FRACAS
FRAmed Channel Access Simulator

USER MANUAL

Nedo Celandroni
Erina Ferro
Francesco Potortì

CNUCE/C.N.R. Institute
Via S.Maria 36 - 56126 Pisa (Italy)

Tel. : +39-50-593207 / 312 / 203
Telex: 500371 CNUCE
Fax : +39-(0)50-904051 / 052

e-mail: {n.celandroni | e.ferro | pot}@cnuce.cnr.it

CNUCE Report C95-27

November 1995

TABLE OF CONTENTS

LEGEND

a is optional.

one or more instances of **a**.

a|b|…|f	one and only one item among **a**, **b**,…, **f**.
TRU	Traffic Unit. Data size unit measurement.
TMU	time Unit. Time unit measurement. The traffic unit measurement is TRU/TMU.
observable	any measurement on which statistics can be collected.
history	the values assumed by an observable.

1. INTRODUCTION

FRACAS (FRAmed Channel Access Simulator) is a simulation tool for TDMA satellite networks, which provides users with a library of satellite channel access schemes in TDMA.

Once the network configuration has been defined, the user can choose the band allocation policy from a set of predefined policies. Statistics on the performance of the chosen access scheme in the conditions of the network simulated are then collected.

FRACAS is aimed at all those research centres in the field of communications via satellite in TDMA, and to service suppliers who need support from simulative studies in order to choose the best allocation policy and to tune up the relevant parameters.

FRACAS is useful for comparing different satellite access schemes in traffic load conditions chosen by the user. Three classes of traffic,called *datagram, stream and VBR* are considered. *Datagram* includes all the jitter-tolerant applications, *stream* and *VBR* include all the real-time applications.

Datagram traffic is a connectionless type of traffic without any particular delay requirements. It can tolerate out-of-order delivery of packets and a high jitter, but usually it requires a low bit error rate. The delay introduced by the network(s) crossing is not a critical constraint. However, especially on a high delay network, like the satellite network, the end-to-end throughput of such traffic can be heavily impaired by bit errors or packet losses that trigger retransmissions in higher level protocols.
Datagram packets delivery is not guaranteed in the sense that the datagram transmissions may be momentarily suspended (e.g. when congestion is detected), and that packets exceeding the system buffering capacity may be dropped. Moreover, the delay a datagram packet experiences can 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 the satellite network should maintain a low and constant delay on the arrival of the information. This kind of traffic is generated by applications that have constant throughput, like voice, slow-scan TV, fixed rate video conference, measurement data and so on. It is referred to as FBR (Fixed Bit Rate) traffic.

By *VBR* we mean Variable Bit Rate video, which is one of the most interesting and challenging real-time applications. Constant quality VBR encoders attempt to keep the quality of video output constant, resulting in a highly variable and bursty output bit rate. VBR video traffic is both highly variable and delay sensitive, so transmission over high-speed networks (for example ATM) is generally implemented by assigning peak rate bandwidth to VBR video applications, and by using the residual bandwidth for non-real-time traffic.

2. OVERVIEW OF FRACAS

FRACAS is a batch program that reads its input from one or more files and outputs the results onto one or more files or terminals. The general syntax for invoking FRACAS is summarised in the following screenful, which FRACAS prints when it is called with no arguments:

```
usage: fracas [options] files
Where options are:

-e     print all known values on stdout, skip all computations
-s     print the entire source file before the computation results
-t     terse - do not print headers on standard output
-v     verbose - write messages showing the process of computations
```

At least one input file name must be specified on the command lines, but several are accepted. The input files are individually read and elaborated by FRACAS.

The **`-e`** (emulate only) option skips all the computations on the output values, runs the emulator and writes all observable values on the standard output. Only the packet delay observable (*tru_delay*) is not printed, because it requires a computation to be evaluated. The `-e` option is useful for using FRACAS as an emulator and for giving the task of computation to some other program.

The **`-s`** (source file) option is useful when the output of FRACAS is redirected to a file, that, at the end of the run, will contain both the source of the run and its results.

The **`-t`** (terse) option is useful when the output is redirected to a program.

The **`-v`** (verbose) option is useful for looking at the progress of the simulation.

FRACAS is a discrete time simulator. *Everything in the emulation happens at intervals of one frame, and no measure makes sense in between*. In the input file, the length of a frame is defined as an integer number of TMUs. *This is useful only to have all the times expressed in some handy time unit, but does not affect the behaviour of the simulator in any way*. For example, if a particular simulation uses a time frame 20 ms long, it is useful to express all the times in milliseconds, by setting the time frame to 20 TMUs. This version of FRACAS does not allow a non integer time frame to be specified.

Moreover, the time frame must be kept as small as possible if the packet by packet delay has to be computed. This is because the maximum computable must fit into 16 bits (2^{16}-1=65535). In our example packet by packet delays of up to 65 seconds could thus be computed. There is no limit on the delays if they are measured as a frame by frame mean delay. Since all the timings are multiple of a frame, no delay shorter than this quantization unit can be resolved. This implies, for example, that if in the real system the delays are less than one frame long, they are rounded down to zero and invisible to FRACAS.

In the input files keywords are used to define the network configuration, the channel allocation policy and the statistics to be collected. The syntax is described in the Appendix. The main entries in the input file define:

* the time length and the data size of the frame,

* the configuration of each station,

* the channel allocation policy,

* the statistics to collect,

* the duration of the simulation,

* the *end* keyword.

- *Station Configuration*

For each station the following parameters can be defined:

* maximum queue length for stream, VBR and datagram,

* stream request,

* maximum and minimum VBR request.

For each station an arbitrary number of different traffic generators can be defined. Examples of traffic generators are:

* Constant rate,

* Periodic on-off with constant rate bursts,

* Poisson,

* Two state Markov-modulated Poisson,

* Fractional Gaussian Noise.

Additional traffic generators can be added by coding them in C language.

- *Channel Allocation Policy*

Users can choose the channel allocation policy for a simulation session. The available policies at the time of writing include:

* Fixed TDMA
* FODA/IBEA (Fifo Ordered Demand Assignment / Information Bit Energy Adapter)
* FID/VBR (FODA/IBEA Derived / Variable Bit Rate)
* DRIFS (Distributed allocation with Requests In Fixed Slots)
* FEEDERS (Faded Environment Effective Distributed Engineering Redundant Signalling)

Apart from Fixed TDMA which is a well known policy, the others were studied at CNUCE. In particular, FODA/IBEA (which has a centralised control) was developed in the framework of the Olympus project and its validity has been confirmed by simulative and experimental results. FODA/IBEA does not support VBR traffic. FID/VBR is the FODA/IBEA protocol with support for VBR video traffic. DRIFS and FEEDERS are distributed control protocols. Work on FRACAS was begun to evaluate the performance of these last two protocols, and it is expected to grow to include other protocols, which can be easily added by coding them in C.

- *Computers*

Once the allocation policy has been chosen, various statistics such as minimum, maximum, average, variance, percentile and sample listing on the following observables can be collected:

* input traffic
* length of the input queues
* output traffic
* unused allocation space
* frame-by-frame and packet-by-packet delay
* packets lost due to internal buffer overflow

Additional statistics on the observables listed above can be added by coding the relevant routines in C.

- *The end keyword*

The *end* keyword on a line after the *computers* section means that the description of a run is finished and another one follows. Therefore, an arbitrary number of runs can be concatenated into a single file. The last file needs no *end* keyword. To concatenate runs into a single file is the same as writing them in different files and listing the file names on the FRACAS command line.

3. REQUEST ALGORITHMS

A requester function is called at each frame for each station. It computes the request the station makes in that frame for each of the three kinds of traffic (stream, VBR, datagram). All the stations in FRACAS share the same allocator, so only one allocator needs to be specified.

3.1. queue

All the requests are equal to the length of their respective input queues. This requester is only useful for simple tests on the behaviour of the FRACAS simulator itself.

3.2. fodaibea (FODA/IBEA and FID/VBR protocols)

This requester is supposed to be used together with the *fodaibea* allocator (see below). When used together, they behave like the FODA/IBEA or FID/VBR protocols.

The stream request is equal to *sreq* as specified in the station definition.

The VBR request is equal to the VBR *minreq* or *maxreq* specified in the station definition. The *maxreq* value is requested when the input traffic is greater than the VBR *minreq* value. The input traffic is measured as the average input in a sliding window whose length in frames is the fodaibea requester parameter *vwin*. The *minreq* value is requested when the input traffic is less than or equal to *minreq*.

The datagram request is proportional to the traffic coming into the station plus the backlog, i.e. the volume of data waiting for transmission to the satellite:

$$request = H\ traffic + backlog$$

where H is a temporal constant. The input traffic is updated every *dwin* frames; it is computed as the average of the input in the last *dwin* frames. Both H and *dwin* are parameters of the requester. A datagram request in a frame can only be made if the station has some allocation (either stream, VBR or datagram) in that frame. If not, no datagram request is made.

3.3. feeders-drifs (FEEDERS and DRIFS protocols)

This requester is supposed to be used together with the *feeders* or the *drifs* allocators (see 4.3 and 4.4). When used together, they behave like the FEEDERS or DRIFS protocols, respectively.

The stream, VBR and datagram requests are made in the same way as the *fodaibea* requester does. The only difference is that in every frame a datagram request is always made.

4. ALLOCATION ALGORITHMS

An allocator is a function called at each frame. It looks for the requests that the stations made a given number of frames before (the number depends on the allocator type), and computes the allocations for all the stations. Only one allocator can be specified.

4.1. fixed (Fixed TDMA)

The stream traffic receives an assignment equal to the request *sreq*.

The VBR traffic receives an allocation equal to the max VBR request *vmaxreq*.

The remaining space, equal to *framesize* minus the sum of the stream and VBR allocations, is evenly divided among all the stations and assigned as a datagram allocation.

4.2. fodaibea (FODA/IBEA and FID/VBR protocols)

FODA/IBEA is a satellite access scheme for simultaneous transmissions of both stream and datagram data. The quality of service is maintained even when the system is faded, i.e. the transmission signal is attenuated due to bad atmospheric conditions. VBR traffic is not supported [1,2,3].

FID/VBR is a version of FODA/IBEA modified in a such a way as to support VBR traffic too [4, 5].

Both access schemes (FODA/IBEA and FID/VBR) have centralised control. In terms of delay, the delay between a request and the relevant assignment is twice the round trip time *rttitme* divided by *frametime* rounded up plus three. This accounts for all the transmission and the processing delays.

The channel overhead is due to a traffic independent part consisting of the reference burst, an FAS (First Access Slot) every 32 frames (by default), and some extra space needed by the hardware. The reference burst is sent by the control station for synchronisation and contains the assignments. The FAS is a space used by the stations to enter the satellite network. Other overheads are traffic dependent: the only one FRACAS considers is the burst overhead (one per frame), which is the unused space subtracted from the total frame space every time an allocation is given to a station. The *fodaibea* allocator implements FID/VBR, as a superset of FODA/IBEA. However,

since FRACAS has no means to change the signal quality, support for variable satellite channel speed and coding rate is not implemented.

The stream allocation is equal to the stream request.

The VBR allocation is equal to the VBR request.

The datagram requests are organised into a ring, which is scanned to compute the assignments. The length of the assigned transmission window is proportional to the request in a range of values between a minimum and a maximum threshold (*mindall* and *maxdall*). The proportionality factor is proportional to the total number of stations in a range [*allnmin*; *allnmax*]. The complete expression for the assignment is:

$$\mathrm{bound}_{a,\,b}(\cdot) \equiv \mathrm{max}(a,\ \mathrm{min}\,(b, \cdot))$$

$$\text{assignment} = \mathrm{bound}_{mindall,\ maxdall}\ (\text{request} \cdot \mathrm{bound}_{allnmin,\ allnmax}\,(\text{stno}) \,/\, alld)$$

where *stno* is the number of stations in the system. Default values for *allnmin, allnmax* and *alld* are 5, 50 and 100 respectively, meaning that the proportionality factor is bounded to a range of 5% to 50% by default.

After each assignment, the datagram request is decreased by the assignment itself and the next request is analysed, if space is still available in the frame. The first assignment that does not fit entirely into the current frame is analysed as the first assignment in the next frame where the rest of the computed amount is assigned.
Any space available in the frame after an entire assignment cycle, i.e. the time between two consecutive allocations to the same station, is shared among all the active stations, even those which had no datagram assignment in that frame.

4.3. DRIFS

This is a distributed control assignment algorithm [6]. The allocation requests are broadcast to all the stations in the system; therefore the delay between a request and the relevant assignment is about one round trip time plus one frame.

No reference burst is transmitted. A control slot is permanently assigned to each active station in a position in the allocation cycle that is fixed, provided that no station enters or leaves the system. The control slot is used to send the requests. Only *stinframe* slots are accommodated in each frame, 8 by default. Therefore the allocation cycle length is given by the number of active stations divided by *stinframe* (rounded up). A fixed quote (*dquote*) of the request is given to each station. If the sum of the assignments is greater

than the length of the allocation cycle, then the assignments are compressed to fit into the assignment cycle space, otherwise the excess space is divided among all the stations. No assignment is less than *mindall* (equal to the burst overhead length *bovh* by default), and *maxdall* is the maximum allocation, equal to half an allocation cycle by default.

4.4. FEEDERS

This too is a distributed control assignment algorithm [6]. A reference burst is transmitted every *na* frames by a master station; *na* is also the length of the allocation cycle. Every station receives an allocation in every frame. The allocations are changed only once every *na* frames. There is no minimum allocation, but a maximum allocation *maxdall*, equal to half a frame length by default. A mechanism similar to the one used by DRIFS is used to compute the allocations, using a fixed quote *dquote* of the request.

5. THE INPUT FILE

The input file consists of a sequence of one or more *run descriptions*. The run descriptions are sequenced and separated by a line containing only the *end* keyword. Putting several run descriptions into a single file is the same as putting them into separate files and writing the file names on the FRACAS command line.

A comment can be put anywhere in the input file. All characters in a line which follow a # sign are ignored. Also, entire lines whose first character is # are ignored, i.e. those lines are not considered as blank lines. This difference is important, because blank lines are significant in the input file syntax.

5.1. GLOBAL VARIABLES

One global variable per row must be specified in the input file.

framesize=	Size of the frame expressed in TRUs.
frametime=	Time length of the frame expressed in TMUs.
rttime=	Round trip time expressed in TMUs.
histlen=	Size of the memory allocated to record the histories, expressed as a number of samples. A sample per frame is generated for each observable statistic different from tru_delay. To be removed in a future version.
[trudhistlen=]	Size of the memory allocated to store the history of tru_delay, expressed in number of samples. A sample is generated for each group of consecutive TRUs sharing the same delay. Default = 2*histlen.
[warmup=]	Number of TMUs to expire before starting to record the data. Default = 0.
[seed=]	Seed for all the random number generations that do not have an explicit seed selection. Zero means that the system time is used as the seed. Default = 0.
[max_cslen=]	Maximum number of categories (bins) to use when the *categorise* computer is called with an unspecified number of bins. Default = 25000.

[ref_traffic=] Used as a reference traffic value by those generators which specify their mean throughput with a factor to be multiplied by this quantity rather than with an absolute number.

5.2. INITER LINE

Only one *initer* can be specified. Since the allocators usually impose a delay of one or two *rttime*, the first few frames have no requests. The initer decides the allocations of these first frames.

5.2.1. Equally distributed datagram allocation initer (*even*)

The stream allocations are equal to the *sreq* value for each station; the VBR allocations are equal to *vminreq* for each station; the remaining available datagram space is evenly divided among all the stations.

syntax: initer even

5.2.2. No datagram allocation initer (*zero*)

Stream and VBR are allocated as in the *even* initer; no allocations are given to the stations for the datagram traffic.

syntax: initer zero

5.3. REQUESTER LINE

Only one *requester* can be specified. It computes the requests for each station. It is called for each station once per frame.

5.3.1. Queue requester (*queue*)

For each type of traffic a request is issued, equal to the current relevant queue length.

syntax: requester queue

5.3.2. FID/VBR (*fodaibea*)

A station makes a request only when it has an allocation (either stream, VBR or datagram). For the stream traffic a request is made equal to the *sreq* parameter of the *streamreq* keyword. For the VBR traffic, a request is made equal to the

vminreq parameter of the *vbrreq* keyword if the input traffic is less than *vminreq*; otherwise, the request is equal to *vmaxreq*. The VBR input traffic estimation is changed at each frame, using the average value of the last *vwin* frames. The datagram request is equal to the datagram queue length plus *dH* times the input traffic, where *dH* is a temporal constant.

The datagram input traffic estimation is changed every *dwin* frames, using the average value on this time interval.

syntax: requester fodaibea dH= dwin= vwin=

where:

dH= temporal constant. Non negative floating point number.

dwin= positive number. Default=1.

vwin= positive number. It may be omitted if no station specifies *vminreq* and *vmaxreq*.

5.3.3. FEEDERS and DRIFS

Same as the *fodaibea* requester. The only difference is that a request is always made, irrespective of whether the station has an allocation or not.

5.4. ALLOCATOR LINE

Only one *allocator* can be specified. It computes the allocations of each station. It is called once per frame, it looks at the requests of all the stations and computes the allocations of all the stations.

5.4.1. Fixed TDMA *(fixed)*

The stream allocations are set equal to the *sreq* value of each station. The VBR allocations are set equal to the VBR request, bounded by the *vminreq* and *vmaxreq* values of the *vbrreq* keyword. The remaining space in the frame is equally divided among the stations and assigned as a datagram allocation, independently of the stations' datagram requests.

syntax: allocator fixed

5.4.2. FID/VBR *(fodaibea)*

The channel space is shared among the stations according to the FID/VBR access scheme. The stream allocations are set equal to the *sreq* value of each station. The VBR allocations are set equal to the VBR request, bounded by the *vminreq* and *vmaxreq* values of the *vbrreq* keyword. The datagram allocations are computed

by using the FID/VBR algorithm. The delay between the request and the corresponding allocation being assigned to the requesting station is an integer number of frames, equal to 3+2 (*rttime* / *frametime*), where the division is an integer excess division.

syntax: **allocator fodaibea bovh= cs= csovh= [csevery=] [fas=] [fas_every=] [mindall=] [maxdall=] [allnmin=] [allnmax=] [alld=] [trace=]**

where:

bovh=	overhead, in TRUs, for burst transmission. Must be less than *framesize*.
cs=	size in TRUs of the control slot. Must be less than *framesize*.
csovh=	overhead, in TRUs, for all the allocations given to a station as redistribution gift or control slot. Must be less than *framesize* and not greater than *cs*.
[fas=]	size in TRUs of the First Access Slot. Must be less than *framesize*. Default = 0.
[fas_every=]	number of frames lasting between two consecutive FASs. Default = 32.
[mindall=]	minimum datagram allocation, in TRUs. Default = <bovh>.
[maxdall=]	maximum datagram allocation, in TRUs. Must be greater than <bovh>. Default is framesize/2.
[allnmin=]	minimum number of stations used for computing the quote of the datagram request that is assigned. Default value = 5.
[allnmax=]	maximum number of stations used for computing the quote of the datagram request that is assigned. Default value = 50.
[alld=]	the quote of the datagram request assigned to each station in the assignment cycle is equal to 100 times the number of stations divided by *alld* (default 100). The minimum and maximum number of stations used for these calculations are *allnmin* and *allnmax*, respectively (defaults 5 and 50).
[trace=]	flag for enabling tracing [y/n]. Default = n.

5.4.3. FEEDERS *(feeders)*

The channel space is shares among the stations according to the FEEDERS access scheme. The stream allocations are equal to the *sreq* value of each station; the VBR allocations are equal to the VBR request, bounded by the *vminreq* and *vmaxreq* values of the *vbrreq* keyword; the datagram allocations are computed by using the FEEDERS algorithm.

The delay between the request and the corresponding allocation is an integer number of frames, equal to 2+*rttime* /*frametime*, where the division is an integer excess division.

syntax: allocator feeders na= rbovh= dquote= [bovh=] [maxdall=] [trace=]

where:

na=	number of frames in which the allocations do not change (assignment cycle length expressed in frames).
rbovh=	reference burst overhead, in TRUs. Is subtracted from the available space once every *na* frames.
dquote=	floating point number. Fraction of the datagram request assigned in the datagram assignment cycle.
[bovh=]	burst transmission overhead, in TRUs. Must be less than *framesize*.
[maxdall=]	maximum datagram allocation, in TRUs. Must be greater than <bovh>. Default is half the available space.
[trace=]	flag for enabling tracing [y/n]. Default = n.

5.4.4. DRIFS *(drifs)*

The channel space is shared among the stations according to the DRIFS access scheme. The stream allocations are equal to the *sreq* value of each station; the VBR allocations are equal to the VBR request, bounded by the *vminreq* and *vmaxreq* values of the *vbrreq* keyword; the datagram allocations are computed using the DRIFS algorithm. The delay between the request and the corresponding allocation is an integer number of frames, equal to 2+*rttime* / *frametime*, where the division is an integer excess division.

syntax: allocator drifs stinframe= stovh= dquote= [bovh=] [maxdall=] [trace=]

where:

stinframe=	maximum number of control slots in a frame.
stovh=	control slot length in TRUs.
dquote=	floating point number. Fraction of the datagram request assigned in the datagram assignment cycle.

[bovh=] burst transmission overhead, in TRUs. Must be less than *framesize*.

[maxdall=] maximum datagram allocation, in TRUs.
Must be greater than <bovh>. Default is half the available space.

[trace=] flag for enabling tracing [y/n]. Default = n.

5.5. STOPPER LINE

This line specifies the name of a routine that is called at each frame. Each time it is invoked, the stopper decides whether or not the simulation must end.

5.5.1. Elapsed time stopper (*stopper*)

Interrupts the simulation after a specified time interval.

syntax: **stopper maxtime time= |frames=**

where:

time= number of TMUs after which the simulation must be stopped.

frames= number of frames after which the simulation must be stopped.

5.6. STATION DESCRIPTION

5.6.1. The station identifier (*station*)

syntax: **station <number>[:<number>]**

where:

<number> station number. If n_1:n_2 is specified, the current station block definition is valid for stations from n_1 up to n_2. The stations must be declared in ascending order, starting from 1. No holes are allowed in the numbering.

5.6.2. The stream allocation request (*streamreq*)

For each station a request for stream traffic allocation can be specified. Allocators and Requesters use this number for their computations (if they need them).

syntax: **streamreq sreq=**

where:

sreq= number of TRUs requested for the stream traffic in each frame.

5.6.3. The VBR allocation request (vbrreq)

For each station two requests for VBR traffic allocation can be specified. Allocators and Requesters use these numbers for their computations (if they need it).

syntax: **vbrreq vminreq= vmaxreq=**

where:

vminreq= is the minimum number of TRUs requested in each frame.

vmaxreq= is the maximum number of TRUs requested in each frame.

5.6.4. The maximum queue length

For each station the maximum lengths (in TRUs) of the stream, VBR and datagram queues can be specified. Zero means infinite queue. When a queue length is different from 0, incoming TRUs are rejected once the respective queue length has exceeded its maximum length. The number of packets discarded is registered in the *s_dropped*, *v_dropped* and *d_dropped* observables, respectively.

syntax: **maxqueuelen [s=] [v=] [d=]**

where:

[s=] maximum stream input queue length. Default = 0.

[v=] maximum VBR input queue length. Default = 0.

[d=] maximum datagram input queue length. Default = 0.

5.6.5. The traffic generators

For each station, several traffic generators can be defined. The available generators are: *constant*, *poisson*, *impulse*, *fgn*, *external*.

Each generator generates one *traffic type*, chosen from datagram (*d*), stream (*s*) or VBR (*v*).

Each non deterministic generator has a *seed* parameter, which can optionally be set to an integer. The seed is used to generate pseudo random numbers. If a seed of 0 is given, the seed for that generator is generated at startup by using the global variable *seed*. In this case, the seed is generally different if the overall structure of the simulation changes, that is, if stations are added or removed, or the generators

changed, or anything else changes. Moreover, if the global *seed* parameter is 0 and the generator *seed* parameter is 0, the seed used for the pseudo random number generation changes at each run, provided that the runs are at least one second apart, because the global seed variable is initialized with the system time. If the same sequence of traffic from a generator in different runs is required, a seed different from 0 must be set. The default for the *seed* parameter is 0.

Each generator generates bursts of TRUs. In the generator descriptions, whenever a TRU is mentioned, a burst of TRU is meant if the *burst* parameter is set to a number greater than 1. The rate of the burst generation remains the same whatever the burst length is. The number of TRUs generated in a frame is simply multiplied by the burst size (an integer number). The default burst length is 1. Each generator has a parameter that indicates the mean total traffic generated (TRUs per TMU). This can be either a *traffic* floating point number indicating the absolute value of the traffic mean, or a *tfactor* floating point number indicating the traffic mean relative to the global parameter *ref_traffic*. In the first case the traffic generated is equal to *traffic *burst* , in the second case it is equal to *tfactor *ref_traffic *burst.*

5.6.5.1. Constant traffic generator ***(constant)***

This generates TRUs with constant interarrival times. The use of the *start*, *stop* and *cycle* optional parameters allows an On-Off generator to be created with a constant duty cycle and fixed interarrival times between TRUs during the On states. *cycle* is the repetition period length. A cycle is composed of an Off-On-off sequence. Within the cycle, *start* and *stop* are the times when the On period begins and ends, respectively. *cycle* set to 0 means no repetition (infinite period length). *Stop* set to 0 means that the “on” period lasts until the end of the cycle. Thus, an infinite step of constant traffic has both *cycle* and *stop* equal to 0, while a single impulse of constant traffic has *cycle* equal to 0 and *stop* different from 0. All the parameters are floating point values.

syntax: **generator <traffic_type> constant traffic=|tfactor= [burst=] [start=] [stop=] [cycle=]**

where:

[start=]	see explanation above. Default = 0.
[stop=]	see explanation above. Default = 0 (which means *"never stops"*)
[cycle=]	see explanation above. Default = 0 (which means *"infinite cycle*").

5.6.5.2. Poisson traffic generator ***(poisson)***

This generates TRUs with exponentially distributed interarrival times. S*tart, stop* and *cycle* optional parameters allow an On-Off generator to be created with a constant duty cycle and a Poisson generation rate during On states. The meanings of the *cycle*, *start*, and *stop* parameters are the same as in the *constant* traffic generator.

Seed is an integer. All the other parameters are floating point values.

syntax: generator <traffic_type> poisson traffic=|tfactor= [seed=] [burst=] [start=] [stop=] [cycle=]

where:

[start=]	see explanation for the *constant* traffic generator. Default = 0.
[stop=]	see explanation for the *constant* traffic generator. Default = 0 (which means *"never stops"*)
[cycle=]	see explanation for the *constant* traffic generator. Default = 0 (which means *"infinite cycle*").

5.6.5.3. Impulsive traffic generator ***(impulse)***

This is a two states-Markov modulated-Poisson generator. Generates TRUs with exponentially distributed interarrival times where the exponential constant depends on the state where the traffic generator is. The probability of switching to the other state is constant, so the time of permanence in each state is exponentially distributed. The resulting traffic is Poisson distributed with two possible transmission rates, chosen at random. Four parameters are needed to define the statistical behaviour of such a generator. One is the average generated traffic, which is specified using the *traffic|tfactor* parameter. The other three parameters are: the mean cycle length *cycle*, which is the sum of the mean times of permanence in the two possible states; the mean duty cycle *duty*, which is the ratio between the mean permanence in state 1 and the mean cycle length; the burstiness *burstiness*, which is the ratio between the mean traffic rate instate 1 and the mean traffic overall traffic rate. The ratio of the traffic rate in state 1 to the traffic rate in state 2 is expressed by: $ratio12 = \frac{burstiness - burstiness \cdot duty}{1 - burstiness \cdot duty}$ and, conversely: $burstiness = \frac{ratio12}{1 - duty + ratio12 \cdot duty}$.

seed is an integer. All the other parameters are floating point values.

syntax: **generator <traffic_type> impulse traffic=|tfactor= cycle= duty= burstiness= [seed=] [burst=]**

where:

cycle=	the mean cycle length (length of state 1 plus length of state 2) in TMUs.
duty=	the ratio between the mean length of state 1 and the mean cycle length.
burstiness=	the ratio between the mean traffic rate of state 1 and the overall mean traffic rate (the one indicated by the *traffic*|*tfactor* parameter).

5.6.5.4. Fractional Gaussian Noise traffic generator (***fgn***)

The number of TRUs generated per frame by this generator approximates a Fractional Gaussian Noise (fgn) process. The generator uses the Random Midpoint Displacement (rmd) algorithm to approximate the *fgn* process. The algorithm generates a cluster of pow(2, *rmdn*) samples at a time. If the length of the simulation is greater than this value, a new cluster of samples is generated, not correlated with the previous ones. The mean of each cluster of samples is forced equal to the traffic requested.

syntax: **generator <traffic_type> fgn traffic=|tfactor= peakedness= H= [rmdn=] [seed=] [burst=]**

where:

peakedness=	the ratio *a* between the variance and the mean of the samples per TMU. The peakedness a_T of the process over an interval of length T is $a_T = a^{2H}$.
H=	the Hurst parameter of the process. Usually in the range between 0.7 and 0.85.
[rmdn=]	binary exponent of the length of a cluster of samples. Default = 18.

5.6.5.5. External traffic generator (***external***)

This generates TRUs at times specified in an external data file. The file is in ASCII format, with an integer number per line. If the *type* is set to *packet,* the numbers in the file are integers which represent the number of TRUs generated in successive frames. The first line contains the number of TRUs generated in the first frame and so on, one line per frame. If the *type* is set to *interval*, the numbers

are floating point numbers representing the distance in TMUs from one TRU generation to the next. The first line contains the generation time of the first TRU, the second the time elapsing from the generation of the first to that of the second and so on, one line per TRU. The times for the *interval* type can also be scaled (usually to do time unit conversions) by multiplying them by a constant indicated with the *scale* parameter. An external process can feed the generator with the numbers, instead of having them stored in a file. In this case the parameter *ipipe* instead of *ifile* must be used. This is the only generator that does not allow a *traffic|tfactor* parameter.

syntax: generator <traffic_type> external ifile=|ipipe= type= [scale=] [burst=]

where:

ifile=	name of the file containing the numbers in ASCII format, one per line.
ipipe=	the command line used to invoke a process which writes on its standard output a sequence of ASCII numbers, one per line.
type=	the string *packet* or the string *interval.*
[scale=]	floating point number that multiplies the numbers read from the file or pipe. Only allowed with the *interval* type. Default = 1.

5.7. COMPUTER LINES

A computer is a routine that operates on observables. Each computer makes a different operation. Computers can be added to FRACAS.

5.7.1. The observables

Observables have names starting with *s_* or *v_* or *d_*. These prefixes stand for stream, VBR and datagram respectively, meaning that the statistics collected only relate to the respective type of traffic. Here is a list of the available observables and a brief discussion for each of them.

s_input, v_input, d_input

Number of TRUs presented to the input of a station per frame. The TRUs are generated by the generators declared in the station blocks. These numbers are the sum of those generated by the station's generators at each frame.

For the pseudo-station *sum* the observable is the sum of the inputs to all the stations.

s_queue, v_queue, d_queue

Length of the input queue of a station. Each station receives the TRUs generated by its generators and adds them to its input queues, one for each type of traffic. The lengths of these queues are registered in the observables above; the queues are then reduced by the number of TRUs to be sent in the current frame.

For the pseudo-station *sum* the observable is the sum of the queues of all the stations.

s_sent, v_sent, d_sent

Number of TRUs transmitted in a frame by a station. If the length of the queue of a specific type of traffic is greater than the relevant allocation received, this number is set equal to the allocation for that frame, otherwise it is equal to the queue length for that frame.

For the pseudo-station *sum* the observable is the sum of the TRUs sent by all the stations.

s_request, v_request, d_request

Request of a station in a frame for each type of traffic. These are the numbers computed by the requester at each frame.

For the pseudo-station *sum* the observable is the sum of the requests made by all the stations.

s_allocation, v_allocation, d_allocation

Number of TRUs allocated to a station in each frame for each type of traffic. These are the numbers computed by the allocator.

For the pseudo-station *sum* the observable is the sum of the allocations given to all the stations.

v_extraspace, d_extraspace

There is a hierarchy among the types of traffic. At each frame, the space not used by stream traffic is given as extra space to the VBR traffic, and the space not used by VBR traffic is given as extra space to datagram traffic. Here is how the allocation sequence for each station works.

1. The stream TRUs are sent. If *s_queue* is less than *s_allocation*, *v_extraspace* for this frame is set to the difference, and *s_sent* is set equal to *s_queue*. Otherwise *s_sent* is set equal to *s_allocation*.

2. The VBR TRUs are sent. If *v_queue* is less than *v_allocation* plus *v_extraspace*, *d_extraspace* for this frame is set to the difference, and *v_sent* is set equal to *v_queue.* Otherwise *v_sent* is set equal to *v_allocation* plus *v_extraspace*.
3. The datagram TRUs are sent. If *d_queue* is less than *d_allocation* plus *d_extraspace*, *d_sent* is set equal to *d_queue*. Otherwise *d_sent* is set equal to *d_allocation* plus *d_extraspace.*

For the pseudo-station *sum* the observable is the sum of the extra spaces of all the stations.

d_unused

The space (in TRUs) unused in a frame by a station. It is equal to *d_allocation* plus *d_extraspace* minus *d_sent.*

For the pseudo-station *sum* the observable is the sum of all the TRUs not used in the system.

s_delay, v_delay, d_delay

The frame delay in TMUs for each type of traffic. Each sample is the mean of the delays of the TRUs generated in a frame by a station's generators. The delay is equal to the round trip time plus the queueing time, which is an integer number multiplied by the frametime.

For the pseudo-station *sum* the observable is the mean delay of the TRUs generated in a frame by all the stations.

s_trudelay, v_trudelay, d_trudelay

The TRU delay in TMUs for each type of traffic. This observable provides a sample per TRU, while the frame delay observable provides one sample per frame. The delay is equal to the round trip time plus the queuing time, which is an integer number multiplied by the frametime.

For the pseudo-station *sum* the observable is the delay of each packet generated in the system. The ordering does not make much sense: it is the listing of the delay of packets generated by each station in turn, so making a graph or correlation of it would yield nonsense.

5.7.2. The computers

A computer is a routine that performs calculations on some station's observables; usually some kind of statistics are computed. A computer line can specify more

than one station. The <*stations*> argument can be either a number (the station number), or two numbers separated by columns (the first and the last station numbers). In this case, the computer is invoked repeatedly on each of the stations specified. As a special case, the pseudo-station *sum* can be specified. Each observable has a special meaning for this pseudo-station, which is described in the section about observables. One more special case, *all*, exists: the computer is run on all the stations, including the *sum* pseudo-station. The pseudo-station *sum* is indicated as station number 0 in the output listings.

All computers produce their output on the standard output by default. Each computer can also be instructed to send its output to a file or to an external program. One of three arguments can optionally be added to the argument list of each computer:

[ofile=]	send the output to a file specified as an argument. The previous contents of the file are overwritten.
[afile]	append the output to a file specified as an argument.
[opipe]	pipe the output to a program whose command line is specified as an argument.

For each of the three possibilities, a string must be supplied after the equal sign. The string may be enclosed between double quotes if it contains other than alphanumeric characters plus the period and the minus sign. A double quote within the string can be quoted by preceding it with a backslash.

The output produced on the standard output is generally more verbose than the output directed to files or external programs. Details are provided in each computer's description. It is also possible to obtain a terse printing on standard output by invoking FRACAS with the *-t* (terse) option. In this case, the output sent to *stdout* is equal to the output sent to files or pipes.

5.7.2.1. Simple statistics computer ***(simplestats)***

This computer prints on standard output the minimum, maximum, number of samples, mean and standard deviation of the specified observable. These statistics are printed on a single line, preceded by a header line and followed by a blank line. The header and the blank line are not printed with the *-t* (terse) option.

syntax: computer <stations> <observable> simplestats [ofile=|afile=|opipe=]

5.7.2.2. Quantile computer ***(quantile)***

This computer prints the quantile of an observable. The 0.9 quantile of an observable is the sample such that 90% of samples are less or equal than it. Each

output line contains the requested quantile and the result, preceded by a header line and followed by a blank line. The header and the blank line are not printed with the *-t* (terse) option.

syntax: computer <stations> <observable> quantile [ofile=|afile=|opipe=] q=

where:

q= the fraction of the number of samples less or equal to the value printed by the quantile computer. It is a floating point number in the [0; 1] interval.

5.7.2.3. Sample listing computer ***(listing)***

This computer lists the values of all the samples observed. One parameter, *every*, and three styles are available.

The *normal* style prints one value per line, taking one sample every *every* samples.

The *aggregate* style prints one value per line, which is the sum of *every* successive non-overlapping samples.

The *compact* style prints two values per line. It groups successive equal samples in one line: the first number printed is the number of equal successive samples, the second number is their value. The parameter *every* makes no sense with this style.

The listing of the values is preceded by a header line and followed by a blank line. The header and the blank line are not printed with the *-t* (terse) option.

syntax: computer <stations> <observable> listing [ofile=|afile=|opipe=] [type=] [every=]

where:

[type=] a string whose value is *normal*, *aggregate* or *compact*. See above for explanation. Default = *normal*.

[every=] a positive integer. See above for explanation. Default=1.

5.7.2.4. Distribution computer ***(categorise)***

This computer divides the samples of the observables into “categories”, or “bins”. The number of categories is a parameter. All the categories have the same size and span the entire range of the observable, from its minimum to its maximum value. When zero is specified as the number *catn* of categories, the categories are built with a width equal to 1, centred around all integer values plus 0.5, covering the entire range of the observable. This feature is useful when the observable is known to take on integer values. The maximum number of bins computed when *catn* is 0

is *max_cslen*, a global variable to be specified at the beginning of the input file. If the range of the observable is greater than this parameter's value, the bins are enlarged to integer widths, using the smaller width such that the number of categories is less than *max_cslen*.

A list of lines is produced, with two values in each line: the centre value of the category and the number of samples in it. The sum of all the samples in the different categories is equal to the number of samples of the observable.

The listing is preceded by a header line and followed by a blank line. The header and the blank line are not printed with the *-t* (terse) option.

syntax: computer <stations> <observable> categorise [ofile=|afile=|opipe=] catn=

where:

catn= number of categories in which the range of observable values is divided. 0 makes categories of width 1, centred on integer numbers plus 0.5. Default is 0.

5.7.2.5. Computer for graphing the distribution on a tty ***(ttydistrib)***

This computer creates a graph—printable on a tty or a character printer—which represents the mass density function of the observable. The graph is preceded by a header line and followed by a blank line. The header and the blank lines are not printed with the *-t* (terse) option.

syntax: computer <stations> <observable> ttydistrib [ofile=|afile=|opipe=] [rows=] [cols=]

where:

[rows=] number of rows used for representing the character graph. Default = 22.

[cols=] number of columns used for representing the character graph. Default = 78.

5.7.2.6. Computer for graphing the observable versus time on a tty ***(ttygraph)***

This computer creates a graph—printable on a tty or a character printer—which represents the observable versus the time. The graph is preceded by a header line and followed by a blank line. The header and the blank lines are not printed with the *-t* (terse) option.

syntax: computer <stations> <observable> ttygraph [ofile=|afile=|opipe=] [rows=] [cols=]

where:

[rows=]	number of rows used for representing the character graph. Default = 22.
[cols=]	number of columns used for representing the character graph. Default = 78.

5.8. INPUT FILE SYNTAX

A FRACAS input file is composed of a sequence of one or more run descriptions.

file contents:= <run description>
{ <blank line>
end
<blank line>
<run description>}

run description:= <global variable section>
<initer section>
<requester section>
<allocator section>
<stopper section>
<station section>
{<station section>}
<computer section>

global variable section:= <global variable line>
{<global variable line>}
<blank line>

global variable line:= <global variable name>=<global variable argument>

initer section:= **initer** <initer name> <initer arguments>
<blank line>

requester section:= **requester** <requester name> <requester arguments>
<blank line>

allocator section:= **allocator** <allocator name> <allocator arguments>
<blank line>

stopper section:= **stopper** <stopper name> <stopper arguments>
<blank line>

station section:= **station** <number>[**:**<number>]
[**streamreq sreq=**<argument>]
[**vbrreq vminreq=**<argument> **vmaxreq=**<argument>]
[**maxqueuelen** [**s=**<argument>] [**v=**<argument>] [**d=**<argument>]]
<generator line>
{<generator line>}
<blank line>

generator line:= **generator s**|**v**|**d** <generator name> <generator arguments>

computer section:= <computer line>
{<computer line>}

computer line:= **computer** <ids> <computer name> <observable> <computer arguments>

ids:= <number>[**:**<number>] | **sum** | **all**

REFERENCES

1. N. Celandroni, E. Ferro, N. James, F. Potortì
"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.

2. N. Celandroni, E. Ferro, F. Potortì
"FODA/IBEA-TDMA. System Description. Final Report", CNUCE Report C94-18, September 1994.

3. N. Celandroni, E. Ferro, F. Potortì
"The performance of a TDMA satellite system for non real-time and real time traffic", CNUCE Report C95-12, February 1995.

4. N. Celandroni, E. Ferro, F. Potortì
"Satellite bandwidth allocation schemes for VBR applications", CNUCE Report C94-24, December 1994.

5. N. Celandroni, E. Ferro, F. Potortì, M. Conti, E. Gregori:
"A bandwidth assignment algorithm on a satellite channel for VBR traffic", submitted to the International Journal, February 1995.

6. N. Celandroni, E. Ferro, F. Potortì
"Study of distributed algorithms for satellite capacity assignment in a mixed traffic and faded environment", CNUCE Report C-95-28, September 1995.

APPENDIX

An example: input file

```
## point 1/6
##
ref_traffic 0.6
#
framesize 615          # traffic units
frametime 20           # frame length is 20ms
rttime 252             # round trip time is 252ms
histlen 300000         # length of histories
warmup 12000           # wait 12s before registering
seed 0

initer even

requester feeders-drifs dH=400 dwin=5 vwin=25

allocator drifs dquote=0.05 bovh=8 stinframe=8 stovh=16

stopper maxtime time=6000000

station 1
generator d impulse cycle=3000 duty=0.15 burstiness=5 burst=4 tfactor=0.42

station 2
generator d impulse cycle=3000 duty=0.15 burstiness=5 burst=4 tfactor=0.31

station 3
generator d impulse cycle=3000 duty=0.15 burstiness=5 burst=4 tfactor=0.21
```

```
station 4
generator d impulse cycle=3000 duty=0.15 burstiness=5 burst=4 tfactor=0.06

compute sum simplestats d_input
compute sum simplestats d_trudelay
```

An example: FRACAS output

```
Number of `d_input' samples for station #0 is 299400.  Statistics:
samples     minimum     maximum     average     var   stddev
299400      0     328   48.04 2224  47.16

Number of `d_trudelay' samples for station #0 is 14381736.  Statistics:
samples     minimum     maximum     average     var   stddev
14381736    272   312   272.1 1.29  1.136
```