Advanced Generation Tool of Application's Network Traffic*

Attila Petróczi Sándor Molnár

Department of Telecommunications and Telematics
Technical University of Budapest
H-1117 Budapest, Pázmány Péter sétány 1/D, Hungary
{petroczi, molnar}@ttt-atm.ttt.bme.hu

Abstract

We introduce a new traffic generation and monitoring tool which was developed for performance evaluation of TCP/IP networks. Our tool, called Agent-Ant, supports a number of TCP/IP applications and new applications of future can also be easily included. Agent-Ant supports multiple connections and can measure different traffic characteristics on the application level. The main concepts, the architecture and some performance results of the development of Agent-Ant are described in this paper.

1 Introduction

The main targets of the Internet research are to understand the characteristics of Internet traffic and the mechanisms of the new applications on the web and to evaluate the performance of new protocols. To achieve these goals different approaches can be taken. One of them is to use simulations. A number of successful simulation tools have been developed and applied recently, e.g. REAL [7] or NS [11]. However, the applications of simulation approaches are not adequate in several cases since they eliminate the impacts of different network mechanisms such as the operating systems. Another way to achieve our goal is to develop and use network performance measurement tools, which use actual data and the measurements are carried out in real environments in contrast to simulations. This benchmarking approach has a number of advantages including the evaluation of actual protocol implementations, the usage of a large number of real traffic generators and the monitoring of end-to-end performance. We have chosen this approach and developed a tool called Agent-Ant which is described in this paper. There is an on-going effort to develop such benchmarking tools. One of them is called NetPerf [6] developed by the Information Network Division of Hewlett-Packard Co. This is a benchmark system that mainly uses bulk data transfer and measures request/response performance with TCP or UDP. It can also mimic the behavior of HTTP. Another development is the LADDIS [17] which can measure the throughput and the average response time for an NFS server. An important tool is the SURGE (Scalable URL Reference Generator) [1], which can generate realistic WWW workload. However, SURGE is not a real benchmarking system since it does not contain any measurement functions. There are many other tools (e.g. ttcp, nettest, etc.) with different but limited features, which can use only single point-to-point connections and can measure only the main throughput. The DBS [12] (Distributed Benchmarking System) developed at the Graduate School of Information Science at Nara Institute of Science and Technology is a successful tool and some of the concepts have been utilized as the basis for our Agent-Ant system. The main goal of DBS was to provide a framework to TCP performance evaluation. DBS supports multiple connections and can measure the throughput in each data transmission. The application of DBS provided a good insight into the working mechanisms of TCP. Another successful tool, called NetSpec [9, 10], was developed by the Information and Telecommunication Technology Centre of the University of Kansas. It is an end-to-end performance tool including a number of traffic source generators and working modes. NetSpec also supports multiple connections and different protocols (TCP, UDP). The NetSpec system has the same framework as the DBS, but it contains several simple models.

In spite of the success of the DBS and NetSpec in performance evaluation they do not have advanced features in generation of realistic network workload. Our tool implies the functionality of these tools and it is further developed to extend its features to advanced

*The research was supported by the Inter-University Centre of Telecommunications and Informatics.

workload generation. The Agent-Ant is able to incorporate complex request/reply based application models to generate realistic network traffic. To achieve this goal the Agent-Ant has an embedded Tool Command Language Interpreter that executes the model programs. This is the unique and strongest point of our tool, because in this way the development of various models is highly independent from the system and there is no need to recompile the Agent-Ant when a new model is under development. It is possible to set several measuring parts of Agent-Ant at various locations of the network which provides global network performance evaluation.

2 The Architecture of Agent-Ant

Our aim was to develop a flexible, model-based traffic generator for TCP/IP-based applications, which is able to generate realistic traffic workload.

We had the following requirements to this tool:

- support as many models of TCP/IP-based applications as possible,
- allow the easy development and integration of new models,
- use detailed models that mimic the real interactions on the network,
- measure as many traffic characteristics as possible (on the application level),
- generate workload for arbitrary IP networks,
- support multiple connections between the hosts according to the needs of the application models.

Besides these features we intended to develop a highly configurable and easy-to-use centrally controllable measuring and traffic generating tool. The generator system was given the name "Advanced GENeration Tool of Applications Network Traffic", shortly Agent-Ant.

2.1 The Layering of Agent-Ant

The Agent-Ant has been split into three parts to achieve the easy development of the application models. The first part is the "communication framework", the second part is a TCL (Tool Command Language) interface, and the third is the model level. The purpose of the communication framework part is acting as an application programming interface to the TCL interpreter that provides the running environment for model programs. In other words, the basic architecture of the traffic generator is the communication framework together with the built-in TCL interpreter. Users can write varying model programs in TCL language. In the paper the communication framework together with the built-in TCL interpreter will be referred as "basic framework". Figure 1 shows the layering of the Agent-Ant. Since real TCP/IP-based applications are client-server applications, the model programs work the same way. According to this requirement there are client and server models on the model level that use the functions provided by the basic framework. With these functions the client model programs open and close connections to their server model, transmit messages in both directions, both ends log the information of the messages (such as time-stamp, message type, size etc.). The model programs generate traffic using the functions provided by the basic framework. The framework subsystem ensures the centralized control, provides the functions to the model level and implements many other tasks and features. To fulfill these tasks the framework has a distributed architecture.

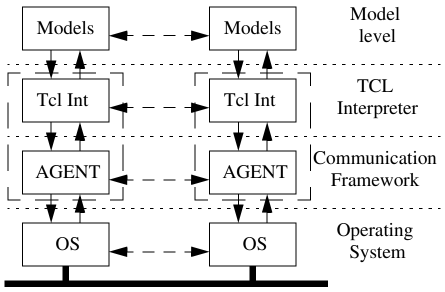

Figure 1: The Layering of the Generator System

2.2 The Basic Framework

One control daemon and many test daemons constitute the Framework. The control daemon is responsible for the centralized control with tasks:

- taking over the configuration information from the user,
- processing the configuration information,
- sending the appropriate information to the test daemons,
- keeping the test daemons under control while running the program without generating additional network traffic,
- ensuring the synchronization between the daemons,
- collecting the results from the daemons at the end of the test.

The test daemons have to provide an environment to the model level with tasks:

- launching the model programs according to the received configuration information,

- creating and closing multiple and bidirectional connections of specified types to the other daemons with specific options on demand of the model programs,
- ensuring the transmission of the messages that came from the model level through the network,
- logging the main characteristics of the messages sent or received on the connections and saving that log data into a local file,
- scheduling the model programs,
- ensuring timing, transmission, reception, random and specifically distributed number generation and log functions to the model level,
- sending log files and status information to the control daemon at the end of the test.

According to the requirements described in the previous subsection we selected the basic concepts of the Distributed Benchmarking System to developing our framework [12]. Figure 2 shows the architecture of the traffic generator.

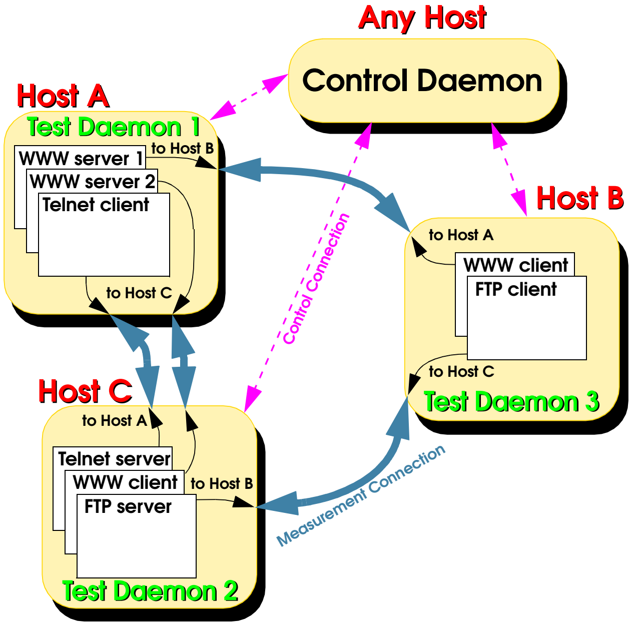

Figure 2: The Architecture of Agent-Ant

2.3 The Functionality of the Basic Framework

The user has to create the command file first in the Agent-Ant. The command file contains the configuration data in blocks. Each block has the configuration parameters of one application client model like the type of the client model, the physical host where the client has to act, the destination host where the server model runs, the clients own TCP port (optional), the start- and the end-time of the client models activity, path and filename for the local log-file (optional). The control daemon has to check and process the command file. This daemon determines which application servers should run on each participant hosts and creates their configuration data for the application servers according the following requirements: Only one parent server of an application model should exists on a participant host. This server-stub must run until all its dynamically connecting clients and child servers finish their job.

To understand these requirements we must take a short outlook. In real applications the server listens to its well-known port for connection requests. When a connection request arrives, the server duplicates itself. The new server accepts the connection and handles the connected client. In the Agent-Ant system the communication framework has to provide this functionality of the servers because the model programs must be easy to develop. This is because only one server-stub of an application model type (e.g. WWW) should be started on a host and it has to listen to the well-known port of the particular model. At the arrival of a connection request, the original (parent) server will launch a duplication of itself (child server) that will handle the request. Figure 3 shows how a model client is connected to a model server in the Agent-Ant.

After the test daemons are started, the control daemon has to connect to them and send the configuration data as well. When a connection request arrives from the control daemon to a test daemon, it duplicates itself. See step one in Figure 3.

The new copy of the test daemon will accept the connection and the original one will wait for a new connection. The new process will receive its configuration data from the control daemon. After the transmission of the parameters the test daemon has to synchronize itself to the control daemons clock. According to the received configuration data a test daemon can decide whether its application model would be client or server type. See step two in Figure 3.

If the model is a client then the test daemon has to wait until start time and then launch the execution of the application model in its TCL interpreter. If the model is a server then the test daemon will act as a "parent server". Like in real applications, in the framework the "parent server" is not a real server model but only a base process, which accepts connections, duplicates itself and launches the "child" server.

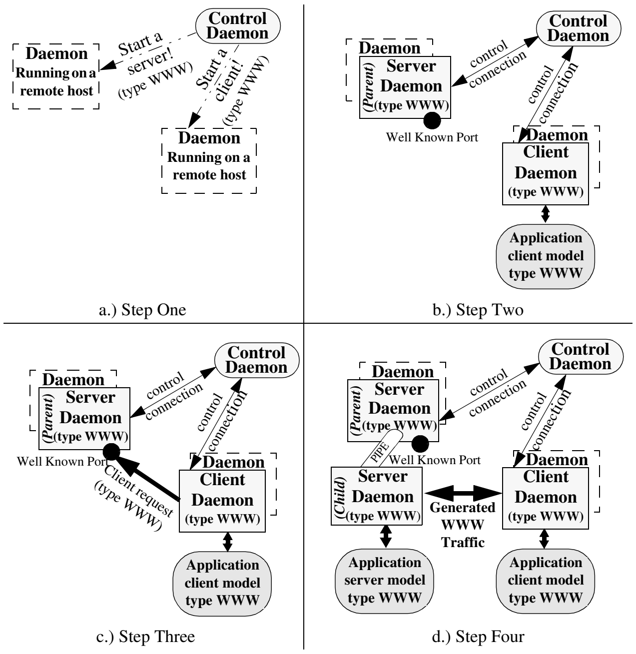

Figure 3: Connection Setup in Agent-Ant

Also the "parent server" listens on its well known port for the connection requests of the client models and duplicate itself at the arrive of a connection. See step three in Figure 3.

The new copy has to launch the model server and the "parent server" will wait for new connections until the end of the test. In this way the client models dynamically open new connections to the model servers (in the reality to the "parent server") . The "parent server" has to keep its copies (in other words its children) under control. See step four in Figure 3. Note that the test daemon server and the model server exist just when the model client tries to connect to the "parent server" at its well-known port. The well-known ports of the "parent servers" are configurable. These can be defined by the user in a file containing the kind of models and its assigned port numbers. This file is processed by the control daemon and it has to transmit the information of port numbers to the test daemons.

Now, the traffic generation is performed by client and server models using the functions of the framework. The framework has to log some characteristics of the generated traffic into a local file on the host for further study. The log-file contains the characteristics of every message sent like the transmission and reception time (timestamp), the name (or address) of the destination or source host, the type of the message (request/reply) the size of the message and some other information that are optional. It is the users' responsibility to check that the log-file is not on a network mounted volume, because writing to such a volume would cause unwanted network traffic.

2.4 The Model Level

It consists of many programs that generate network workload by using the functions provided by the framework. The workload generation can be accomplished in many ways. Both constant bulk transfer and detailed application model can be applied that mimics the functionality of the real applications. Between these two extremes many other simple or complex application models exist. The user level models are part of the model level. Nearly every kind of traffic model based on the client-server principle and request-response communication can be incorporated into Agent-Ant [3, 4, 5, 8, 9, 14, 15]

There are some limits on the traffic generation. A problematic issue is the generation of aggregated traffic. In this scheme aggregated source models can be built as well, but these models can not use the request-response mechanism. Although the traffic generated by many single-source models is more realistic than the traffic generated by an aggregated model the performance of the operating system can strongly limit the number of the simultaneous processes.

A workload generator such as Agent-Ant that works on the application level can not well utilize packet level models. In case of interactive traffic there are significant differences between the traffic generated on the application level and the traffic can be captured on the packet level. This is because the delayed acknowledgement and the Nagle algorithm have a strong effect. As a result of both algorithms several pieces of interactive application data (e.g. keystrokes) will not be transmitted immediately after generation, but suffer some delay. For bulk data transfer the TCP and operating system mechanisms determine how data leaves the host independently of how the application has sent it. Similar to the other workload generators, the options of the TCP or UDP sockets can be adjusted in Agent-Ant. With the tuning of these options users can influence the operation of the transport layer. For example, in most UNIX implementation Nagle algorithm can be disabled, but it is not possible to disable the delayed acknowledgements.

The C language was selected to complete the implementation. We took care to retain the portability of the system during the development. The Agent-Ant is able to run under UNIX operating systems like

Solaris, Linux etc. The software was compiled under SUN Solaris 2.6 with Gnu C++ compiler 2.7.2.3.

3 Measurements in Agent-Ant

Agent-Ant measures the generated traffic on the network level in every endpoint with an extern packet-collecting tool. The analyzing part of the Agent-Ant has not been complete yet, however, it will be able to evaluate the characteristics of the network, the TCP and operating system mechanisms based on the recorded and detailed traffic data. These characteristics, among orders, are the loss and delay performance of a chosen connection, the statistical characteristics of the packet trace and overall performance characteristics of different end-to-end connections. Agent-Ant can also be used for the performance evaluation of different models developed and suggested in the literature [5, 9, 15].

There are two main types of traffic models used in closed generator systems. We called them one-directional and interaction-based traffic models accordingly to their operations.

In case of a one-directional model the traffic originator sends messages to the receiver which simply logs relevant information upon reception. The generator does not do handshaking or request-reply interaction between the sender and the receiver. There are two possible goals of the sender models. One is to evaluate the performance of the network or computers, and the other is to generate characteristic network workload based on one-way traffic models. These models generate the appropriate workload and transmit it through the network as fast as possible to evaluate the performance.

Interaction based models can mimic the real network interactions and they are capable of capturing the effect of fast traffic workload changes.

The performance evaluation assumes the configurability of the network sockets (e.g. socket buffer size) because they have significant impacts on the resulted packet trace. In the Agent-Ant there exists a function to configure the sockets. A workload generator working on the application level can approximately emulate the packet level traffic using packet level models (adjusting the socket options), but the actual traffic on the network will be far from the workload generated on the application level. Therefore, it is significant to explore the connection between the application level and packet level traffic. The Agent-Ant is able to examine this connection in many aspects.

4 Performance results of the development of Agent-Ant

Our generation tool uses an embedded TCL interpreter. One of our targets in the development of Agent-Ant was to answer whether the communication channel of the TCL interpreter is fast enough for traffic generation or not. The complex request-reply models require non-blocking communication channels. For the TCL modeling we need non-blocking channel handling, but unfortunately this is not the practical case. These functions are not included in the present TCL versions [13, 16]. To provide the functions in the TCL we implemented them in C and integrated them in the interpreter. For comparison purposes a thorough analysis of the communication channel using the included TCL functions or our external C program was carried out. This analysis resulted in interesting results.

In the measurements different-sized messages were sent continuously between two SunSpark stations with the Agent-Ant. The measurements were taken in pairs, one measurement with TCL communications functions and the other with transmission functions written in C. We captured the packets on the network. The resulted traffic patterns are different.

Figure 4 shows the interarrival times between TCP packets in a 300 second long measurement where 5 MB long messages were sent through 32KB socket buffer using the external C functions integrated into the TCL interpreter.

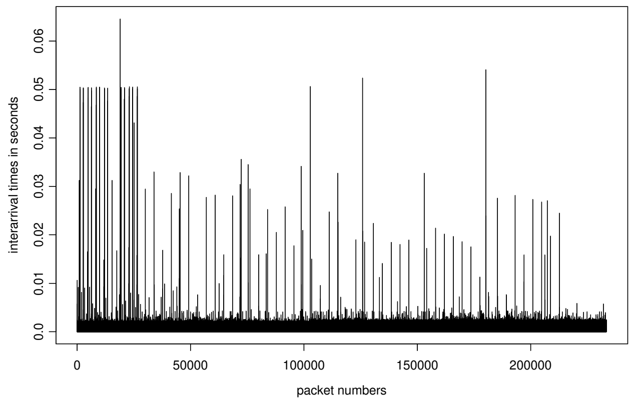

Figure 4: 5MB Message Transfer with C Functions

It can be seen that there are large delays periodically and many smaller delays between them. These small delays are caused by the operating system and the TCP mechanism. The reasons of the greater interarrival times and its periodicity are under our research focus.

Figure 5 shows the result of the same measurement where the embedded TCL communication functions were used.

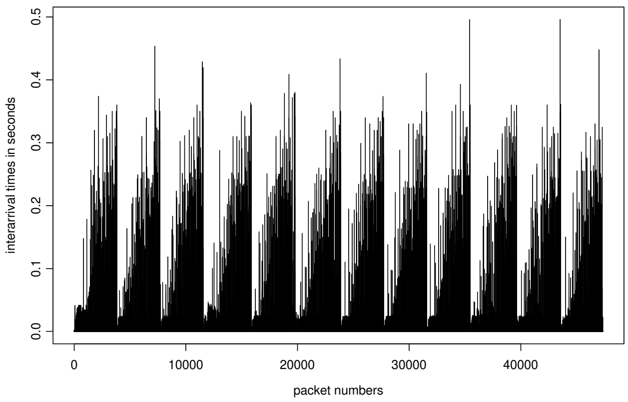

Figure 5: 5MB Message Transfer with TCL Functions

At the first glance the characteristics of this trace is strongly different. The great saw-teeth fits to a 5 MB message sending. It means that at the end of every message sending there is a great delay and the delays are increasing until this point. Of course, there are small delays caused by the operating system and TCP too, but there are intermediate length interarrival times between the great delays. The interarrival times shows three-level periodicity. The reason of this behaviour is the TCL buffering mechanism. In the TCL Interpreter there are two application buffers that can be used for emulate the buffered I/O mechanism on the communication channels. One of them is well configurable and can be turned off, but one can not bypass the other. By this buffer the user can only adjust its size from 8KB to 1MB. We set the buffer size to 1MB during our measurements. Several measurements were taken with different TCL buffer sizes, various message sizes and with several socket buffer sizes. The analysis of these traces are in progress. At this point the conclusion is that the traffic profile is significantly modified by TCL buffering for large file transfers. This effect depends on both the TCL buffer size and the transferred message size. For better understanding we show Table 1 that includes the main statistical results of a measured sequence.

The measurements were taken for 300 seconds long. The number of the transmitted TCP packets generated by TCL functions is smaller compared to our reference generator implemented by C functions. The mean and the standard deviation of the interarrival times was the same using C functions, but they are greater using TCL functions.

	packets	mean (s)	deviation (s)	min (s)	max (s)
32K with TCL	217131	0.001400	0.00209	0.00001	0.17140
32K with C	232403	0.001308	0.001609	0.00007	0.23120
500K with TCL	98930	0.003083	0.012379	0.00006	0.34000
500K with C	231106	0.001318	0.001610	0.00009	0.26176
1M with TCL	113470	0.002694	0.008834	0.00005	0.30003
1M with C	233653	0.001303	0.001016	0.00009	0.26314
5M with TCL	47348	0.0066571	0.031790	0.00006	0.49607
5M with C	233435	0.001305	0.000799	0.00007	0.06454

Table 1: Main Statistical Characteristics

We can conclude that the communication channel should be implemented by external C program in the TCL interpreter instead of the embedded TCL functions in the cases when accurate traffic characteristics are required.

5 Conclusions

We proposed a new traffic generation and monitoring tool for TCP/IP network performance evaluation. Agent-Ant provides a wide range of functions including the easy integration of new source models, the support of multiple connections and the versatile measuring capabilities on the application level. We also described some of our performance results of our developments. Based on this results we have developed the appropriate applicability of TCL in Agent-Ant. We have an on-going research for performance evaluation of TCP/IP networks using Agent-Ant.

References

[1] Crovella, M., Barford, P., "Generating Representative Web Workloads for Network and Server Performance Evaluation," *in Proceedings of Performance '98/ACM SIGMETRICS '98, pp. 151-160, Madison WI.,* 1997.

[2] Danzig, P., Jamin, S., "tcplib: A Library of TCP Internetwork Traffic Characteristics", *USC Technical Report, USC-CS-91-495,* University of Southern California, 1991.

[3] Deng, S., Bugos, A. R., "Access Traffic Model Inferred from WWW Servers", *Proceedings of the International Conference on Telecommunication Systems*, GTE Laboratories, 1996.

[4] Farber, J., Bodamer, S., Charzinski, J., "Measurement and Modelling of Internet Traffic at Access Networks" *1998 EUNICE*, Aug 31.-Sept.3

[5] Frost, V. S., Melamed, B., "Traffic Modelling for Telecommunication Networks," *in IEEE Communications Magazine,* March 1994.

[6] Information Networks Division, Hewlett-Packard Company, "Netperf: A Network Performance Benchmark (Revision 2.0)", 1995.

[7] Keshav, S., "REAL: A Network Simulator", *Technical Report 88/472,* Department of EECS, UC Berkeley. 1988.

[8] Kotsis, G., Krithivasan, K., Raghavan, S. V., "A Workload Characterization Methodology for WWW Applications", *in H. et.al., (eds.): Performance and Management of Complex Communication Networks, Chapman and Hall, 1998, pages 153-173.,* 1998.

[9] Lee, B. O., Frost, V. S., Jonkman, R. "NetSpec 3.0 Source Models for telnet, ftp, voice, video and WWW traffic", *Technical Report ITTC-TR-10980-19*, University of Kansas, January, 1997.

[10] Lee, B. O., Frost, V. S., Jonkman, R., "Introduction to NetSpec", University of Kansas, January, 1997.

[11] McCanne, S., Floyd, S., "ns-LBNL Network Simulator," *Available via http://www-nrg.ee.lbl.gov/ns/*

[12] Murayama, Y., Yamaguchi, S., "DBS: a powerful tool for TCP performance evaluations,"*Performance and Control of Network Systems, Proceedings of SPIE, Volume 3231,* November, 1997.

[13] Ousterhout, J. K., *Tcl and the Tk Toolkit,* Addison-Wesley, 1993.

[14] Paxson, V., Floyd, S., "Wide-Area Traffic: The failure of Poisson Modelling", appeared in IEEE/ACM Transactions on Networking, 3(3), pp. 226-244, June, 1995.

[15] Paxson, V., "Empirically-Derived Analytic Models of Wide-Area TCP Connections", *in IEEE/ACM Transactions on Networking, vol. 2, n. 4, pp. 316-336,* August, 1994.

[16] Welch, B., *Practical Programming in Tcl and Tk,* Prentice Hall, 1995.

[17] Watson, A., Nelson, B., "LADDIS: A Multi-Vendor and Vendor-Neutral SPEC NFS Benchmark," *in Proceedings of the 6th USENIX Large Installation System Administration Conference (LISA VI), pp. 17-32,* 1992.