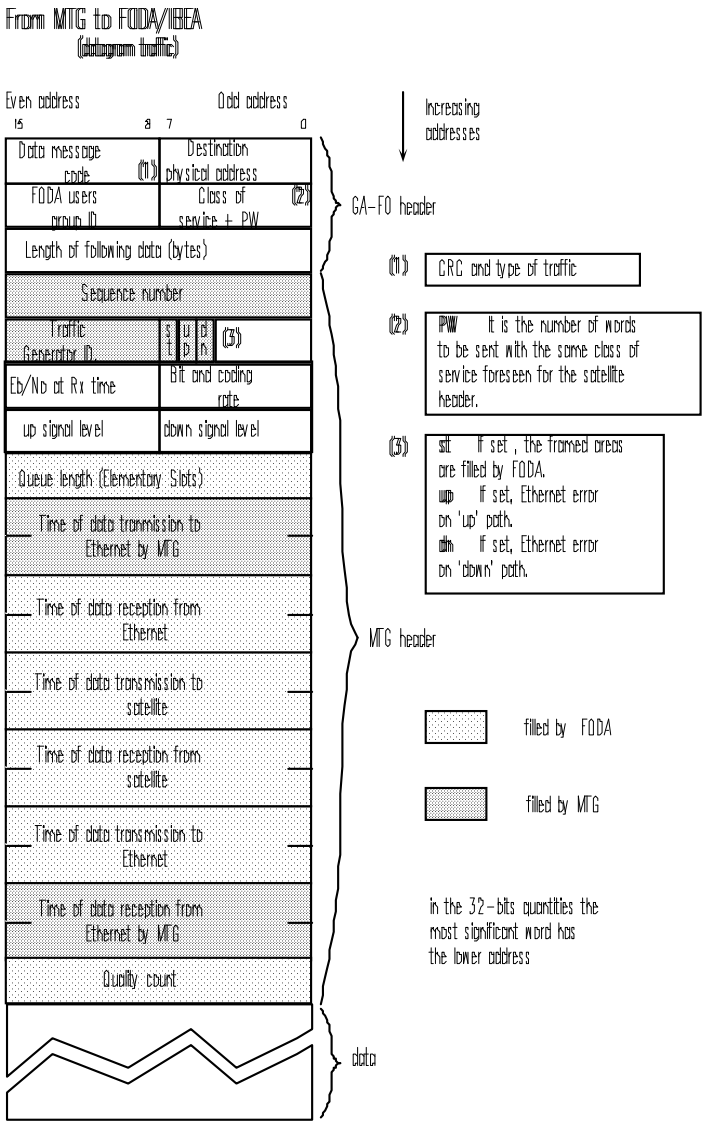

Fig. 9. Header for datagram data sent to FODA/IBEA from MTG

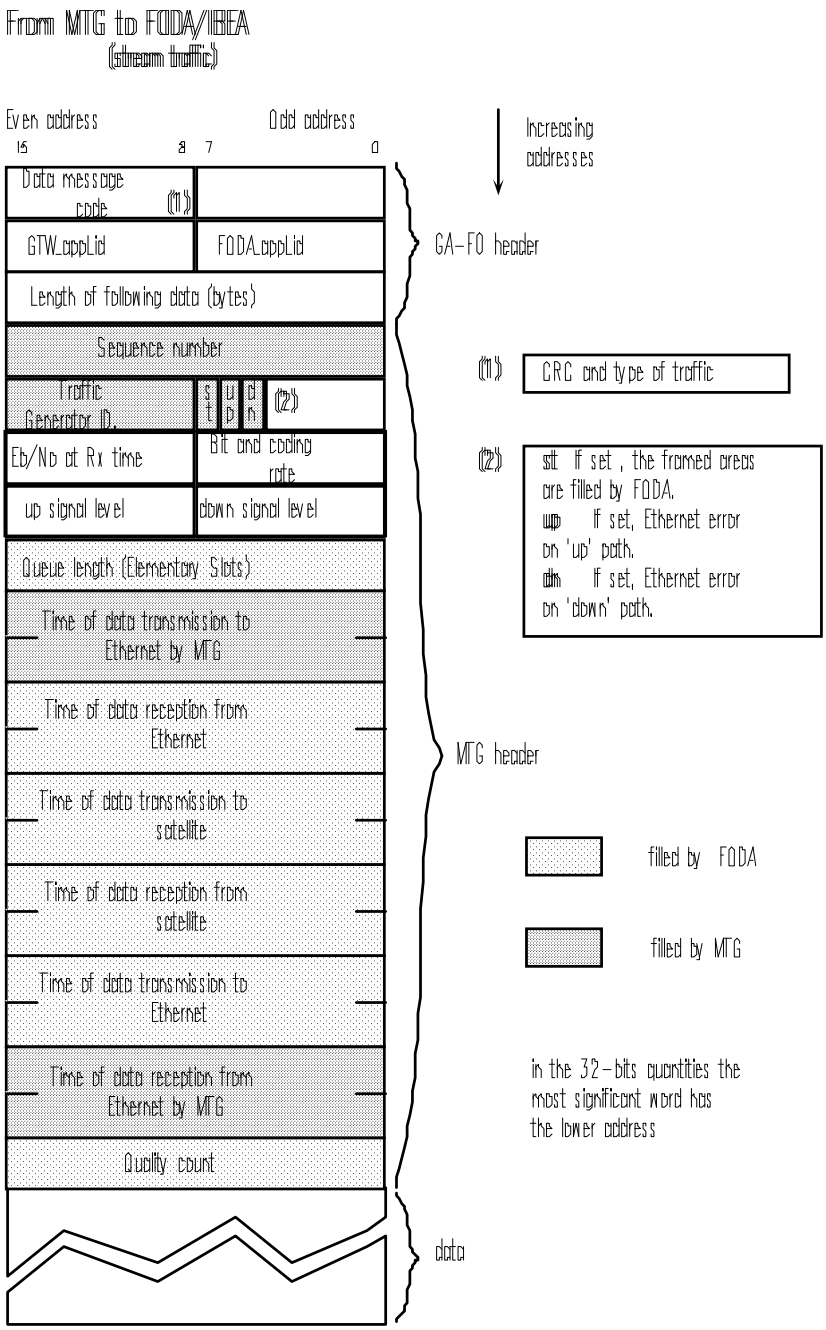

Fig. 10. Header for stream data sent to FODA/IBEA from MTG

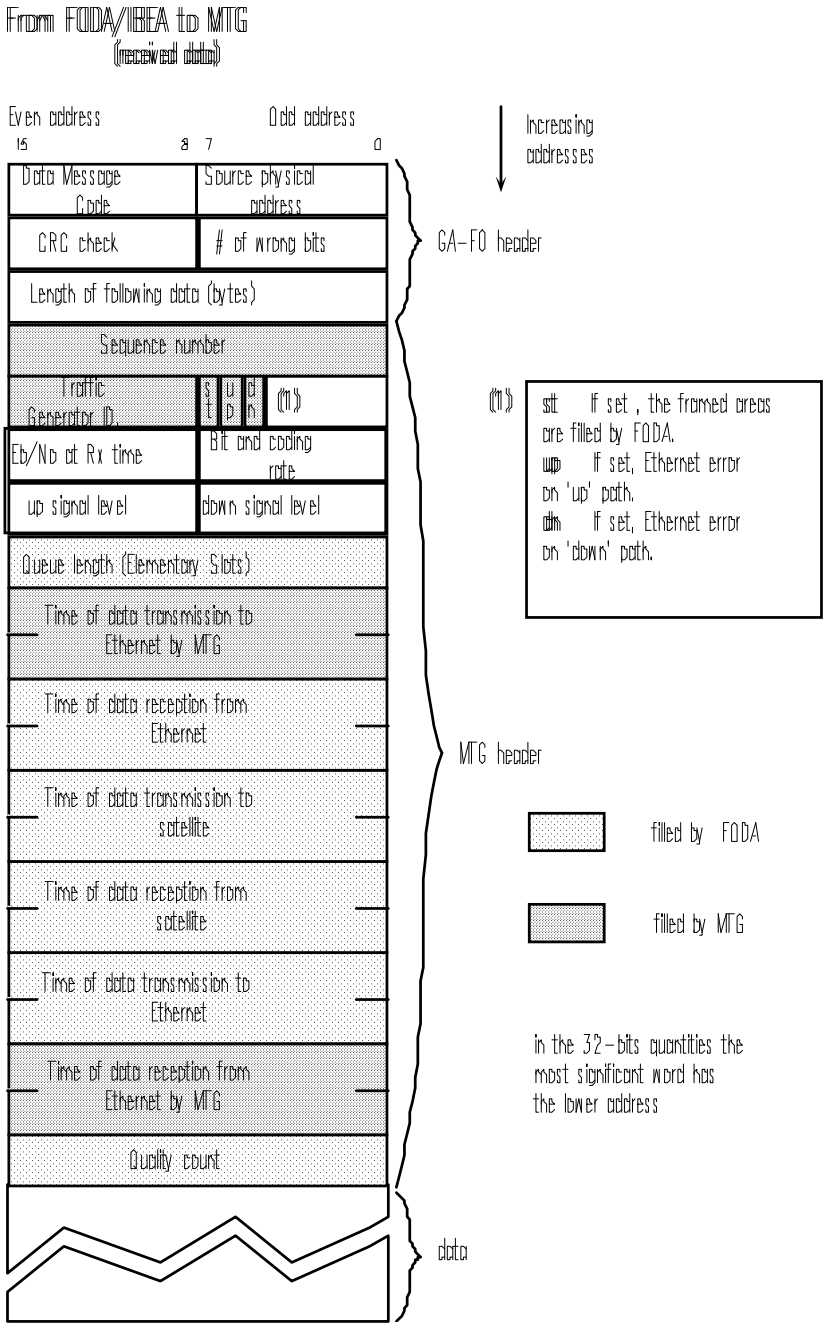

Fig .11. Header for data received by MTG from FODA/IBEA

Figs. 12.a and 12.b show the control message format and the data message format, respectively, as sent to the FODA/IBEA system via the Ethernet LAN.

Fig. 12a. Control message format

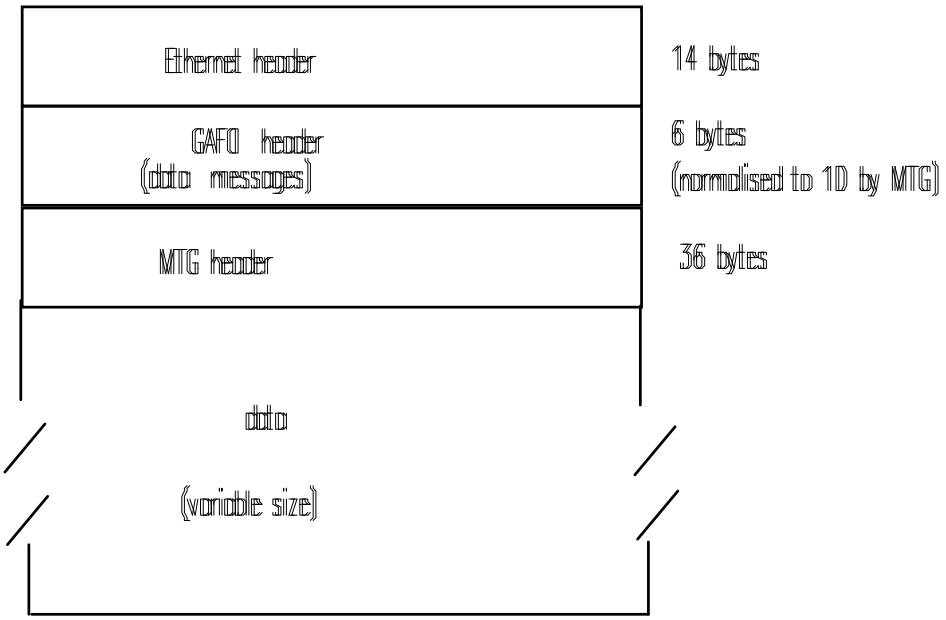

Fig. 12b. Data message format

CHAPTER 4. OPERATOR'S GUIDE

The Man-Machine Interface is implemented as a menu-driven dialogue, controlled by the MTG_MMD module. The Curses-Terminfo System V/68k package is used as the essential implementation tool [7]. The windowing concept is used for information exchange. Three windows are always present on the screen. On the upper side of the screen, a thin window (win1) with the basic MTG information is placed. The central part of the screen (win2) is reserved to display the input file contents and the system messages. The third window (win3), placed at the bottom of the screen, shows the list of the commands available to the MTG operator.

4.1 The input file

As first action to run MTG, the operator must prepare the file containing the TGs description (IDF). The name of such a file is chosen by the operator.
Any number of TGs, compatible with the hardware configuration, can be specified in the IDF. In the following, an example of input file is reported, where only one TG is defined. The definition of each TG must be contained in only one line. More than one TG may be defined in the input file.

```
#Date
#general information (1 line). Valid for all the TGs.
#
msem n   tick n     rbuf n   ttsz n   dkbf n   mask n   refr n
eth1 0xn1 0xn2 0xn3 0xn4 0xn5 0xn6     eth2 0xn1 0xn2 0xn3 0xn4 0xn5 0xn6
eth3 0xn1 0xn2 0xn3 0xn4 0xn5 0xn6     eth4 0xn1 0xn2 0xn3 0xn4 0xn5 0xn6
#
#description of the traffic generators (1 TG per line). Example of one TG.
#
tgid n          tabl n  data n dist t npkts t n1 n2   thro n1 n2
patt  t n       time t n1 n2   qlen n jitt  t n          cos n
opt  n1 n2      addr  t n      seed n         inta n1 n2            upcn n
nowb n          req c n1 n2    fnam c filename
#
```

Variable size records delimited by the # special character may be present wherever in the IDF, reserved for the operator comments. The first valid record must be the general information record, valid for all the TGs. It must be unique in the input file and must precede the record(s) describing the TG(s). A variable number of records follows, each one describing one traffic generator.

Each record contains the mnemonic name of the used variables, each one followed by one or more integer values. The name of the variables are for the operator convenience only. The length of the name of each variable can be from 1 up to 4 characters. A blank character must separate the name of the variable from the relevant values. A variable with its value(s) is called "*item*". A blank character or a tab character must separate two items. All the records are delimited by a <cr> (carriage return) character.

4.1.1 The input file parameters

The *general information record* describes parameters common to all the TGs.

msem n ==> semaphore threshold to get warned about its crossing.
The first time this value is crossed, a message is printed. It means that at least once, a packet left the MTG environment with a delay longer than *msem* msec.

tick n ==> tick length in μs.

rbuf n ==> byte size of the reception buffer.
The reception buffer is unique for the whole MTG system. Normally the reception buffer is allocated before the MTG system starting by means of the UNIX command:

```
rbuf get m
```

where m is the number of 4 Kbyte pages of memory to be allocated.
The buffer must be allocated in contiguous memory.
If at the MTG system starting time the buffer does not result to be allocated, it is allocated by using the *rbuf* value.

ttsz n ==> total number of entries in the timetables of the system.

To increase this value means to reduce the space available for the reception of the packets.

dkbf n ==> byte size of the disk buffer.
Data is copied into this buffer and then from this buffer onto disk.

mask n ==> when > 0, mask some general messages.
`n = 0x01` mask <TG not found> message;
`n = 0x10` mask <Unknown Data Code> message.

refrn n ==> refresh number.
After how many times the values of the "total received packets" must be refreshed when the command *p* is issued. If this value is equal to zero, the value is never refreshed (only frozen values are displayed).

eth Ethernet address. There are as many Ethernet addresses as the UPTDMA controller units are. To each upcn value (specified in each TG descriptor record) an Ethernet address is associated.
Each Ethernet address consists of 6 hexadecimal bytes.

Each *traffic generator record* describes one TG by means of the following parameters:

tgid n ==> traffic generator number .

tabl n ==> Timetable length (in number of entries) for each TG.
Each entry is 8 bytes long.

data t ==> data type**.**
`t = s` **stream.** Synchronous traffic. It requires a guarantied bandwidth on the satellite channel.
`t = b` **bulk**
`t = i` **interactive.**
Both bulk and Interactive data are asynchronous traffic (datagram). The bandwidth allocated on the

satellite channel and, consequently, the transmission delay, depend on the overall system loading conditions. Interactive traffic gets higher transmission priority than the bulk traffic.

dist t ==> distribution type.

- **`t = f`** **fixed.** The inter-arrival time between packets is constant.
- **`t = r`** **random.** The inter-arrival time between packets is constant, while the length of the packets takes random values in a given range.
- **`t = p`** **Poisson.** The inter-arrival time between packets is determined by the generation of pseudo-random numbers following a negative exponential distribution.
- **`t = v`** **voice**. The information flow of a packetized speech is simulated. No packet is generated if the "silent" period is long enough. The talk-silent periods are sorted according to a statistical distribution, relative to the English language.
- **`t = e`** **external**. The iinter-arrival time between packets is read from external input file.

n ==> coefficient for Poisson, if t = p was specified.

pkts t ==> packet type .

- **`t = f`** **fixed** size. Packet length is fixed and equal to: Ethernet header+ GAFO header + MTG header + fixed data size.
- **`t = v`** **variable** size. Packet length is variable and equal to: Ethernet header + GAFO header + MTG header + variable data size.
- **`t = c`** **computed** size. The packet length is computed by the program by using the *inta* and the *thro* values.

n1 ==> maximum packet size (in bytes)

n2 ==> minimum packet size (in bytes). It must be equal to n1 if t=f has been selected.

thro n1 ==> requested throughput (in multiple of 4 Kbit/s);
n2 ==> minimum acceptable throughput (in multiple of 4 Kbit/s).

patt t ==> pattern type
- **t = f** **fixed.** Pattern is fixed and equal to the chosen value.
- **t = r** **random.** Pattern is a random number in the range $0 \div (2^{16}-1)$.
- **t = i** **incremental.** Pattern is incremented word by word, starting from an initial value.

n ==> pattern value. It expresses the pattern Hex value (if t = f is selected) or the initial Hex value (if t = i is selected).

time t ==> time type
- **t = r** **relative** (to the start time of MTG)
- **t = a** **absolute** (with respect to Greenwich)
- **t = p** **Poisson** (according to Poisson distribution)
- **t = u** **user** (on user command).

Poisson is an end-less sequence of Poissonian starts with Poissonian duration. It is used to simulate a phone calls traffic.

n1 ==> start time (hours: minutes: seconds) or the mean Poissonian call time;
n2 ==> duration (hours: minutes: seconds) or the mean Poissonian call duration time.

qlen n ==> max number of packets the FODA/IBEA system maintains in the queue before starting to discard them.

jitt t ==> jitter type.
- **t = n** NO. Data must not be generated with initial jitter.
- **t = y** YES. Data must be generated with an initial jitter.

If t=y, a further delay between 0 and *n* ms, sorted in a uniform distribution, is added to each packet dispatching time.

n ==> jitter value (if *y* specified).

cos n ==> class of service number.

To each class of service number a BER range corresponds. Classes 1-4 (used in normal operating mode) specify that the data bit error rate must be kept within specified limits:

`n = 1`	$BER \leq 10^{-8}$
`n = 2`	$10^{-8} \leq BER \leq 3 \cdot 10^{-7}$
`n = 2`	$3 \cdot 10^{-7} \leq BER \leq 3 \cdot 10^{-5}$
`n = 4`	$3 \cdot 10^{-5} \leq BER \leq 3 \cdot 10^{-3}$

Higher classes (5-20) are used to force the bit and coding rates which the system must use to send the data.

opt n1 ==> option mask.

`n1 = 0x80` Set *st* flag.

The *st* flag is set in the MTG header, so the FODA/IBEA system fills the following fields:

- Eb/No at Rx time
- Bit and coding rate
- Transmit power level
- Receive power level.

`n1 = 0x40` Send data with CRC.

Data are requested to be sent on satellite, by the FODA/IBEA system, appending the CRC. The CRC is checked on reception and a flag is set in the GAFO header if it is wrong.

`n1 = 0x20` Compute number of wrong bits.

The received packets contents are compared with the transmitted ones and the number of wrong bits is set in the GAFO header. This option allows the estimation of the bit error rate.

`n1 = 0x10` Record data every k^{th} .

One packet header, every *k* received, is recorded onto the *MTGout* output file.

k is specified in the next parameter.

If this bit is cleared, no registration is done.

`n1 = 0x01` Hide some messages.

For each TG, the following messages are not displayed to the operator terminal:

- TG #n: Sequence error.
  Expected = n1, received = n2.
- TG #n: Incomplete stream (datagram) received.

The counters relevant to the incomplete and to the out-of-sequence data packets are displayed by means of the *p* command.

n2 ==> k value, if n1=0x10 was specified.

addr t **==>** address type:

`t = m` **to myself**

`t = s` **selected station**

`t = a` **to all (broadcast)**

n ==> destination physical address.

If t = m, n is without meaning. The upcn value will be used as destination address.

If t = s, n is the selected destination address.

If t = a and n = 0, real broadcast.

If t = a and n >0, data are sent in broadcast to the stations belonging to the user group n.

seed n ==> value of the seed for the random number generation for this TG.

inta n1 ==> minimum inter-arrival time (in ms)

n2 ==> maximum inter-arrival time (in ms).

If both the *pkts* and *thro* fields are specified, this field is ignored.

upcn n ==> UP controller number.

To each UP controller number, an Ethernet address is associated. It is used when more FODA/IBEA stations are attached to the same Ethernet cable.

nowb n ==> number of wrong bits.

The operator may express the number *n* of wrong bits that the TG must collect before to display to the terminal, in real time, the relevant computed BER (wrong bits / total bits). If the command *p* was issued, the real time BER value is updated.

If *n = 0* is specified, only when the TG stops the relevant BER is computed and displayed. In this case, the BER relevant to that TG was computed considering the whole activity time of that TG.

If *n > 0* is specified, the BER is displayed each time that TG collects the specified number of wrong bits.

req c n1 n2 ==> Politics used to compute the stream request.

Meaningless in case of datagram TGs.

The stream request is computed as:

c = p meaning that n1 and n2 are the percentages of the troughput used for computation of the stream request.

n1 = n2 = 100 implies that channel request cannot be higher or lower from needed channel space.

c = a n1 and n2 are the requests made for stream applications in Kbit/s.

fnam c filename ==> input file for external distribution.

Filename is the name of a file where a timetable is stored.

Each entry is a line with two blank separated integer numbers in ASCII format (*Tn Ln*). The first number (*Tn*) represents time-triggering value of the packet whose size is given as second integer (*Ln*).

We are considering two time references: *absolute and relative*.

The first packet is sent at time T0.

For c = a (absolute reference) the second packet will be sent at *T1* time, third at *T2* time and so on.

For c = r (relative reference) the second packet will be sent at *T1* time, third at *T1 + T2* time and so on.

When the input file is ready, MTG can be started.

4.2 How to start MTG

MTG is started simply by invoking:

mtg `IDF-name`

Each new session of MTG writes on the special archive file *MTGout* the data from the very beginning of the session.

The *MTGout* file contains the GAFO headers and then MTG headers of all the received data packets and of the received control messages. The header of the MTGout file contains the copy of the current session input file. The MTGout file ends with a special header containing the CODE = 0.

During the session all the data are written sequentially in their own arrival order.

To prevent the file overwriting, when MTG is started, the operator is warned by the following message:

MULTI-APPLICATION TRAFFIC GENERATOR GETTING START

***** WARNING ** This run will overwrite previous results.***
Please save your archive file.

Continue? [y/n]:

If **n** is replied, MTG doesn't start and the user can save the archive file of the previous session (for example, re-naming it).

If **y** is replied, MTG starts and on the operator terminal it appears:

```
MULTI-APPLICATION  TRAFFIC  GENERATOR                    Tick_len
CNUCE - CNR      Version 2.0 September 1993                  n




r          Run TG(s)                e          Ends TG(s)
l          List TGDs, TCBs          q          Quit
p          Packet counters          h          Halt refreshing

Enter your choice >
```

At this point, the user must enter a command.

4.3 The commands

In the following, the actions taken by the MTG system according to the entered command are described.

4.3.1 R

This command starts one specific TG or all the traffic generators specified in the IDF.

When **r** is typed, on the bottom of the screen (win3) it appears:

```
Specific TG or ALL [s,a]? >
```

- to start a specific TG:

  `s <cr>`

  which one? > `TG number <cr>`
- to start all the defined TGs:

```
a <cr>
```

4.3.2 L

This command displays some information from the TGDs or from the TCBs. When issued, the operator is requested to choose which display is required. On the bottom of the screen (win3) it appears:

TGDs or TCBs [d, t]? >

- to display some contents of the TGDs:

```
d  <cr>
```

- to display the TCBs:

```
t  <cr>
```

4.3.2.1 D

Some information got from the TGDs is displayed on the central screen (win2) according to the format described in the following. One line for each TG defined in the input descriptor file is displayed.

TRAFFIC GENERATOR DESCRIPTORS

TG	*TR*	*DS*	*THRO [R/m]* [kbps]	*PKTL [M/m]* [bytes]	*INTA [m/M]* [ms]	*PATT* T [h]	*COS* [h]	*OPT* [h]	*DA dd* T [h]	*QLen* [d]	*JITT* [h]	*UpCTL* [d]

where:

TG traffic generator number.

TR traffic type:

```
s = stream
```

```
b = bulk
i = interactive
```

DS distribution type:

```
f = fixed
r = random
p = Poisson
v = voice
e = external
```

THRO throughput.
R / m = requested / minimum (acceptable).
Both the values are expressed in Kbit/s. The displayed throughput generally is a little bit different from the values indicated in the IDF because it is recomputed on the basis of the chosen packet lengths and inter arrival time values.

PKTL packet length.
M / m = Maximum value / minimum value.
Both the values are expressed in bytes.

INTA inter-arrival time (for stream traffic).
m / M = minimum value / Maximum value.
Both the values are expressed in ms.

PATT pattern. The pattern is an hexadecimal value (h).
T is the pattern type. It may be:

```
f = fixed
r = random
i = incremental.
```

COS class of service. Decimal value (d) .

OPT option value. Hexadecimal value (h).

DAdd destination address. Hexadecimal value (h).
T is the address type. It may be:

```
all     broadcast
```

```
s n     selected (s) station n
g n     user group (g) n
m n     to myself (m); n is the upcn value.
```

QLen queue length (in number of packets) before discarding. Decimal value (d).

JITT jitter value. Hexadecimal value (h).

UpCTL number of the UP controller the MTG is connected to. Decimal value (d).

4.3.2.2 T

The contents of the Time Control Blocks are displayed, one line for each TG. On the central screen (win2) it appears:

TG-id	*STATE*	*TIME-TYPE*	*START/STOP or DURATION TIME [hh:mm:ss]*	*SEQUENCE NUMBER decimal]*

where:

TG-id traffic generator number.

STATE state of the traffic generator. See the list of the possible states in 3.3. Note that in the display "W-" means "wait for..".

TIME-TYPE time type. It may be:
`relative`
`absolute`
`Poisson`
`user.`

START... The six displayed parameters express the start (hh:mm:ss) and the stop time (hh:mm:ss) for time type = relative and time type = absolute, and the *call* and *duration* times, respectively,

for time type = Poisson. If time type = user is selected, the TG starts and ends on user command only.
The value time type = Poisson is used to simulate phone calls. The start and the duration times are random numbers with a negative exponential distribution with mean values equal to the *call* and *duration* times, respectively.

SEQUENCE — sequence number of the last received packet at the command time.

4.3.3 E

This command stops one specific TG or all the traffic generators specified in the IDF. When a TG is forced to stop, all the relevant allocated areas are deallocated. If the TG was devoted to handle stream traffic, the allocated stream channels are released.
When **e** is typed, on the bottom of the screen (win3) it appears:

Specific TG or ALL [s,a] ? >

- to end a specific TG:

  `s <cr>`
  which one? > `TG number <cr>`

- to end all the defined TGs:

  `a <cr>`

When a TG is forced by the operator to end by means of the *e* command, the message:

TG # n --> LSN =n, TOTNoWb = n, TOTNob = n, TOT-BER = n.

appears, where:

LSN sequence number of the last received packet for the specified TG.

TOTNoWb number of wrong bits collected. If nowb 0 was specified in the input file for this TG, NoWb is the total number of wrong bits collected from the start of this TG. Otherwise, it is the total number of wrong bits collected from the last real time printing of the BER.

TOTNob total number of received bits from the start of this TG.

TOT-BER computed BER, following the same rule as for NoWb.

4.3.4 Q

This command **must be issued after the command *e*** to exit from the MTG environment. If issued before *e*, the allocated areas and the allocated stream channels are not released.

4.3.5 P

For each TG, this command displays information relevant to the packets received back by MTG after the satellite transmission.
The displayed values are:

- the number of out-of-sequence data packets
- the number of incomplete data packets
- the total number of received data packets (the out-of-sequence and the incomplete data packets are also included in this value)
- the BER computed in real time, i.e. each time the collected number of wrong bits in the packets exceeds the *nowb* value specified in the IDF

- the total number of wrong bits collected by this TG

- the number of packets having at least one wrong bit (packets with CRC error).

If the option 0x01 is not specified in the IDF for a certain TG (see *opt* field in 4.1.1), the error messages relevant to the out-of-sequence and to the incomplete packets will appear also on the central screen (win2) of the operator terminal at their occurrence for that TG.

Relevant to the whole traffic generated by all the started TGs, the command *p* displays:

- the total number of received packets whose control code was unknown

- the total number of received packets whose originating TG has been not found when returned to the MTG system.
  This value is comprehensive of the following cases:

  - the TG identifier is zero
  - the TG identifier is unknown
  - received bulk data is erroneously associated to a non-bulk TG
  - received interactive data is erroneously associated to a non-interactive TG
  - received datagram data is erroneously associated to a non active TG.

- the total number of packets (control or data) received from the LAN with errors.

Packets received back by MTG without LAN errors are checked for the code validity. Packets with an unknown code are discarded and the relevant counter is incremented. If the data packet code is correct, the originating TG is searched for. If not found, the *TGs not found* counter is incremented and the value updated on the screen.
Both these values must be zero.

When command *p* is entered, on the central screen (win2) it appears:

PACKETS INFORMATION

Unknown Codes -> n TGs not found -> n LAN errors data (?) -> n

TG TRAFFIC OUT-OF-SEQUENCE INCOMPLETE TOTAL RT-BER TOTNoWb With-CRC

where:

TG	traffic generator number.
TRAFFIC	stream, bulk, interactive or stream.
OUT-OF-SEQUENCE	number of out-of-sequence packets received by this TG. This value is updated in real time, i.e. it is refreshed each time the relevant event occurs.
INCOMPLETE	number of incomplete packets received by this TG. This value is updated in real time, i.e. it is refreshed each time the relevant event occurs.
TOTAL	total number of packets received by this TG. The number of the out-of-sequence and of the incomplete packets is included. As this value changes too quickly, it is refreshed only if the parameter *refr* is specified different from zero in the general information of the IDF. Otherwise, the frozen value at the *p* command issuing time is displayed.
RT-TIME	real time BER. It is different from 0 if nowb > 0 is defined in the IDF.
TOTNoWb	total number of wrong bits collected by this TG.

With-CRC	number of packets received with CRC error. This value is updated in real time, i.e. it is refreshed each time the relevant event occurs.

4.3.6 H

This command acts only on the screen displayed by the *p* command. It stops at any time the refreshing of the displayed values. To restart the updating, the *p* command must be issued again.

CHAPTER 5. THE PERFORMANCE OF MTG

The goal of the authors is to present the performance of the MTG system independently of the used satellite access scheme. Therefore the FODA/IBEA system has been simulated by means of an Ethernet responder which redirected to MTG the generated data. Results about the performance of the FODA/IBEA system measured and by means of MTG and in the real environment will be presented elsewhere.

# of TGs	Inter-packet generation time for each TG	packet length (bytes)	traffic volume in TX	traffic volume in Rx	header per # packets	test run #
2	1 ms 512 Kbit/s per TG	64	1.024 Mbit/s	1.024 Mbit/s	1	1
3	1 ms 512 Kbit/s per TG	64	1.536 Mbit/s	1.536 Mbit/s	100.000	2
8	1 ms 512 Kbit/s per TG	64	4.96 Mbit/s	simplex	0	3
32	32 ms 69.5 Kbit/s per TG	22+256	2.224 Mbit/s	2.224 Mbit/s	1	4
90	32 ms 69.5 Kbit/s per TG	22+256	6.255 Mbit/s	simplex	0	5
43	62 ms 195 Kbit/s per TG	1514	8.4 Mbit/s	simplex	0	6
22	30 ms 404 Kbit/s per TG	1514	8.882 Mbit/s	simplex	0	7
12	16 ms 757 Kbit/s per TG	1514	9.084 Mbit/s	simplex	0	8
4	5 ms 2345 Kbit/s per TG	1466	9.382 Mbit/s	simplex	0	9

Table 1. Performance evaluation tests scenario

In Table 1 the MTG performance under different test conditions is presented. Each test was run for 300 seconds. Ethernet was used both in simplex and in full-duplex. 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 transmit side architecture of MTG. In fact only one packet at a time can be handled in order to set up accurate time stamps. Anyway, the main purpose of MTG is to allow adequate accuracy in the parameters estimation, rather than to attain the maximum possible throughput on Ethernet.

An Ethernet cable connected MTG with the responder during all the tests aimed at measuring the recording capabilities of MTG. The traffic looped back by the responder was collected by MTG and the GAFO header plus the MTG header of each received packet (see Fig. 11) were recorded onto disk. The responder was not used in the tests where the traffic generation performance only was investigated.

Tests 1÷3 were aimed at measuring the maximum number of packets per second MTG can handle. Transmission only and transmission/reception with and without disk recording were the test conditions. The packet length does not influence the MTG performance, as long as Ethernet can be viewed as an infinite bandwidth medium, so very small packet length was used to avoid collisions.

Tests 4 and 5 give the maximum number of supportable 64 Kbits/s applications. Two results are given, with and without the packet header recording onto disk. The packet length was set to 256 bytes plus a 22-byte header containing the Ethernet header and the GAFO header.

In the last five tests the maximum reachable total throughput was investigated with different numbers of running generators. The results are plotted in Fig. 13. The packet length was chosen as long as possible, in order to reduce the overhead ($\sim$60μs) MTG imposes on each transmitted packet. Fig. 13 shows clearly how the total throughput increases when the number of generators decreases, due to the overhead required by MTG for each traffic generator. The throughput with only one generator is smaller because this test was performed with a shorter packet.

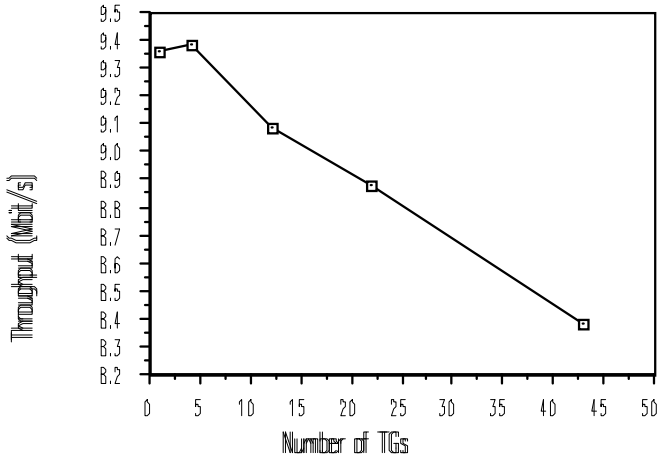

Fig. 13. Max throughput as a function of the number of TGs

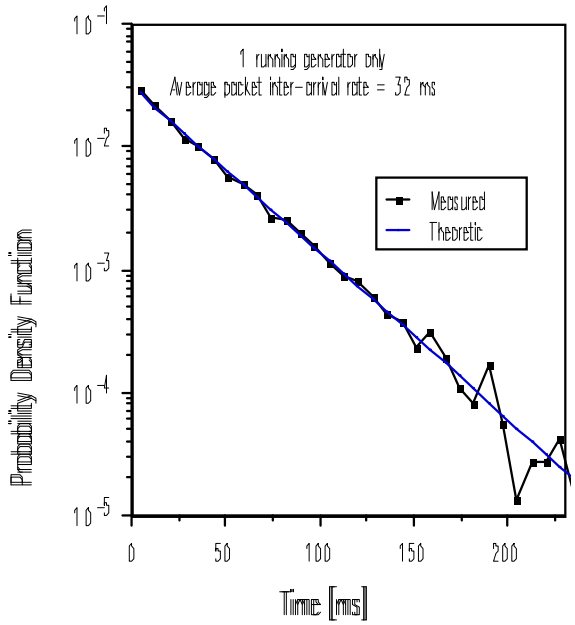

Fig. 14. Packet sending times distribution measured for 1 Poisson generator

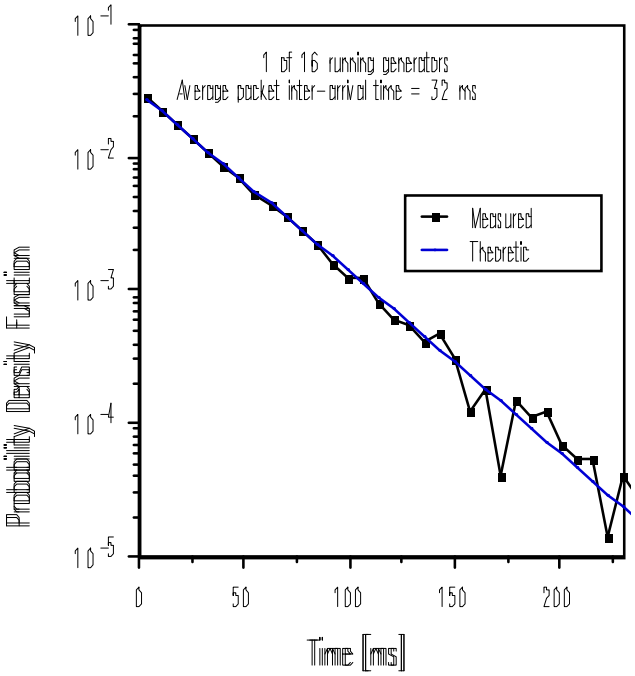

Fig. 15. Packet sending times distribution measured for 1 of 16 Poisson generators

The ability of the system to generate packets according to a chosen statistical distribution is shown in Figs. 14 and 15.
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 (a run of about 300 s). The generator was running alone (Fig. 14) or together with 15 other generators (Fig. 15) of the same distribution type and with throughputs ranging from 16 to 256 Kbit/s. It can be seen that the offsets from the theoretic 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 sensibly worsen the performance.

It must be pointed out that the precision of the measures made by MTG is hindered by the limited bandwidth of Ethernet. An unlimited bandwidth medium would be required to insure that the scheduling times of the packets generated by each TG were the computed ones. Indeed, all the effects introduced by Ethernet, such as delays, errors or collisions, are undesired effects — even though they can be easily filtered during the analysis of the collected data. The effects of the limited bandwidth medium become significant only when Ethernet is loaded beyond a certain level, above which the validity of the measures is impaired. When such a high throughput is requested, our implementation (based on Ethernet) would require the introduction of one more Ethernet board in the MTG hardware and the adoption of two LAN stubs — one for transmission and one for reception.

CHAPTER 6. MEASURES WHICH CAN BE PERFORMED

The four FODA/IBEA time stamps present in the MTG header allow the measurement of:

- the application end-to-end delay,
- the satellite network delay,
- the queuing times of packets received by Ethernet and waiting for transmission to the satellite,
- the queuing times of packets received by the satellite and waiting for transmission to Ethernet,
- the packet jitter before and after the satellite network crossing.

The end-to-end delay is experienced by a packet from when it leaves the generator until it arrives back to it and is recorded onto disk. This parameter is significant only when a packet is sent by an MTG system and received back and recorded by the same system.

The queuing delays are the times spent by each packet inside the UP and the DOWN controllers. The packet queue lengths can be monitored as well. All these quantities are extremely useful in tuning the FODA/IBEA system parameters in order to improve the performance. Generally, it is possible to retrace the history of the delay and of the jitter of each packet at different test points of the system on trial.

It is also possible to tune-up the fast Eb/No estimator (internal to the controller), based on the soft decision level of the data coming from the demodulator and to check the choice of the bit and coding rates made by the system during variable conditions of the channel quality.

The FODA/IBEA system supports variable-bandwidth applications which can be requested to reduce their bandwidth in particular fade conditions of the station. The

simulation of these types of applications is another feature supported by MTG. The congestion problem can also be studied and verified.

Another measure performed by the MTG system is the real-time BER measurement, by means of the *nowb* parameter specified in the input file (see 4.1.1).
Reassuming, the measurements which can be performed by means of MTG are relevant to:

- the application end-to-end delay,
- the satellite network delay,
- the queuing times,
- the packet jitter,
- the congestion,
- the variable throughput,
- the fast Eb/No estimator,
- the choices relevant to the fade countermeasure,
- the BER measurements.

Statistics on the data recorded onto disk by MTG are performed by means of a specialised statistical analysis program (MTGSAP), running on the UNIX operating system, which has an interactive interface to the user [12].

The main functions of the programs constituting MTGSAP are listed in the following.

- To copy the data recorded by MTG on a file named by the user. In this way, data relevant to different sessions may be compacted.
- To filter the data according to different parameters and/or logic connections of the parameters themselves. An ASCII file is created (named by the user)

containing the data relevant to the packets whose parameters are in the range of values specified by the user.

- To counter the total number of received packets, of the total amount of transferred data and of the total length of the test.

- To generate the statistics of: total, maximum, minimum, average, variance and given percentile. The statistics are computed on the data contained in the MTG header, on the jitter, the number of wrong packets, the number of incomplete packets, the number of miss packets and the number of repeated packets.

- To compute the distribution functions (as the probability mass function p_i and the cumulative distribution function C_i) for significant parameters specified in the MTG header and for various parameters relevant to the jitter. The program defines a series of n value intervals and computes p_i and C_i for each of the computed interval. The two functions are defined as the total number of occurrences in the interval $([X_i , X_i + \Delta V] / K)$ and as the sum of p_i, respectively.

- To compute the bit error rate (BER) with relative confidence interval, if the packets are collected in specified time intervals. The program foresees two options to compute the BER: the standard deviation or the time interval. If the interval of time where the BER must be calculated is specified, the program computes the BER and the standard deviation at each time interval. If the standard deviation (sigma) is specified, the program computes the BER and the time intervals necessary to collect a number of wrong bits to get the BER with the specified standard deviation.
  Being $p << 1$, the formula $sigma^2 = n\ p\ (1-p)$ giving the variance of the BER, has been approximated to $sigma^2 = n\ p$ where sigma is the BER standard deviation, n is the number of investigated bits and p is the bit error probability.

- To perform the CCITT G.821 analysis . The description of the CCITT Rec. G.821 can be found in (CCITT Red Book 1984).

  Definitions necessary to the understanding of this function are here reported, keeping into account that a 64 Kbps channel is considered.

Errored second is any second in which at least one bit (but less than 64 bits) is wrong. Therefore, the BER results: $1/64000 <= BER < 10^{-3}$.

Severely errored second is any second in which at least 64 bits are wrong. Therefore, the BER results $>= 10^{-3}$.

Degraded minute is any interval of 60 sec with BER $>10^{-6}$.

Available time: is all the investigation time minus the unavailable time.

Unavailable time is a period of unavailable time begins when the BER in each second is worse than 10^{-3} for a period of 10 consecutive seconds. These 10 seconds are considered to be unavailable time. The period of unavailable time terminates when the BER in each second is better than 10^{-3} for a period of 10 consecutive seconds. These 10 seconds are considered to be available time.

The following values are computed:
- the interval of time where the analysis is made (in sec),
- the total available time (in sec),
- the percentage of severely errored seconds (referred to the available time),
- the percentage of errored seconds (referred to the available time),
- the percentage of degraded minutes (referred to the available time).

REFERENCES

[1] Celandroni N., Ferro E., Mihal V., Potorti' F.
"FODA/IBEA-TDMA. Satellite access scheme for mixed traffic at variable bit and coding rates. SYSTEM DESCRIPTION", CNUCE Report C92-05, April 1992.

[2] Celandroni N., Ferro E., James N., Potorti' F.
"FODA/IBEA: a flexible fade countermeasure system in user oriented networks", International Journal of Satellite Communications, Vol. 10, N. 6, pp. 309-323, November-December 1992.

[3] Celandroni N., Ferro E., James N., Potorti' F.
"Design and implementation of a flexible, software based TDMA system", proceedings of the IEEE Global Telecommunications Conference, GLOBECOM' 92, December 6-9, 1992, Orlando (Fl), USA.

[4] 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.

[5] N. Celandroni, E. Ferro
"The GAFO Protocol. The protocol to access the FODA/IBEA system", CNUCE Report C94-03 / Rel. 3.0, January 1994.

[6] G. Iazzeolla
"Introduzione alla simulazione discreta", Serie di Informatica, Boringhieri.

[7] Motorola UNIX SysV68 Documentation.

[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, N. 4, pp. 233-251, August 1983.

[12] Consorzio Pisa Ricerche
"MTGSAP. Statistical Data Analysis Software Package", September 1993.

[13] Alcatel 8643
"ATM Traffic Generator. Alayzer ATGA.", 1992.