A Methodology for Implementing a Stress Workload Generator for the GTP-U Plane*

David Navratil, Nikolay Trifonov, Simo Juvaste

University of Joensuu, Department of Computer Science
Post Box 111, 80101 Joensuu, Finland
{david.navratil, nikolai.trifonov, simo.juvaste}@cs.joensuu.fi

Abstract. We present a framework for developing a traffic generator that produces massive, realistic network payloads. The techniques and methods in this article can be easily applied to any stress workload generator for network traffic simulation. Here, as the system to be tested, we use the UMTS/GPRS backbone including SGSN and GGSN, which utilizes GPRS Tunnelling Protocol (GTP-U) user plane messages to carry user data packets. The proposed workload generator system is characterized by high, real traffic load, economical standard hardware, scalability, and flexible extensibility. A large number of independent participants, such as mobile users and Internet servers, are modelled. The realism of traffic is achieved by using a layered modelling approach starting from the user/application level and ending at the network layer. High system throughput is obtained by exploiting preconstructed packet buffers (templates), packet filters, network interface polling, and an efficient, adjustable time resolution scheduler.

1 Introduction

General Packet Radio Services (GPRS) introduced new network elements in the public land mobile network architecture. The new elements are the Serving GPRS Support Node (SGSN) and the Gateway GPRS Support Node (GGSN). The role of the SGSN is to handle mobility management, authentication, and register functions. The GGSN provides access to a public data network, such as the Internet or the X.25. From the external networks' point of view, it is a router to a subnetwork. These two nodes are directly interconnected in the network architecture and form the core of the mobile data network. Therefore, the requirements on them are very high and careful attention must be paid to network planning and testing before the network is launched for public use. If the performance of the network does not satisfy customers' demands, then the situation can lead to the loss of customers. It is not unusual that a GSM network includes only one GGSN, through which all data transfers between Internet servers and subscribers' terminals are routed. In 3G networks (such as UMTS) the role of the nodes is similar, but the requirements are even higher. The services for 3G networks include high bandwidth, low latency applications, such as video-calls.

* This work was supported by Nokia Research Center

Network testing can be seen as a two-phase process. A communication network is a distributed system with many different components communicating through standard interfaces. Thus, the communication between nodes has to be tested first. This type of test can be called an interoperability test. Interoperability tests cross-check the functionality of network elements. In the second phase of network testing, the goal is to test the throughput of the network and to find the limits of and bottlenecks in the system. Such tests are known as stress tests. To perform the stress test, a workload generator, which creates data traffic for the network being tested, is needed. Also, the stress test generator measures the performance of the system being tested.

In case of SGSN and GGSN stress testing, the performance and quality requirements for the workload generator are at least as high as the requirements for the actual support nodes. As mentioned above, a GPRS/UMTS network can include only one GGSN; this node must be able to handle data packets at a rate of several Gbps. Therefore, if engineers want to massively test the core network (i.e., SGSN and GGSN of the network), they need a workload generator that is able to generate data traffic at such a high rate that it saturates the network. Moreover, in addition to answering questions such as “How many packets per second can the nodes handle?” and “What is the actual throughput?”, engineers are also interested in testing the quality of services (QoS) and in observing the behaviour of the network from the mobile user's point of view. Thus, the workload generator must allow engineers to describe the generated traffic in terms of mobile users and applications.

Our approach to using a workload generator for testing the SGSN and GGSN consists of two parts, a control and signalling module and a data generation module. The first module performs signalling and control communication with the node being tested; the second module takes care of data generation. This article concentrates on the data generation module, which utilizes the GPRS Tunnelling Protocol (GTP). We describe a methodology for an efficient and inexpensive implementation of a stress workload generator.

Existing Generators and Related Work

Generally, a generator, which sends data through a network at some level of abstraction, simulates a real process or an application. The sending and receiving of data packets, which takes place in a discrete time, is considered to be an event in the system. Thus, the generator can be seen as a discrete event simulator. Methods and modelling techniques developed for discrete event simulation are used during an implementation of the generator. The level of abstraction used during the design of a workload generator plays an important role. Some workload generators use models represented by a stochastic process that simulate only the packet size and the arriving time of the packets at a low-level network interface. Other types of workload generators are built on models that try to describe user interaction with a system. Such models have a layered architecture and allow a user to create a large set of various user-behaviour profiles [5]. Moreover, the layered models can produce traffic patterns that are more realistic. Naturally, the generators implementing the layered approach are more complex and require more processing time during simulation than generators using one stochastic process. As a result, however, the performance of the system can be observed at both user and application levels.

Demands of the current market for such generators and testing tools have inspired companies around the world to create the appropriate products. Among such tools are MGTS *i3000* by Catapult Communications [2], Cellular Performer Analyser by RADCOM[13], Cutting edge GPRS Support Node Testing offered by Hughes Software Systems [6], and EAST for UMTS by ipNetfusion [9]. These products are complex test suites for testing many network interfaces.

2 Architecture

The endpoints communicating via a GPRS/UMTS network are mobile terminals and Internet servers. We designed the data generation module (DGM) framework to be based on the simulation of these elements. Therefore, the basic item in the data generation module is a model that represents an application running on a mobile terminal (e.g., a WAP or WWW browser) or, correspondingly, an Internet server providing a service (e.g., an HTTP server). Obviously, the data traffic that is generated depends on the models running in the system. In other words, the data generation process is controlled through the models. The control over models includes creating, starting, stopping, and deleting models and setting model characteristics.

Scalability
DGM is designed to run on "inexpensive, standard pieces of hardware", which are, in practice, personal computers or entry-level servers. To increase generator throughput, we can add more computers to DGM. These issues lead us to the framework shown in Figure 1.

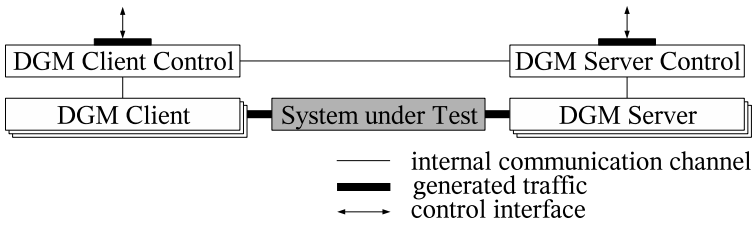

Fig. 1. DGM scheme

DGM consists of two control nodes and several data nodes. The controls nodes are DGM Client Control (DGCC) and DGM Server Control (DGSC). They provide control interfaces for users of DGM. DGM Client (DGC) and DGM Server (DGS) are data nodes that host the models. Data traffic is generated between the DGM data nodes.

We consider the GPRS/UMTS core network as the system under test as shown in Figure 1. We simulate the mobile terminals and Internet servers communicating over the network. However, the communication channel between the end elements also includes other components of a mobile network, see Figure 2. In order to test SGSN and GGSN, the DGM data nodes replace all other parts of the communication channel. The number of replaced components depends on each test scenario. Examples of the test scenarios are displayed in Figure 3. The scenarios show that the DGM clients can occupy the place of MT, GERAN/UTRAN, and possibly SGSN. Correspondingly, the DGM servers substitute Internet servers located in the public data network (PDN) and possibly GGSN during testing.

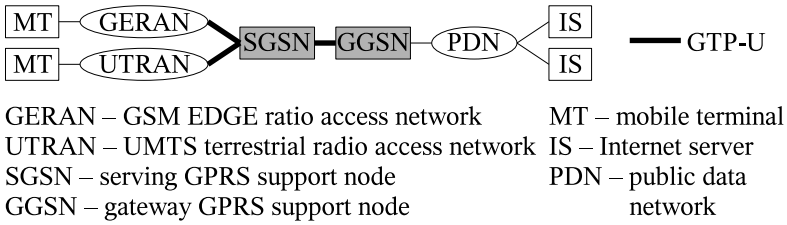

Fig. 2. GTP-U in GPRS

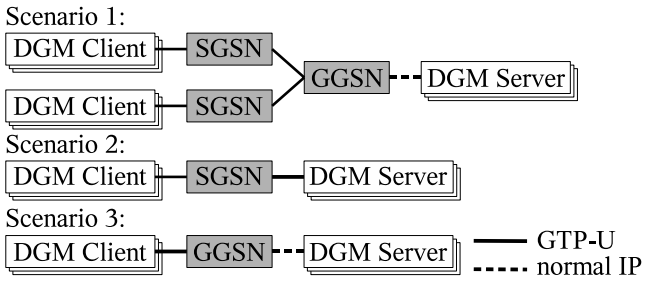

Fig. 3. Test scenarios

Scenario 1 shows the stress test where both SGSN and GGSN are involved. The DGM clients model traffic that comes from UTRAN/GERAN. Between the DGM client, SGSN, and GGSN, the data are transferred over a GTP tunnel. After passing through GGSN, normal IP network traffic is used. The DGM servers simulate Internet servers with various services. In scenario 2, only SGSN is being tested. In this case, the DGM clients replace UTRAN/GERAN and DGM servers replace GGSN. Both connections use GTP tunnelling. Scenario 3 is similar to the first one, except that DGM clients also simulate SGSN nodes.

2.1 Control Nodes

The control nodes provide an interface for simulation management. The duties of the control node include three main control tasks: monitoring and regulation of the load on system resources, user/application model manipulation, and statistical information reporting.

The data that are generated are sent through GTP-U tunnels; thus, the responsibilities of the resource management include taking care of the assignment of tunnel end-point identifiers and distributing load among the data nodes. In the simplest scenario, the client control and the server control distribute the load among the data node uniformly by assigning the same number of models to each data node. If system speeds of the data nodes vary, the loads of the nodes have to be monitored. In this case, a new model would be created at the node with the lowest load. This approach, however, fails to take into account varying states of the models. A large set of idle nodes might activate simultaneously, thus overloading a node. Consequently, the uniform distribution of the models among the data nodes and running the nodes on the same hardware appears to be safer.

DGCC chooses the DGC where a model is going to be created. This information is not known outside the DGM; therefore, all model manipulation has to be done through the control interface of DGCC. For this purpose a unique identification number is associated with each model. Model manipulation is implemented by sending commands, including the identification number, to DGCC.

Statistics reporting utilizes the client-server architecture. The statistics about the simulation can be retrieved from the system by any process connected to DGCC and by requesting a report of the statistics. After the request is received by DGCC, the statistics are periodically sent to the process. Then the process can visualize and/or store the statistics data in its own way. The statistics measured by the system include 36 values, such as current traffic volume (bits/s, packets/s), average round trip time, TCP retransmissions, and number of dropped IP packets. The values are measured for each protocol and traffic class.

2.2 Data Nodes

DGC and DGS are discrete event simulator components of the DGM. Both client and server have the same design schema, which is shown in Figure 4. The process flow of the data node is based on the processing of events that appear asynchronously in the system.

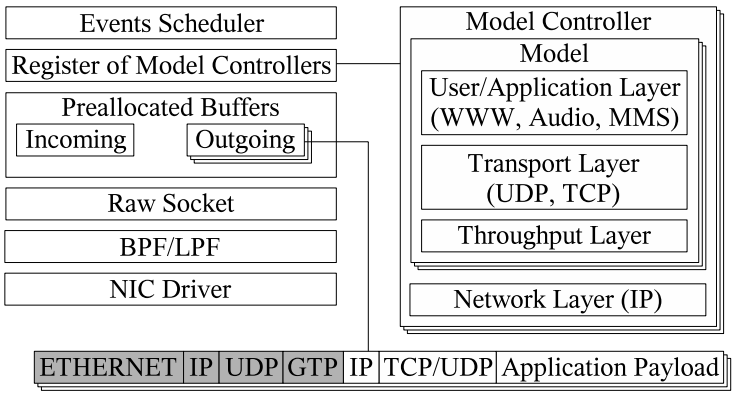

Fig. 4. Data node scheme

A Layered Approach to Implementing Models

The model is an object that simulates an application from the user's point of view. Inside DGM, the model is a composite of smaller model entities arranged in a layered hierarchy. Because DGM simulates network applications, it is obvious that the model entities correspond to the protocol hierarchy; see Figure 4. The lowest model entity simulates the network layer protocol, which is in our case Internet Protocol (IP). The throughput layer is not usually part of the ordinary protocol stack. Because the workload generator is intended to simulate a mobile environment, where the connection speed is limited by Packet Data Protocol context (PDP), this layer is included in the hierarchy. The purpose of this layer is to artificially limit the communication speed according to the PDP context assigned to the model. The next highest layer is the transport layer. Model entities of transport protocols such as UDP and TCP are found at this level. The User/Application model entity is situated at the top of the layered hierarchy. This layer is responsible for the simulation of user and application behaviour. The top layer may be further divided into two separate layers. This division offers more flexibility during concurrent simulations of different user's profiles [5]. In any case, the hierarchy presented here already gives testers the possibility to observe the behaviour of each protocol layer as the load in the tested system increases. This is achieved by gathering statistics for each modelled layer.

The model entities in the layered hierarchy need to interact with other models. For this purpose, every layer provides a predefined set of services to a higher layer. The communication between layers is accomplished by means of service primitives. The service primitives are divided into four sets: requests, indications, responses, and confirmations. The request primitive is used when a higher layer requests a service from the next lower layer. The lower layer invokes the indication primitive in the next higher layer in order to provide information about activity on the lower level. The response primitive is used to confirm the receipt of the indication primitive received from the next lower layer. In the same sense, the confirmation primitive notifies the next higher layer about the successful accomplishment of the activity invoked by the

request primitive. One request primitive usually requires a couple of other request primitives in the lower layer, and a couple of indications in the lower layer are needed to generate the indication in the next higher layer. All this depends on the services being modelled.

A distributed system is often characterized by its stochastic behaviour and time constraints within the system. In our case, stochastic automata are used to describe the behaviour of each model [1][3]. A stochastic automaton does not differ much from a finite automaton. The main difference is that a stochastic automaton has clocks. The clock can be set to a value upon the entry of a desired state. After the clock is set, a countdown of the clocks begins. The automaton can move from one state to another if there is an edge between the states, the input symbol is the same as the symbol assigned to the edge, and the set of clocks associated with the edge are all zeros. Further, we extend the stochastic automaton to the input/output automaton by changing the transition function. The transition function includes input/output events. An event will be generated in the system when a transition from one state to another is implemented and an output event is assigned to the transition used. We implement the countdown process through the scheduler. The event representing the achievement of zero in the clock is scheduled. The scheduler notifies the appropriate model by invoking the indication primitive.

Model Controller

We take into consideration mobile users who can use several services simultaneously, e.g., web-browsing and audio streaming. A multi-service server can run several services of different types at the same time. From the implementation point of view, this means that for each mobile user, there can be several client application models running on the client side. Correspondingly, there will be a server model for each client application model on the server side. We present a model controller that acts as a user on a client node and as a multi-service server on a server node. The model controller stores all running models of the user/server.

Scheduler

To run a number of users and applications concurrently, some simulators start new network models by forking a new process or thread in the operating system. This approach causes the following problems. First, the number of processes and threads (and thus simulated entities) running simultaneously is limited in the operating system. Second, it takes additional effort to implement logical time for every running entity and synchronizing them with the whole simulation process. Finally, process/thread context switching takes processor time. Thus, considering modelling aspects, a more efficient, single process solution is applied in DGM.

DGM simulates various network entities by running their models through the life cycle of the simulation. They behave as independent participants and the events produced by models are asynchronous by nature. Every event is marked with a logical time-stamp and put to the scheduler. The scheduler implements a logical time line and triggers the scheduled events.

The scheduler is a dual ring buffer consisting of slots that stand for discrete time units of the simulation. Every slot consists of a list of events scheduled to the same

time unit. The DGM time accuracy is defined by the time slot resolution of the scheduler. The default resolution is 1 ms in our generator. The length of the scheduler buffer can be extended months ahead to support simulation scenarios using user/application models with very low message frequency. We use dual buffers to achieve both high resolution and to schedule events very far in the future. Events scheduled to trigger within, e.g., the next minute are stored in the buffer with 1ms resolution. More distant events are stored in the low resolution buffer until they are moved to the high resolution buffer, just before the scheduled time. The models progress by processing the list of events in the slot. The logical time corresponds with real time accurately, except for occasional short term gaps, mostly in the case of process switches.

System Register

The system register is a container of model controllers. Its functions include insert, delete, and find operations. Because DGM should support a huge number of users/servers, the register implementation has to perform the operations in constant time. Obviously, the solution is to represent the register as a hash table. We use the open addressing hash table, with a double hashing, as a collision resolution method. The hash functions are optimised for usage of IP addresses as keys of the hash table.

Preallocated Buffers

Considering the performance of the system, sending packets is also critical as an efficient implementation of models. However, if we use a standard OS UDP/TCP implementation, the preparation of outgoing packets usually involves buffer copying. It is one of the bottlenecks of stress generators because it consumes memory bandwidth and CPU time. To avoid this problem, we use preconstructed templates for outgoing packets. A packet template is a preallocated buffer with already filled-in fields of all network protocol headers and payloads that are the same for a set of models. Creation and initialization of templates is done only once during the simulation. There are various templates that we use in DGM clients and DGM servers. Considering headers of Ethernet, IP, UDP, and GTP protocols, which are shown in light grey in Figure 4, most of their fields remain unchanged during the simulation. For example, source and destination ports of a UDP tunnel header can be fixed for the specific template. The processing of arriving packets also can involve undesirable buffer copying. Because incoming packets are processed in a sequential order of arrival, only one preallocated buffer for incoming packets is needed. Actually, instead of receiving the whole packet, it is enough to inspect a few pieces of data from each packet. Summing up, the approach that was presented above speeds up the process of constructing the packets to be sent and received.

Raw Socket

To apply templates of the Ethernet datagrams explained above and to avoid packet data copying, we decided to use raw sockets. Such a socket allows us to communicate directly with the network driver when sending or receiving a packet; i.e., the usual protocol stack-handling, such as IP/TCP or IP/UDP processing, is avoided. In other words, the Ethernet frame goes directly from the network card driver to the

application and vice versa. In a Linux system, the socket mentioned above is presented by a PF_PACKET protocol family starting from post-2.0 kernel releases [8].

Filter
Having opened a raw socket, the application starts to receive all Ethernet frames arriving to the host. All incoming packets have to be processed, even those that are not relevant to the application. The validation of packets destination on the application level would cost an extra reception overhead. To avoid this, the elegant solution is to use a filtering mechanism right after the network driver to drop the packets which do not pass through the filter. Such filters are BSD Packet Filter [11] and Linux Packet Filter [7][8].

Network Interface Card Driver
The main goal of a stress workload generator is to be able to saturate the target node that is being tested with a heavy network load. The given requirement assumes optimization of all components of DGM involved in traffic generation, including the network interface card (NIC) driver. The usual way to notify the processor of a network event, such as the arrival or transmission of a packet, is done through interrupts. This works well on low bandwidth networks, but it degrades results at high rate of packet arrivals. Indeed, an interrupt is a too time-consuming operation to be called, e.g., 200,000 times/s. The alternative method is known as polling. In contrast to using interrupts, in pure polling systems, the device driver “listens” for incoming packets and for transmit-completion events. Because several arrival packets or transmit completion events can be read over a single poll, the overhead of network events handling is reduced. The possible hybrid implementations of NIC device drivers combining both interrupt and polling techniques are presented in [4][12].

3 Results and Conclusions

We have presented a framework for implementing a stress workload generator. Our primary goal was combining real-like traffic patterns with the high efficiency of a generator.

Modelling Users and Applications
The similarity between real and generated traffic patterns depends on the accuracy with which the models were designed. Perhaps the most popular network model is WWW. Let us consider this model as an example demonstrating the advantages of the layered approach.

The behaviour of the WWW model as the composition of the model entities depends on parameters that specify the behaviour of each layer. In the network and transport layer, the parameters of the model entities such as maximum transmission unit and receiving and sending window size are the same as in implementations of the corresponding protocols. In the application layer, the WWW model is defined by the

average request/response size, the number of embedded objects in a WWW page (e.g., pictures, JavaScripts, external CSS), and the number of concurrent connections opened to a server. The user behaviour is specified by the average session length, the number of pages visited in one session, and reading time. In our case, the application and user layers are modelled by one entity. Furthermore, the throughput layer artificially limits the speed of connection between the mobile terminal and the Internet server. Statistics of real (a) and generated (b) network traffic of a user reading the news with the Firefox browser are presented in Figure 5. The application/user model parameters were modified according to this special case. The average values of the parameters were 30 seconds for reading time, 500B for request size, 10KB for reply size, and the number of embedded objects was around 20. Since Firefox uses two concurrent connections for retrieving data from a server, the same situation was simulated. Because real traffic was measured on a workstation with a fast Ethernet connection, the throughput layer parameters were set to values specific for wired connection.

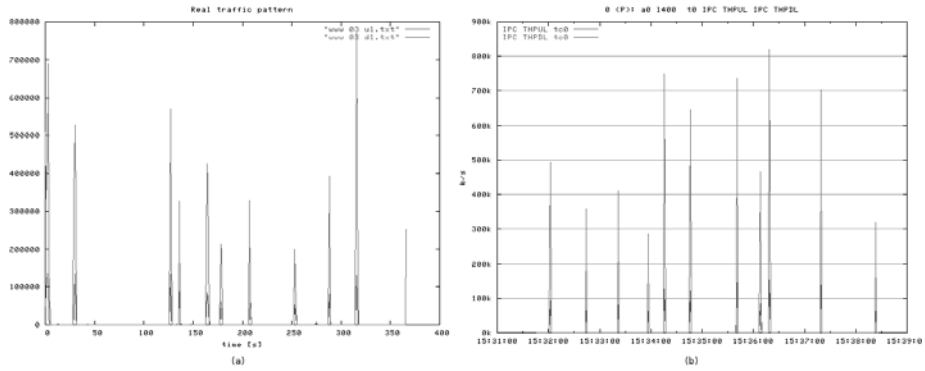

Fig. 5. WWW traffic in wired networks- (a) real, (b) generated

By setting up specific model parameters, predefined model behaviour is expected. However, it is typical that the network is utilized by various applications. Multiple traffic flows are transmitted through the same network elements. That leads to such situations as network congestions and queue overflow. In other words, all network participants, consuming limited network resources, affect each other.

Our DGM implementation has built-in capabilities for simulation of different applications, which include WWW, unidirectional UDP and TCP audio streaming, E-MAIL, and MMS. All these models can be simulated concurrently in the system. The concurrent simulation of any combination of models gives a user the possibility to test the GSN nodes under lifelike conditions.

Proceeding with the WWW example, let us consider how web traffic competes with other applications for the network resources. We created one thousand WWW PDP contexts, and then ran one thousand UDP audio streaming models to produce

additional network load. The snapshot of the seven minutes simulation can be seen in Figure 6.

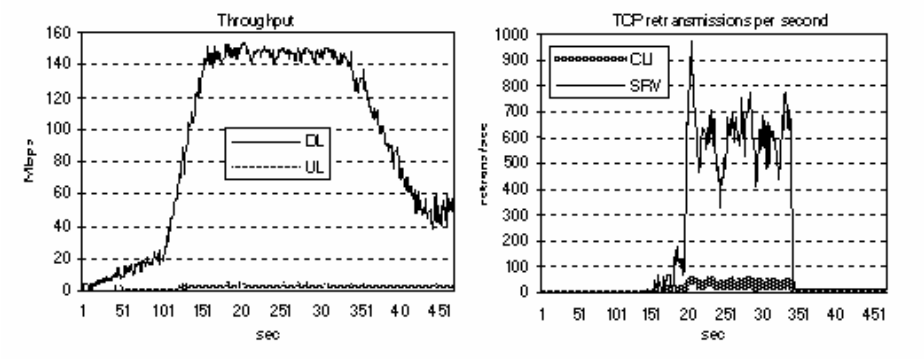

Fig. 6. WWW and UDP streaming traffic pattern

The downlink direction is more interesting to analyse because it is always more loaded for both model types. At the beginning only WWW models were created in the system. The number of TCP retransmissions was zero at that moment. After about 100 seconds of the simulation, UDP streaming users started their activity. The throughput peak of about 150 Mbps shows clearly this situation. As it can be seen in the second graph, after the UDP streaming traffic appeared in the network, the number of TCP retransmissions from web servers to mobile clients increased rapidly. This is usually caused by congestion or by other problems in the network.

GPRS/UMTS networks are still evolving. The number of mobile users is increasing, their behaviour is changing, and new mobile applications are becoming available on mobile devices. In order to make proper tests, a real network has to be monitored and the parameters must be set according to actual conditions. Some studies on how to rapidly parameterise models have already been made [10]. Moreover, some assumptions about the evolution of the network and future requirements on the network should be taken in account.

The DGM performance tests were made with two PCs, each hosting one data node process. Both server stations had the same hardware configuration: Intel Xeon 2.40GHz, 512MB RAM, PCI-X (64 bits/100 Mhz), Intel 82544EI Ethernet controller. The current version of the DGM was able to produce about 500 Mbps with 90 000 packets per second in this configuration. Verification tests of DGM with real network elements were also conducted.

Future Work

We will consider the extension of the system with new models such as Voice over IP applications, which will also require a new model entity for Real-Time Transport Protocol (RTP). Although new models can be easily added to the layered hierarchy, there is a small drawback, which is related to the integration of new models, to the

current system. The implemented models are completely built into the system and the addition of new models requires small changes inside the core of the system.

The DGC and DGS are implemented as one-thread applications. Properly threaded applications can benefit hyper-threading and dual-core technologies by increasing their operational speed. Hence, the next step for improving the performance of the DGM would be to implement the data nodes with a few threads to utilize multi-threaded, multi-core, and multi-processor systems. In order to change the structure of the data nodes, the relation between the events within the system has to be found and the sets of independent events identified. The disjointed sets can be processed in parallel by using threads.

References

1. Bryans, J., Derrick, J.: Stochastic Specification and Verification. In *Proceeding of 3rd Irish Workshop on Formal Methods, Electronic Workshops in Computing*, pp. 20, July 1999.
2. Catapult Communications: *MGTS i3000.* Internet WWW-page, URL: http://www.catapult.com (30.11.2004).
3. D'Argenio, P. R., Katoen, J.P., Brinksma, E.: An algebraic approach to the specification of stochastic systems (extended abstract). In *Proceeding of the IFIP Working conference on Programming Concepts and Methods*, pp. 126-147, 1998.
4. Dovrolis, C., Thayer, B., Ramanathan, P.: HIP: Hybrid Interrupt-Polling for the Network Interface. In *ACM SIGOPS Operating Systems Review*, vol. 35, iss. 4, pp. 50 – 60, October 2001.
5. Hlavacs, H., Kotsis, G.: Modeling User Behavior: A Layered Approach. In *Proceedings of the 7th International Symposium on Modeling, Analysis and Simulation of Computer and Telecommunication Systems*, pp. 218-225, October 1999.
6. Hughes Software Systems: *Cutting edge GPRS Support Node Testing*. Internet WWW-page, URL: http://www.hssworld.com (30.11.2004).
7. Insolvibile, G.: Inside the Linux Packet Filter, Part II. *Linux Journal*, vol. 2002, iss. 95, pp. 7, March 2002.
8. Insolvibile, G.: Linux Socket Filter: Sniffing Bytes over the Network. *Linux Journal*, vol. 2001, iss. 86, pp. 8, June 2001.
9. ipNetfusion: *EAST for UMTS*. Internet WWW-page, URL: http://www.ipnetfusion.com (30.11.2004).
10. Lan, K.-C., Heidemann, J.: Rapid model parameterization from traffic measurements. In *ACM Transactions on Modeling and Computer Simulation (TOMACS)*, vol.12, iss. 3, pp. 201-229, July 2002.
11. McCanne, S., Jacobson, V.: The BSD packet filter: A new architecture for user-level packet capture. In *Proceedings of the Winter 1993 USENIX Conference*, pp. 259—269, January 1993.
12. Mogul, J. C., Ramakrishnan, K. K.: Eliminating Receive Livelock in an Interrupt-Driven Kernel. In *ACM Transactions on Computer Systems (TOCS)*, vol. 15, iss. 3, pp. 217 – 252, August 1997.
13. RADCOM: *Cellular Performer Analyzer*. Internet WWW-page, URL: http://www.radcom.com (30.11.2004).