7

Performance analyzer for Intelligent Network

Jorma Jormakka
Communications Laboratory
Helsinki University of Technology
Otakaari 5, 02150 Espoo, Finland
E-mail: Jorma.Jormakka@hut.fi
tel. 358-0-4512360, fax. 358-0-4512345

Abstract

This paper describes a prototype of a tool which can be used in dimensioning Intelligent Networks (IN). A EURESCOM pre-study of currently used dimensioning methods for IN showed that conventional methods and tools for dimensioning telephone networks do not as such suit to IN and that the commercially available simulation tools and environments are not easy to use as dimensioning tools. EURESCOM P308 project and it's continuation project aims to developing methods and later prototyping a suitable tool. The prototype tool described here is not the tool to be developed by the EURESCOM project though the author has been working on the project E-P308. This tool can be used as input to the EURESCOM prototype which is expected to address a larger part of dimensioning problems. The tool described here is only limited to evaluating some Grade of Service parameters and to analyzing the effects of load control on the net.

MAIN PARTS OF THE TOOL

The tool consists of two editors by which the network structure and IN protocols are modeled, a simulator, an approximate solution generator and a plotter.

Network editor

Figure 1 shows the network editor. It is a graphical editor and will be used for studying televoting in a network of one Finnish telephone operator. There is a strong interest for this service for instance audience voting in television programs and the basic question with this service for an operator is what amount of televoting traffic can be allowed. Televoting as a

J. Harju et al. (eds.), *Intelligent Networks*

source of revenue for an operator may not be very important but this service can disturb the network and the grade of service for this IN service should be on acceptable level, that is, the level expected from this service. The network is modeled by the editor and the model is run with a simulator which tracks the amount of generated traffic, blocking probabilities, interarrival time distributions, service time distributions and other relevant features. Currently the televoting is not using SCP and the implemented version of this network is basically a circuit-switched net but the model will be upgraded when the underlying network structure is changed. Interesting problems here are a possible unfairness of televoting blocking probability and the effects of different congestion control solutions. The telephone images in the editor are traffic sources where the rate, distribution, time-dependancy of the distribution and user impatience statistics are given. Other elements are channels and switches which are expected to model the existing equipment in sufficient detail from traffic point of view.

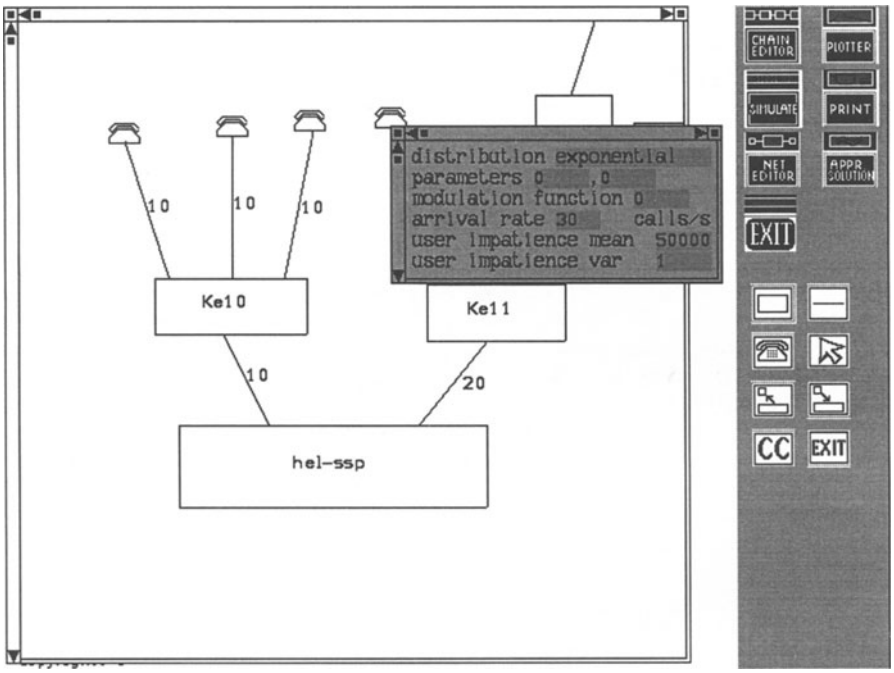

Figure 1.

Work-load-chart editor

Another editor is used to describe IN protocols. The editor is seen in figure 2. In this editor an IN service protocol is described as a work load chart by which here means a message sequence chart where each message is assigned a work load. Each work load are given the mean and the distribution. In a communication channel the work load is the transfer time and in a network node the work load is the processing time. Channels and nodes are both described as elements

in the editor and the editor has for each element a set of parameters including number of servers, waiting room size, processor sharing mode in a server, selection rule for a free server and load control filter method. The main usage of this editor is to model signaling messages for a given IN service and then use either the simulator or another part of the tool, the approximate solution which will be described later, to obtain some grade of service parameters such as connection establishment and termination delays and rejection probability for the signaling part. This description of the network by work loads is phenomenological: the protocol is known and will be standard but the actual equipment can be very different and of it's performance little may be known. This solution avoids the problems of modeling the actual equipment and models only the protocol on some layer, say TCAP-layer. The equipment appears then only in the work load of the messages and in the few selected parameters for the elements.

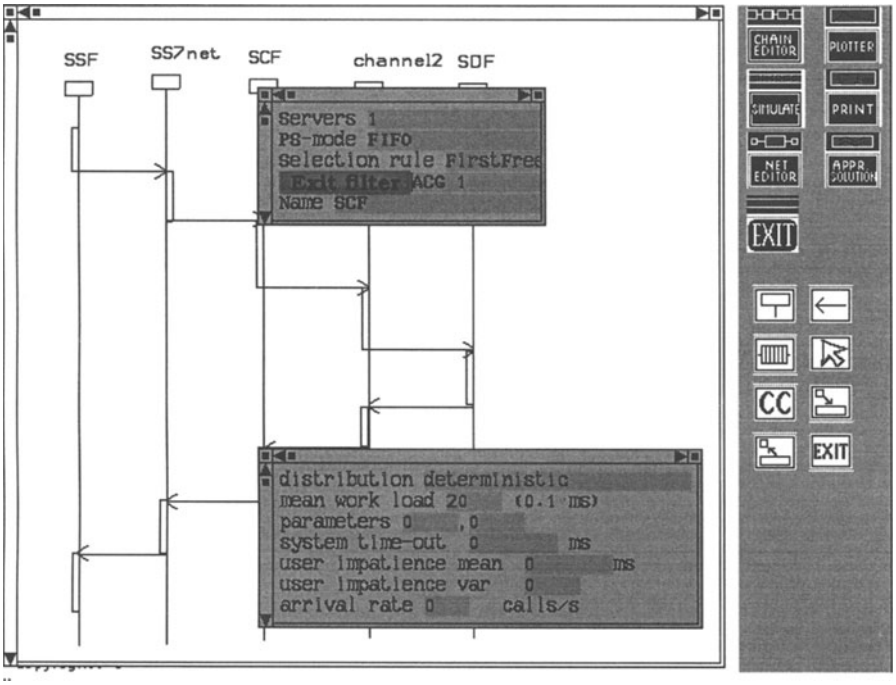

Figure 2.

Simulator

The simulator is seen in figure 3. It is not a discrete event simulator but of type which occasionally is called a scan-next-event simulator. The simulator keeps in each task the amount of received work and remaining work for each job and selects from all active jobs next time quanta which will be processing up to the next event which is the time when the number of

active jobs changes. In a discrete event simulator (DES), which is the solution used by most simulators, the ending time of each job is called an event and all these events are put in time order to an event list which a scheduler is reading. The difference between these simulator types is that the amount of received service for each job is not available in a DES-simulator which makes difficult for instance implementation of such processor sharing algorithms where the selection of which job gets next time quanta depends on the amount of received or remaining service. Simulation of different processor sharing algorithms was seen essential to IN since SDF is likely to be a database and this was the main reason that the simulator was implemented as a scan-next-event simulator. A DES simulator is in general faster but the implemented simulator is sufficiently fast for the intended applications. Mostly the simulator is needed for determining grade of service parameters and for selecting a good load control algorithms. These applications involve only a relatively small network and the simulator performance is not so important as the ability to correctly model different features.

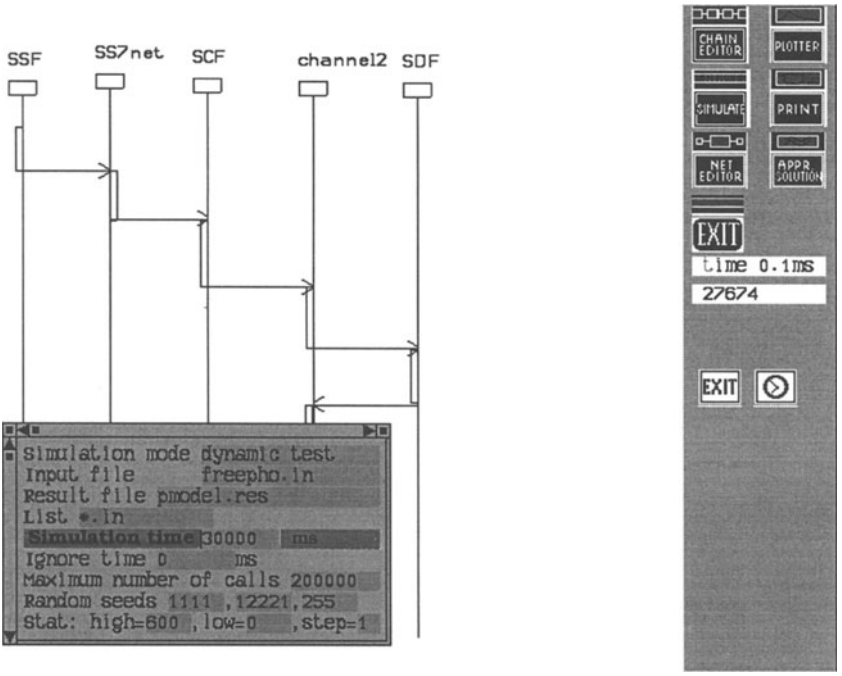

Figure 3.

Approximate solution finder

This dimensioning tool is not basically a simulator though a simulator is a necessary part for investigating the real behavior of the simulation model. Simulation can, however, be slow and it gives results as curves and not as expressions from which follows that generality of the

results is often not clear. The approximate solution finder has another approach: Also here the editors are needed to model the network and protocols. But the models are not simulated, instead using the information in the models it is in many cases possible to solve some traffic related parameters using other mathematical and algorithmic methods than direct simulation. Applying the methods described in EURESCOM P308 project [1] this part of the tool creates a very simple model for the network as seen from SSP: it approximates the whole network as a one-server queue and a load dependent delay matching the maximum throughput to the queue and matching end-to-end delay to the load dependent additional delay for two different loads. This model may seem a bit too simple but if we ignore call rejection, then the whole network could be modeled as one queue with some unknown service time distribution, so our model is actually even unnecessarily complicated.

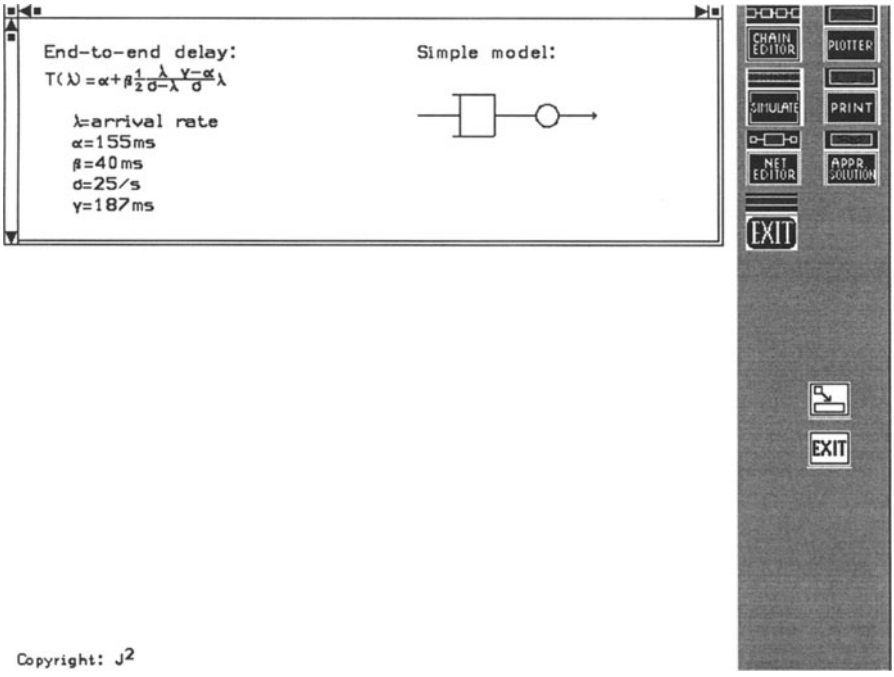

Figure 4.

The idea behind this model is that since, under mild conditions, all queues create in high load exponential delays, we separate the exponential part to a 1-server queue with similar high load behavior and the remaining part turns out to grow almost in a linear manner in many cases. The real growth of the remaining part is much more complicated but never the less, this simple model is useful as a rough model for calculating end-to-end delays. The tool uses an algorithm [2] for determining the queue service time and the additional delay parameters. In this model most of the more complicated features of the network are omitted but [2] describes some ways to take them into account.

The proposed way to include load control into this simple model is to have a finite waiting room for the queue and to modify the arrival process from a typical Poisson process, which may be a good approximation for some IN services such as FreePhone, to a process which is leaving a call gapping algorithm when Poisson process is entering the algorithm. It is not difficult to calculate the filtered process for some call gapping algorithms, see [2] for formulas and [1] for derivations. Knowing that real life SSPs and SCPs use several methods for load control one may wonder if the suggested ways are sufficient. It is likely that transfer from the load control parameters in this model to the actual load control parameters will not be one-to-one.

One way to include time-outs in steady state is to modify the service time in the one-server queue from the balance equations as in [2]. In [2] are suggested ways to include other features into the model such as rejection caused by automatic call gapping, however work of EURESCOM P308 on these formulas is not finished and so the actual methods may be quite different.

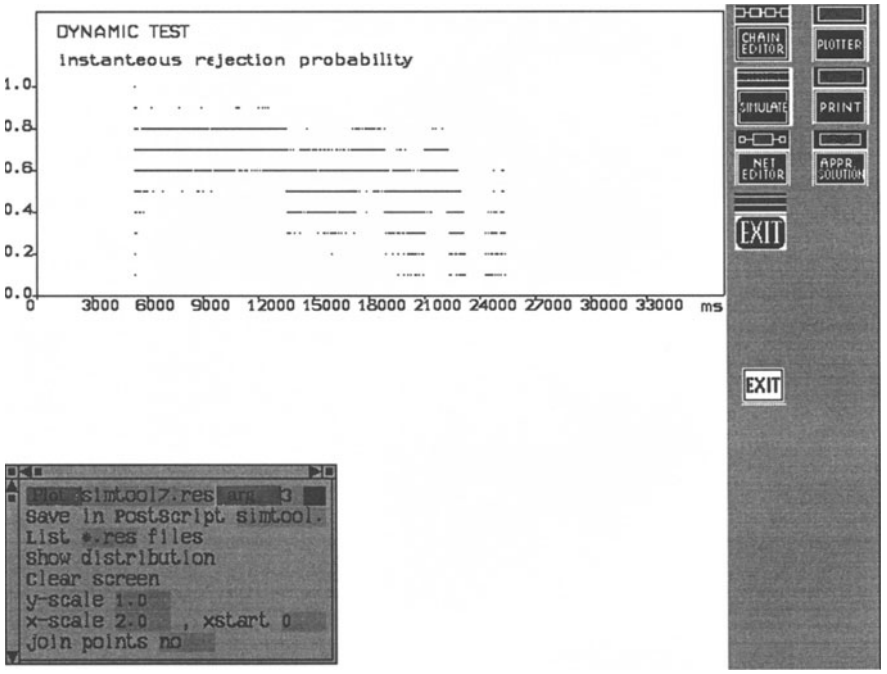

Figure 5.

The principle of the approximate solution finder is seen in figure 4. Whatever the actual methods for finding formulas for Grade of Service (GOS) parameters will be from P308 project, the tool calculates what it can calculate from the models defined with the editors and produces formulas for some GOS-parameters, such as connection establishment delay and rejection probability. These formulas can be used in another dimensioning tool which optimizes

the network capacities. It is necessary to stress here that this presented tool is not the only dimensioning tool that would be needed, this tool helps to find some GOS-parameters, study the effects of load control algorithms and maybe something else. There are other needs, such as calculation of traffic matrices, optimization, planning which this tool does not at all address.

Plotter

Plotter (figure 5) is used for viewing the results. It can create a PostScript file for printing. A feature which is lacking from many commercial simulators is plotting a given curve against the results. In this plotter this task is easy.

The tool contains also a print-button by which the figures in this paper are created. It dumps the screen as a compressed color image to a file from which the image can be read to a text processor.

EXAMPLE OF DIMENSIONING PROBLEM, LOAD CONTROL ALGORITHM

As an application of the simulator in IN dimensioning figures 6-13. There the problem is to determine a good way to control overload. In IN load control is made in several ways and different call gapping algorithms have a central place. In [3] an adaptive load control algorithm for call gapping in circuit-switched network is presented. It uses occupancy of a node and tries to keep the occupancy at a target level, say 0.85. This way of dynamic control is also suggested in P308 [1] for the circuit-switched part.

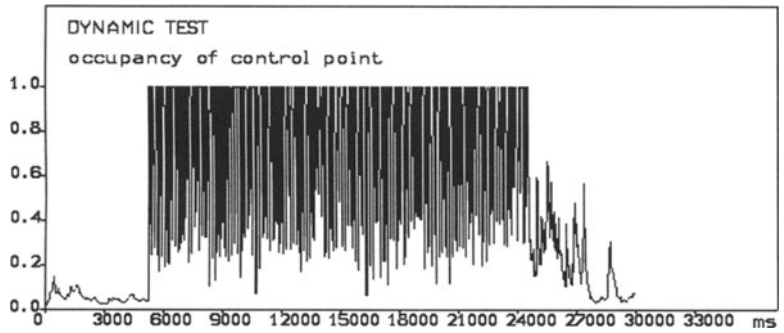

Figure 6. Occupancy using the algorithm in [3] for the simulated signalling network. The control does not manage to keep the occupancy in a given value.

For the signaling network this performance measure is worse than some other performance measures such as queue length in the control point or relative wasted time defined as received service/total waiting time for a message for two reasons: The occupancy is well-defined only in a steady state, under changing traffic conditions occupancy must be averaged and then arises a question how to calculate effectively this measure, in the simulations occupancy was averaged using a circular buffer over 100 last changes weighted by the length of the periods between

changes, such calculation is a relatively heavy operation. Another problem with occupancy is that it is sensitive to overload only if the target value is much below 1. Since the signaling network is basically a waiting system, maximum performance as measured by number of serviced calls as suggested in [3] is obtained by having all servers busy if there is work to be done. Even if additional work to be done by the network nodes is taken into account, it is probably better to have some small queue in SDF and SDF occupancy close to 1 in case overload control is used. Then a suitable measure of load is queue length or relative wasted time which is closely related to the queue length. A suitable measure of goodness is then not the number of calls serviced because this measure will be, in all cases, almost the same as the number of calls serviced without any overload control, since it is limited by the service speed if servers are mostly busy. Better measure of goodness is the average waiting time or as in the following simulations, the queue length or the relative wasted time. All of these measures of goodness are related and give similar results. Control of the waiting times can be made by controlling either queue length or relative wasted time.

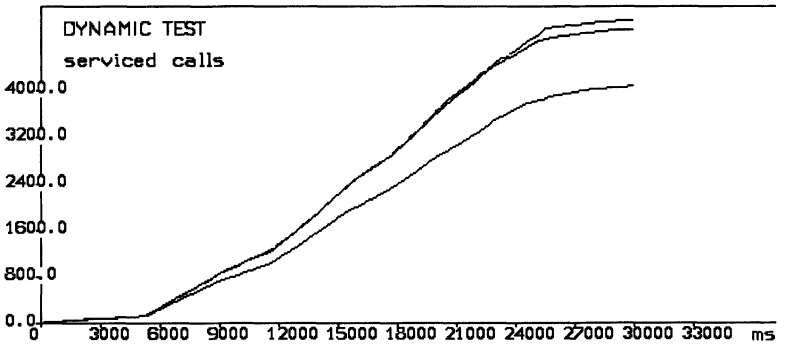

Figure 7. Number of serviced calls is smaller for the algorithm using occupancy (lowest curve) than for SQLA (simple queue length alg.) for the obvious reason that more calls are being served if the servers are all the time busy. The highest curve is SQLA and the middle curve uses constant gap size set to nominal maximum allowed gap size, then occasionally servers are not busy when the offered rate is low resulting to a bit lower performance than SQLA.

The tool contains a dynamic test where for 5 s (seconds) comes Poisson distributed traffic with a rate that can be serviced (in the example 25 calls/s), then from 5 s to 10 s the rate is raised to a high level (1000 calls/s) and then lowered again to the acceptable level (25 calls/s). The call gapping algorithm in all simulations was the basic call gapping algorithm where time is divided into time slots (gaps) of fixed length and the first call in each gap was accepted. Other variants of call gapping are also implemented in the tool.

A simple queue length control algorithm (SQLA) where two gap lengths were used changing them when the queue length in the control point (SDF) reached given upper and lewer bound levels results to some level of oscillation. The first oscillation is higher than the upper level (figure 8). As analyzed in [1] the first high oscillation is caused by the calls which have already arrived to the system before the overload is noticed in the control point. These calls create a queue which will not have time to resolve. Similar phenomenon occurs also using other measures of load. A simple improvement is to first stop all traffic for a short time by

putting gap length to a high value and then lower the gap to the maximum level that can be serviced. This rejecting all arrivals for a short time allows the queue to resolve and waiting times are reduced (figure 11).

The simple queue length control algorithm has another problem when the offered call rate goes fast down, the algorithm changes the gap size many times (figure 12). In the simulations there comes another peak in queue length, this is caused by retrials since in the simulations rejected calls are retried 2 times with retrial time normally distributed with mean 5 s and variance 1. The control algorithm behaves rather poorly in the falling call rate part. One easy improvement is to change the gap length to a smaller value gradually. In figure 13 the gap length is decreased with a linear function of queue size instead of putting it abruptly to a small value as in SQLA. The second peak in queue length is reduced and in general the behaviour is better.

These simulations as such do not strongly recommend any dynamic load control algorithm since the model of the actual net has major importance. In these simulations good performance was obtained by an algorithm which, when gap should be increased puts it first to a high value for a short number of arrivals to the control point and then sets it to the maximum level that can be serviced. In the opposite way, decreasing rejection probability by decreasing gap length is made gradually.

It is also possible to simulate with the tool several sources, i.e., several SSPs using same SCF and SDF. It is just these kind of situations where dynamic load control is needed, if only one SSP is generating traffic to SCP, then call gapping could be all the time on with cap size put to the maximum level of traffic which can be serviced. With several SSPs, one way is to use two levels: a garanteed arrival rate for SSP and a maximum allowed arrival rate from one SSP. Dynamic control of gap size is used to change the traffic coming from SSPs between these levels depending on the activity of other SSPs. SCP must keep track of the SSP where call originates. One computionally effective load measure is queue length obtained as a difference of calls entering the control point and calls leaving the control point, such counter is kept for each SSP controlled by the SCP.

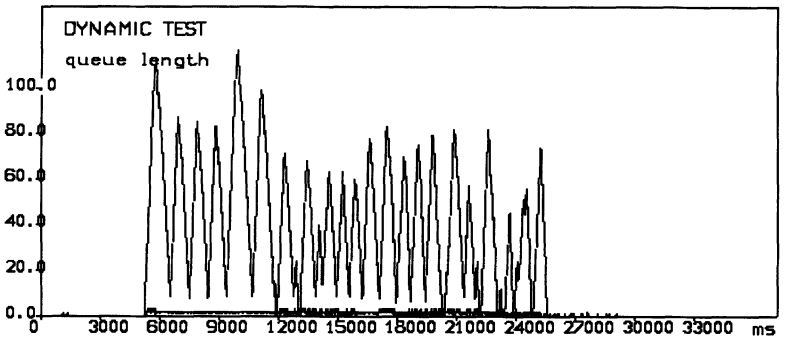

Figure. 8 SQLA can result to oscillating queue length (upper curve) but the oscillating can be made smaller by selecting the upper and lower value more suitably. Notice that SQLA results to one high oscillation when the rate changes (5 s and 10 s). The lower curve is queue length when a constant gap of maximum allowed rate is used.

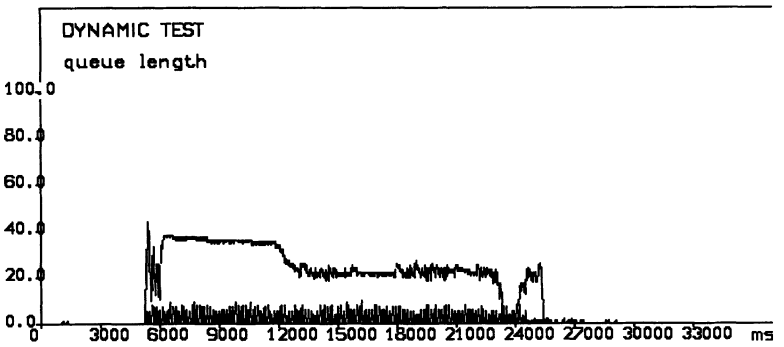

Figure 9. Queue length in the control point using wasted time (upper curve) and occupancy (lower curve).

The relation (received service time/total waiting time) in the controlling element is here called wasted time, as a load measure it is similar to the queue length and an algorithm similar to SQLA can be made using this measure with upper and lower bounds for load and two values for gap size. This algorithm gives similar results to those of SQLA since the load measures are closely dependent. The upper curves in figures 9 and 10 show respectively the queue length and the serviced calls if wasted time is measure and bounds 0.1 and 0.5. In the lower curves the algorithm is as in [3] (in [3] it the algorithm is not intended for signalling network).

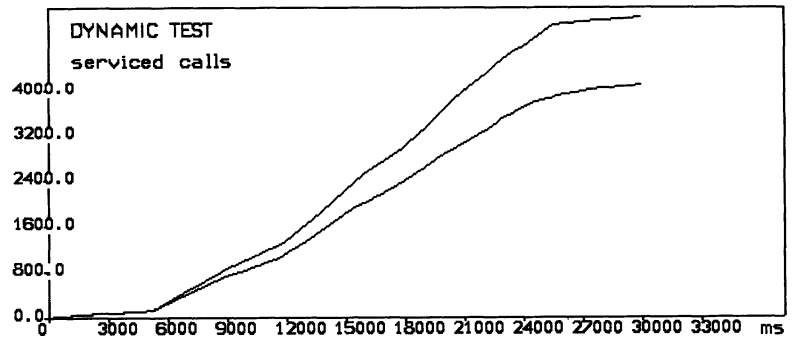

Figure 10. Number pf serviced calls using as load measure wasted time (upper curve) or occupancy (lower curve).

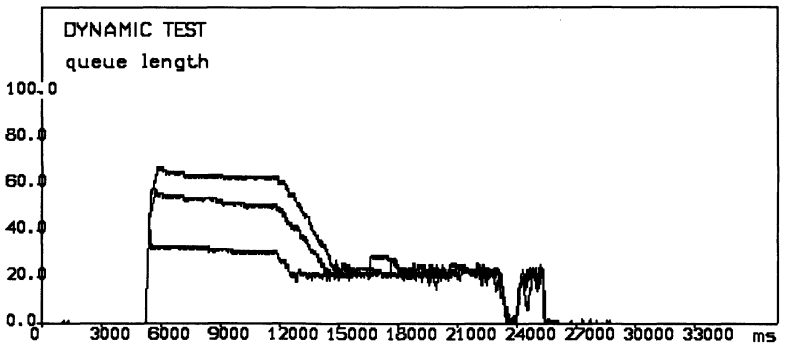

Figure 11. By stopping arrivals (putting gap size to a high value) for a small time, the waiting time of a call can be reduced. The waiting time depends on the queue length. In the lower curves the arrivals are stopped, highest is only SQLA.

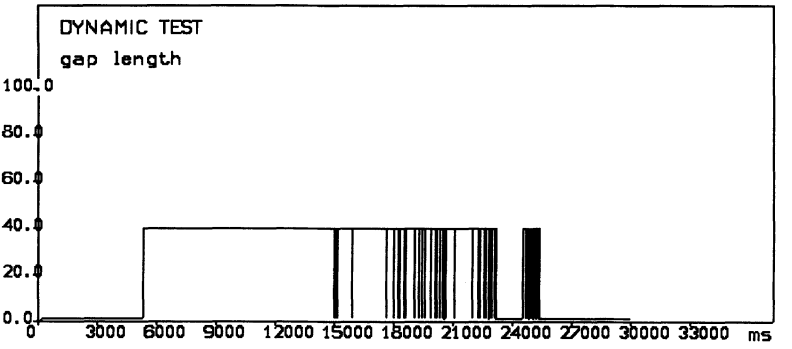

Figure 12. When the arrival rate goes down, SQLA has difficulties with stabilizing, in the figure the gap size changes many times. This results also to longer waiting times for calls than what is necessary.

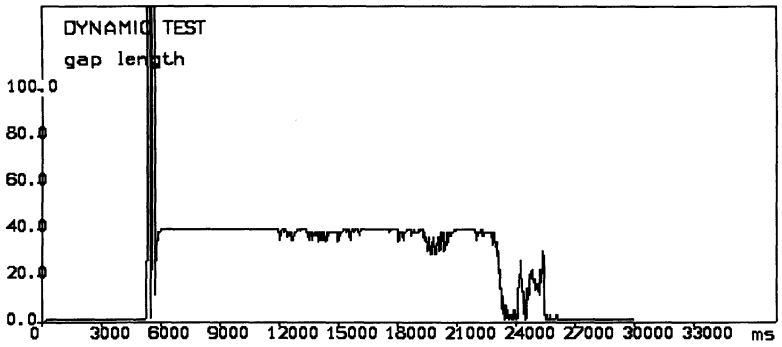

Figure 13. By reducing the gap size gradually, as in the algorithm above, the improved behaviour gives shorter queue lengths and consequently shorter waiting times.

ADDING NEW FEATURES TO THE TOOL

A user would probably sooner or later want to modify and add new features to the dimensioning tool. It is probably not possible to create a tool which has all of the necessary features implemented and parametrized so that a user can only change parameters. Reasons why this is not likely is that IN field is changing rapidly, there are non-standard solutions in existing networks and different solutions will always exist for achieving good performance, so a tool cannot model performance of any IN network using a small number of options. Many commercial simulation tools have solved this problem by creating a special simulation language by which any system can be described, the tool itself is often licensed software product of which sources are not given. In this tool a user makes changes by writing new C-code and all source code of this tool is available and can be modified at will. The advantage of this is that there should not be unclarity of what the tool does. The penalty is that changing somebody else's code is not one of the pleasures of the world. There are currently two major parts in this tool which should be understood by a modifier, the simulator and the graphical interface.

The simulator is a short piece of code, in each element queues as linked lists are kept for each server and a common queue to all servers. A scheduler is selecting next time quanta by asking from each element the time to the next instance of finishing service and selecting the minimum of these proposed times for each element and the time to next arriving call. All servers are advanced by this time quanta. The principle in the simulator is that a call is a chain of messages and the messages are queued in the server queues, this is not quite the same as an ordinary queueing network. When a call is generated, it is assigned a time-out which is the minimum of system time-out and user impatience time. Time-outs are checked between every time quanta from a special time-out list which is made effective by special pointers. In general, understanding the simulator code should be easy.

The tool makes use of a user interface development tool GRAPH created by the author. GRAPH is a part of a medium size software kit (about 40000 lines of C-code) which contains in addition to GRAPH a database, a special C-language compiler and interface to transport service and interface routines to some operating system calls. This set of tools forms a

collection of local interfaces so that protocol software written in sligthly restricted C can be ported to another system implementing these interfaces (user interface, database interface, transport interface, system call interface) and also moved through net and compiled with the special C-language compiler. This enables fast creation of high level services such as multi-player games or other graphical programs which should work in the net but will not be standardized as application layer protocols, a feature which might be interesting in IN service creation once graphical services appear in large scale. Other parts of the kit than GRAPH will not be described here.

There exists several good tools to develop user interface such as MS-Windows, X Window System, Visual Basic so it may feel unnecessary to have a new tool for that. GRAPH is probably easier to use than most user interface development tools, especially for creating graphical editors. However, the simplicity is largely a result of lacking many advanced options. GRAPH has ready the following features:

- press buttons, icons, cards, basic graphical routines
- menu, text and vector graphics window with operations: resize, move, change to icon, scroll vertical and horizontal, pop, pull, rotate)
- vector graphics, that is movable and resizable objects made out of line segments and hot points where can be menus for implementation of a graphical editors
- fonts, fixed and proportionally spaced, created with a font editor
- two-dimensional editor for making bit map pictures, for instance for icons
- three-dimensional editor for making pictures which can be rotated in three dimensions
- animation editor for making animation objects
- creation of PostScript and compressed images (*.pcx)

GRAPH is object based, an objects has three functions: makeobject, object, freeobject. A display manager is managing the graphical objects. GRAPH is written for 286-486 DOS PCs but porting GRAPH is relatively easy since the dependency of operating system is only in extended memory handling, display of files in a directory and in mouse handling. Dependency of graphics display and processor is only in basic drawing routines (read and display a rectangle, bit, byte in a screen). In total, the non-portable code in GRAPH is a very small and isolated part of the code most of which is portable C-language code.

Main advantage in implementing the dimensioning tool prototype with GRAPH is that this interface tool contains several ready window types which in many other interface tools would be made with lower level coding. The connection to the simulator is also implemented and easily modified as is the plotter for viewing results from ASCII-tables produced by the simulator.

Using GRAPH for creating a new editor to the dimensioning tool would involve creating a new press button to the main menu. The press button image is made with the 2-dimensional editor. A editor window is made by creating new figures to the existing vector-graphics-window object in GRAPH. A sub-menu for the new editor is made with press buttons, again drawing the images with 2-dimensional editor. The menus for each object in the new editor are made using the menu-object of GRAPH. The necessary coding work for menus is to write fill-routines for the menus giving fields, the menu texts and default values. The actions of each menu choice are also written to a special file where actions of all menus are listed. The information from a vector-graphics-object will come to certain structures from which the data

is read to the simulator or what ever is the purpose. In general, creating a new graphical editor with GRAPH is a matter of days.

REFERENCES:

[1] J.Jormakka-M.Negro-M.Butto:E-P308 Deliverable 2, EURESCOM (to appear).

[2] J.Jormakka: A model for SSP for dimensioning Intelligent Networks, IN'94 Heidelberg 1994.

[3] F.Langois-J.Regnier:Dynamic congestion control in circuit-switched telecommunications networks, teletraffic and datatraffic, ITC-13, Elsevier,1991.