Analysis and experimentation of an open distributed platform for synthetic traffic generation

Donato Emma, Antonio Pescape' and Giorgio Ventre
Dipartimento di Informatica e Sistemistica, University of Napoli "Federico II" (Italy)
demma@napoli.consorzio-cini.it, {pescape,giorgio}@unina.it

Abstract

This work presents an open distributed platform for traffic generation that we called Distributed Internet Traffic Generator (D-ITG), capable of producing traffic (network, transport and application layer) and of accurately replicating appropriate stochastic processes for both IDT (Inter Departure Time) and PS (Packet Size) random variables. We implemented two different versions of our distributed generator. In the first one, a log server is in charge of recording the information transmitted by senders and receivers and these communications are based either on TCP or UDP. In the other one, senders and receivers make use of the MPI library. A complete performance analysis among centralized version and the two versions of D-ITG is presented. To our knowledge, no similar works are available.

1. Introduction

Network management has so far been dominated by passive monitoring. Emerging networking technologies however force the development of active testing and performance analysis tools. In the case of studies related to the Internet, the experiments should not only reflect the wide scale of real scenarios, but also the rich variety of traffic sources, in terms of both protocol typologies and data generation patterns. As a consequence, traffic models can be applied to the generation of synthetic, yet realistic traffic to be injected into networks. In order to be as realistic as possible, traffic models should accurately represent relevant statistical properties of real data flow [1]. For this purpose, we developed a tool, named D-ITG (*Distributed Internet Traffic Generator*) [44] that generates network traffic according to the models proposed for different protocols. We implemented traffic generations for several protocols belonging to layers from 4 to 7 of the TCP/IP stack. The user can simply choose a protocol and is not requested to know its model. In addition, the user can generate a specific traffic pattern by using several random distributions to model the IDT (*Inter Departure Time*) and PS (*Packet Size*) processes. Besides incorporating theoretical models into our generator, we also focused on improving the performance achieved by the sender (in terms of generated data rate) and the receiver (in terms of received data rate). This goal led us to the implementation of two kinds of distributed generators. In the first distributed version, a log-server is used by senders and receivers to maintain the information needed to compute statistics about the experiment made. Data between sender and log-server and data between receiver and log-server can be carried out using either UDP or TCP. In the second distributed version, senders and receivers have been implemented using the MPI (*Message Passing Interface*) library. We move our research towards a distributed architecture able to de-localize logging processes in order to minimize the interference at receiver and sender side and to de-localize logging process with the added value of distributing generation tasks on a cluster of network nodes. Since the logging operations are demanded to the log-server, senders and receivers do not waste time in storing data. By eliminating the interference of logging operations on generation and reception activities, the performance of both senders and receivers proved to be improved. The distributed implementations of D-ITG turn out to be advantageous in a heterogeneous mobile scenario made of devices (e.g. PDAs or PocketPCs) having a very small storage capacity. Indeed, a mobile device sends or receives packets while the logging activity is delegated to a log server having more resources. Due to the nodes' limited resources (RAM, storage capacity, video dimension, etc.) in wireless ad hoc networks, scalability is crucial for network operations. In particular a distributed approach to network communication using collaborative mechanisms permits to reach performance comparable to that of a wired scenario. Another interesting feature of the distributed version of our traffic generator is the possibility to use a unique log server in a wide complex network scenario where a large number of processes (senders and receiver) are present.

The rest of the paper is organized as follows. Section 2 shows D-ITG main topics and describes all the components of the D-ITG platform: ITG-CV (centralized version) component, ITG-LS (distributed version with Log Server on a TCP or UDP channel) component and, finally, ITG-MPI (MPI version of ITG) component. In Section 3 a thorough analysis of our experimentations is illustrated (referenced figures are shown in the Appendix). Section 4 provides some conclusion remarks and presents some interesting issues for research.

2. Distributed Internet Traffic Generator (D-ITG)

After having used some of the existing traffic generators (TG Traffic Generators [2], NetSpec [3], Netperf [4], Packet Shell [5], MGEN [6], Rude/Crude [7], UDPgen [8], Linux Traffic Generator [9], Traffic Generator (TG) [10], Traffic [11], PacGen [12], NTGen [13]) for our network testing and measurement operations, we experimented the lack of the necessary characteristics in a single traffic generator. A complete analysis of related works is presented in [15]. The purpose of our Distributed Internet Traffic Generator is to build up a suite that can be easily used to generate repeatable sets of experiments by using a reliable and realistic mixture of traffic typologies. D-ITG enables to generate many traffic scenarios that could be originated by a typical network test-case made of a large number of users and network devices, as well as by different network topologies. Our generator can simulate (and not emulate) traffic. In our vision, for traffic simulation we mean the reproduction of a "traffic profile" according to theoretical stochastic models. Instead, for traffic emulation we mean the reproduction of a specific protocol (i.e. reproduction of http messages without using a browser). In other words D-ITG generates real flows on the base of theoretical statistical model presented in the scientific literature. D-ITG primary design goals are:

- reproducibility of network experiments: exactly the same experiment can be repeated several times by choosing the same seed value for the packet inter-departure and packet size random processes
- investigation of scaling effects: scalability problems can be investigated by using different network loads or different network configurations
- improvement of generation performance with respect to other traffic generators
- measurement of QoS parameters (delay, jitter, packet loss and throughput)

The generation of realistic traffic patterns can help in understanding protocols and applications of interest in today's Internet. Through the use of our tool, a network administrator can evaluate the performance of a network, locate possible problems and trace guidelines for network planning and real implementation. The outcome of our work was a software tool available to network researchers and designers who need a scientific way to prototype new applications and protocols in a real testbed with realistic traffic. D-ITG defines a platform for traffic flows generation with high generation performance. The D-ITG platform consists of:

– ITG-CV : Centralized Version of Internet Traffic Generator
– ITG-LS : Internet Traffic Generator with Log Server : UDP and TCP implementation
– ITG-MPI : MPI (*Message Passing Interface*) version of Internet Traffic Generator

ITG-LS is able to de-localize logging processes in order to minimize the interference on the receiver and sender processes. ITG-MPI is able to de-localize logging process with the added value of distributing generation tasks on a cluster.

2.1. ITG-CV

ITG-CV sender (*ITGSend*) and ITG-CV receiver (*ITGRecv*) use a signaling channel to exchange information on the generation process. Multiple simultaneous flows are handled by different threads, each of which sends packets using a separate data channel. ITGRecv is informed through the signaling channel about the port where to listen for packets and the ending time of the transmission. Each flow to be generated is basically described by the packet inter-departure process and the packet size process. Both processes are modeled as independent and identically distributed (i.i.d.) series of random variables. The user can choose a distribution for these random variables among the many implemented (constant, uniform, normal, cauchy, pareto and exponential). Thanks to Robert Davies' random number generator library [14], it is very simple to add new random distributions, so as to simulate different kinds of traffic sources. The choice of these distributions is automatically made by ITG-CV in case the user desires to simulate the traffic generated by a specific protocol (generated packets can be filled with a dummy payload). ITG-CV has been planned for the generation of network traffic (ICMP), transport layer traffic (TCP and UDP), several "layer 5-7" traffic (HTTP, FTP, TELNET, SMTP, DNS, VoIP, Video, NNTP, …). One of the features of our ITG-CV is the possibility of specifying the seed value for the packets inter-departure and payload size random processes; in this way, it is possible to repeat exactly a particular realization of these random processes. This feature provides for the reproducibility of network experiments. To collect statistics about the generation process and the network behavior it is necessary to store some information in the sent packets. The payload (both UDP and TCP) of sent packets contains the number of the flow the packet belongs to, a sequence number and the time it was sent. This information is stored in a log file, that is processed at a later stage in order to provide, for example, the average delay (either one-way-delay or round-trip-time) and the loss rate experimented by packets. The logging process interferes with the other activities of the sender and the receiver, limiting the maximum achievable generation rate. In order to reduce this interference, ITG-CV components use a buffer to temporarily store the logging information related to a set of packets. When the buffer is full, its content is stored on the hard disk. The log file is a binary file that can be decoded using our decoder utility.

The traffic generation process is also heavily influenced by the CPU scheduling: several processes (both user and kernel level) can be running on the same PC and this has a bad impact on the quality of the generated flow. Since the real-time support of the operating systems where ITG-CV can be used is not very efficient (due to their scheduling mechanism and the inevitable timer granularity), it was necessary to use a strategy. A variable records the time elapsed since the last packet was sent; when the inter-departure time must be awaited, this variable is updated. If its value is less than inter-departure time the remaining time is awaited, otherwise the inter-departure time is subtracted from the value of this variable and no time is awaited. This strategy guarantees the required bit rate, even in presence of a non real-time operating system. Logging of sent packets, one of the features of our ITG-CV, shows that generated traffic strictly adheres to user's requirements. Another propriety of our generator is the possibility of setting a high priority for the generation process (this feature is available in RUDE/CRUDE generator too). If supported by the operating system, this feature enables to achieve even better performance. We have conducted an experiment in order to compare the performance of ITG-CV to those of other traffic generators. Figure 1 illustrates the generation data rate achieved by various generators (ITG-CV, RUDE, TG 2002 and MGEN) and the expected value when the packet size is fixed (1024 bytes) and the packet rate ranges from 10000 to 30000 pkt/s. Figure 1 shows that even the non-distributed version of D-ITG performs better than the other generators in the sense that it is the closest to the ideal curve. In the sequel we will show that further improvements are gained by using our distributed versions. In this paper we focus on performance analysis of the distributed implementations. More details and a performance evaluation of the centralized version of our generator can be found in [15].

2.2. ITG-LS

The generation of traffic flows that are modeled with two random processes (inter-departure time and payload size) calls for very strict constraints on the sender/receiver activity. The transmission time is imposed by the statistical characterization of the inter-departure time. To adhere to the required inter-departure model the sender must have the necessary resource to send the packet.

Other processes running on the sender machine, or some activity of the sender such as the log management, can influence the generation process limiting the maximum sustainable sending rate. For example, writing the log file causes a high amount of interference, since it requires the use of system calls to store the flow information on a slow device (the hard disk). The distributed components of the D-ITG platform delocalize this auxiliary activity on another machine. One of the ideas that drive the D-ITG platform architecture is the reduction of the file system access rate on the machine that sends or receives packets. ITG-LS exploits the possibility of managing remote information using a fast network more quickly than information stored on an local hard disk device [16,17]. In the last few years, to improve the implementation of activities which require the use of the file system, different approaches, based on the use of the memory of remote machines connected through a fast network, have been proposed [18,19].

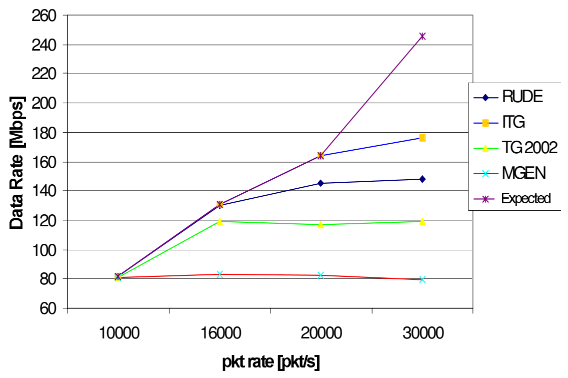

Figure 1. Data Rate Analysis

The new element of ITG-LS, with respect to ITG-CV, is ITGLog (see Figure 2). ITGLog is a "log server", running on a different host, which receives and stores the log information from multiple senders and receivers. The logging activities is handled using a simple signaling protocol. This protocol allows each sender/receiver to register on, and to leave, the log server. The log information can be sent using either a reliable channel (TCP) or an unreliable channel (UDP).

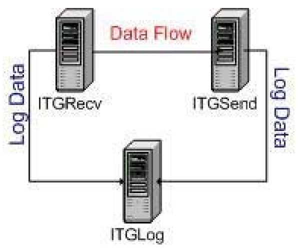

Figure 2. Architecture of D-ITG with *Log Server*

The maximum sustainable sending rate of the ITG-LS implementation of the D-ITG platform is substantially greater than that achieved with ITG-CV. Experimental results that are shown in more detail in section 5. Assuming a loss rate equal to 0, both UDP and TCP implementations of ITG-LS have a sustainable bit rate that is approximately 9% greater than that of ITG-CV. The two implementations of ITG-LS differ if we relax the requirement on the loss rate. If we assume an acceptable loss rate up to 5%, the UDP version of ITG-LS can achieve a maximum bit rate greater than that achieved with the TCP version. However, the TCP implementation of ITG-LS can be used in some scenarios, such as that shown in figure 3, where the use of an unreliable channel for the log packet transfer can lead to some information loss.

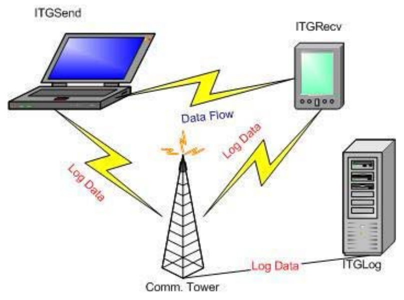

Figure 3. Network Scenario for ITG-LS TCP implementation

2.3. ITG-MPI

The traffic that affects links shared by hosts having different applications running at the same time (such as that of the backbone of the Internet) is the result of the combination of different, statistically independent, flows. It may be possible to simulate it using multiple senders, each of which is associated to one of the component flows. If the different senders run on a single machine, for example if the senders are threads of ITG-CV or ITG-LS, their mutual interference can limit the quality of the generation process. In such scenario only aggregated flows characterized by a low sending rate can be simulated. ITG-MPI addresses this problem using a cluster of workstations to delocalize the generation process of an aggregated traffic flow. Moreover, with ITG-MPI, so as with ITG-LS, it is possible to delocalize the logging process using an appropriate log server. To support the distributed generation, ITG-MPI uses the Message Passing Interface (MPI) [20, 21]. MPI is a well established standard for message passing communication that has been accepted in the current practice of parallel computing for scientific applications. MPI has emerged as the de facto standard for writing portable parallel programs and it includes wide support for collective communication. The MPI interface offers several mechanisms that can be used to exploit specific features possibly provided by the underlying hardware/software transport. Processes in MPI communicate with each other by sending and receiving messages, whether the communication is taking place within the context of an inter-communicator or intra-communicator. Data transfer from one process to another requires operations to be performed by both processes. Thus, for every MPI *send*, there must be a corresponding MPI *receive* performed by the process for which the message is bound. Several works have been performed both on performance [22, 23, 24, 25] and on the optimization of point-to-point and/or collective communications in MPI [26, 27, 28, 29]. Another interesting research field is the use of MPI in network communication [30, 31]. There exist different implementations of the MPI library for different computer architectures [32, 33]. ITG-MPI is based on the LAM [34] implementation of the Message Passing Interface. In order to generate *n* flows, ITG-MPI creates and delocalizes *n* processes on a cluster of workstations (see figure 4). This cluster acts as the sender of the generation experiment. If the log for the sender activity is required (the log for the receiver is always on), to generate *n* traffic flows ITG-MPI creates *n+1* process, the first of which acts as log server. The log information is sent from the senders/receivers processes to the log process using the MPI communication primitives. To optimize this communication, and to limit its interference on the sending/receiving processes, ITG-MPI uses two buffers to store the log information. In such a way, using the asynchronous communication primitives of MPI, ITG-MPI overlaps the log information exchange with the generation process. The overhead due to the use of the MPI communication primitives in sending the logging information can lead to a reduction of the maximum sustainable sending and receiving rate with respect to the TCP/UDP implementation [35, 36]. Experimental results show that the reduction is negligible and an ITG-MPI sender or receiver is comparable to an UDP ITG-LS sender or receiver.

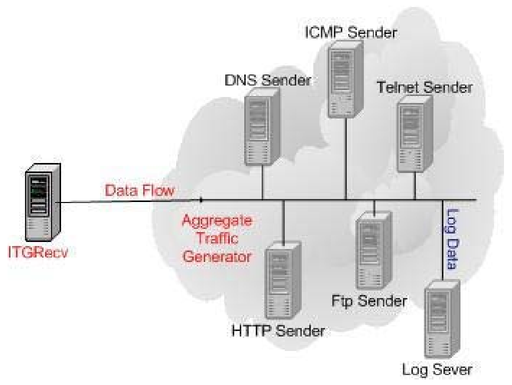

Figure 4. MPI D-ITG architecture

3. Analysis and Performance Evaluation

The main goal of the analysis presented in this section is the determination of an upper bound for the generation rate achieved by D-ITG. We compare the performance of the three different implementations of D-ITG. We focus on the comparison between ITG-LS and ITG-MPI in order to evaluate the possible overhead induced by the use of the MPI library to remotely log information. This evaluation is carried out because we are working on a scenario where processes are able to move on a “Traffic Generator Cluster” in a native way. The evaluation of the performance of the three implementations of D-ITG

presented in sections from 5.2 to 5.3 refers to a constant UDP traffic (constant packet size and constant packet rate) and consists of three steps:
1. given *c*, determining the value of *D* that corresponds to the maximum packet rate achieved while varying *C*
2. given *D* equal to the above value and *C*, determining the value of *c* such that there are no losses
3. given *c* e *D* according to 1 and 2, determining the maximum bit rate such that the losses are negligible while varying *C*

This process has been carried out separately for D-ITG sender and receiver. We considered different configurations (sender and receiver on the same machine or different machines) on various hardware architectures. The results obtained on different architectures are, apart from a scaling factor, congruent. For this reason, we present in the sequel the measures related to a specific implementation. In particular we present the average values on 50 different trials (trails duration is equal to 60 s).

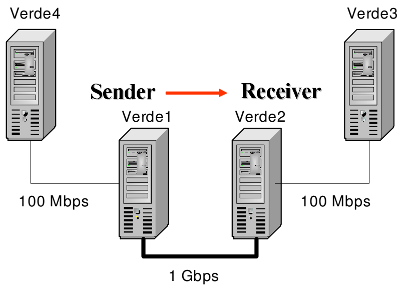

Figure 5. Testbed Architecture

3.1 Testbed architecture

The testbed used to carry out the measurements is depicted in Figure 5. It is a cluster made of four PCs having the Linux Red Hat 8.0 – kernel 2.4.18.14 operating system. The figures indicated in this section are present in the appendix that is downloadable at http://www.grid.unina.it/software/ITG/D-ITGpubblications/UoN-FTDCS04appendix.pdf.

3.2. Analysis of the performance of the Sender

Optimal size of the log buffer: D

Figure 6 illustrates the packet rate achieved by the sender of the three implementations of D-ITG as a function of the size of the log buffer *D* and the required packet rate *C*. It is possible to note for all the implementations that the maximum achieved packet rate grows as the value of *D* grows. The gain obtained while *D* grows decreases and becomes negligible for values greater than 30 for the local implementation and 40 for the other implementations. This means that it is possible to identify an upper bound to the generation capability of the D-ITG senders. The performance improvement can be intuitively explained by considering that the interference of the log operations on the generation of packets decreases as *D* increases, since the log operations are performed less frequently. It is also intuitive that the gain decreases as *D* grows, since further increases on *D* have a negligible impact on the performance of the sender. In order to determine the optimal value of *D*, we can refer to Figure 7 that illustrates the percentage error of generation (packets that the sender is not able to generate) as a function of *D* and the required packet rate. In the case of local implementation, we observe a negligible error rate for *D*=30 and required packet rate close to 28000 pkt/s. The error rate is about 5% for a packet rate close to 30000 pkt/s. We can therefore consider an optimal *D* value of 30, which is related to a maximum achieved packet rate of 28000 pkt/s. In the case of the other implementations, it is easy to draw an optimal *D* value of 40, in correspondence of a maximum achieved packet rate of 30000 pkt/s.

Optimal packet size: c

Figure 8 depicts the error rate (logged_bit_rate/expected_bit_rate) as a function of the packet size for the three implementations of D-ITG. From this figure, we can deduce that the optimal packet size is 1024 bytes for all the implementations. For larger values, the gain in terms of data rate is limited while the error rate blows up (about 30% for *c*=1536 bytes). In this last case we test the behavior with packets length > 1500 bytes (*Maximum Transfer Unit*).

Maximum achieved bit rate: C

Figures 9 and 10 shown the bit rate and the error rate as functions of the packet rate C for all the implementations of D-ITG, given *c* and *D* equal to the optimal values determined in the previous sections. It is possible to deduce that the maximum achieved bit rate is about:

- 218500 kbps for C= 28000 in the case of the local implementation
- 230000 kbps for C= 28000 in the case of the implementation with remote log server (both for UDP and TCP implementation)
- 230000 kbps for C= 28000 in the case of the MPI implementation

The implementation with remote log allows a gain in terms of maximum achieved bit rate equal to 11500 kbps (about 5%) with respect to the local implementation. We can also note that the MPI implementation achieves the same maximum bit rate of the implementation with a remote log server.

3.3 Analysis of the performance of the Receiver

Optimal size of the log buffer: D

As for the performance analysis of the Sender, Figures 11 and 12 illustrates the results of the measures indicated by 1 in Section 5. The resulting trend is very similar to that obtained for the sender: the performance achieved by the receiver improves and the gain decreases as *D* grows. Concerning the evaluation of the optimal size of *D*, from Figures 11 and 12 it is possible to draw an optimal value of 22000 pkt/s with *D*=30 for all the implementations.

Optimal packet size: c

Figure 13 depicts the error rate (logged_bit_rate/expected_bit_rate) as a function of the packet size for the receiver of all the implementations of D-ITG. For values of the packet size up to 1024 bytes, the receiver data rate is equal to that expected. For values of the packet size above 1024 bytes, we can note a considerable packet loss (about 20% in correspondence of *c*=2048). These considerations apply for all the implementations. Therefore, the optimal packet size is again 1024 bytes.

Maximum achieved bit rate: C

Figures 14 and 15 enable to determine the maximum data rate achieved by the D-ITG receiver without losing packets. The maximum data rate supported by ITG-CV is about 163500 kbps. ITG-LS (both udp and tcp) and ITG-MPI implementations of D-ITG present a similar behavior: the maximum supported data rate grow to about 180000 kbps. For the receiver, the implementation with remote log, so as the MPI implementation, allows a gain in term of maximum achieved bit rate equal to 16500 kbps (about 10%) with respect to the local implementation. This gain is greater than that achieved for the sender both in absolute and relative term.

4 Conclusion and Future Work

This work steps from the assumption that currently the Internet traffic generation is an important research task. Indeed, both the tutorials presented at SIGCOMM 2003 [37] and MMNS 2003 [38] have shown that Internet traffic patterns and models are particularly interesting for networking research community. In this work we presented a general framework for traffic generation and performance characterization of our distributed platform named Distributed Internet Traffic Generator (D-ITG). A number of tests were conducted on our real testbed to evaluate important factors such as max data rate, optimal packet size and optimal buffer size. Furthermore this work shows that even the non-distributed version of D-ITG performs better than the other generators. We presented three components of our D-ITG platform: a centralized version, two distributed version with log server (using UDP and TCP channel) and finally an MPI version. We showed how the distributed versions perform better than the centralized version. In addition, we need a distributed version in two kinds of contexts. In the former, the network scenario contains several PDAs. In this case a remote log server (both for sender and receiver phase) is useful because the storage capacity of PDAs is limited. In the latter, in a complex wide network scenario it is useful to have a single log server to coordinate the actions of several processes (senders and receivers). Currently a real network is heterogeneous in terms of access networks, operating systems and end users devices. As far as this last point, we have arranged a realistic scenario where the traffic generation/reception is possible from/to PDAs or Advanced Mobile Phone. Indeed, the introduction of a remote log server is justified not only by the will of increasing performance (by reducing the interference of the logging operations on the generation and reception activities), but also by the lack of available resources on devices such as advanced mobile phones and PDAs. In such heterogeneous scenario, if the sender device is requested to locally log information, the amount of traffic that may be generated is severely limited. In order to carry out a complete characterization of heterogeneous integrated and mobile networks D-ITG has been ported on several different operating systems: Linux, Windows, and embedded operating systems. With respect to this last platform in our testbed we used PDAs running Linux FAMILIAR - kernel 2.4.18 version, and the original source code, with little modifications, has been ported on this destination platform using a cross-compiler version of gcc. Using this implementation is possible to carry out a complete characterization of a real heterogeneous mobile network. Indeed, D-ITG enables performance evaluation of *heterogeneous devices* (Laptop, PC desktop, IPAH, …) over *heterogeneous networks* (Wired LAN, WLAN, GPRS, …). A complete characterization of heterogeneous networks is reported in [39, 40]. Finally we will implement another distributed version of our generator using PVM (*Parallel Virtual Machine*) [41] in order to carry out a comparative study between MPI version and PVM implementation. Finally, currently our testbed allows experiments on a small-scale. We will test the system behavior on a realistic network of a much wider-scale. Thanks to our distributed platform, by using MPI mechanisms and testing Open Mosix [42, 43] environment on a real much wider cluster we will reach a big amount of traffic trunks. D-ITG is currently downloadable and freely available at www.grid.unina.it/software/ITG.

Acknowledgement

This work has been carried out partially under the financial support of the "Ministero dell'Istruzione, dell'Università e della Ricerca (MIUR)" in the framework of the FIRB Project "Middleware for advanced services over large-scale, wired-wireless distributed systems (WEB-MINDS)".

References

[1] M. Zukerman, T. D. Neame and R. G. Addie, "Internet Traffic Modeling and Future Technology Implications", Infocom 2003

[2] http://www.caip.rutgers.edu/~arni/linux/tg1.html
[3] http://www.ittc.ku.edu/netspec/
[4] http://www.netperf.org/
[5] http://playground.sun.com/psh/
[6] http://manimac.itd.nrl.navy.mil/MGEN/
[7] http://www.atm.tut.fi/rude
[8] http://www.fokus.fhg.de/usr/sebastian.zander/private/udpgen
[9] D. Papaleo, S. Salsano, "The Linux Traffic Generator", INFOCOM Department Report 003-004-1999 - University of Rome “La Sapienza”
[10] Paul E. McKenney, Dan Y. Lee, Barbara A. Denny, "Traffic Generator Software" - Release Notes, SRI International and USC/ISI Postel Center for Experimental - January 8, 2002
[11] http://rsandila.ezfish.net/traffic.html
[12] http://pacgen.sourceforge.net/
[13] http://tochna.technion.ac.il/project/NTGen/html/ntgen.htm
[14] Davies, R., Newran02A – a random number generator library on http://webnz.com/robert/nr02doc.htm
[15] A. Pescapè, S. Avallone, G. Ventre “Analysis and experimentation of Internet Traffic Generator”, New2an’04, Next Generation Teletraffic and Wired/Wireless Advanced Networking, pp. 70-75 – ISBN 952-15-1132-X
[16] M. Dahlin, R. Wang, T. E. Anderson, D. A. Patterson. Cooperative caching: Using remote client memory to improve file system performance. In Operating Systems Design and Implementation, pages 267 280, 1994.
[17] G. Ma, A. Reddy. An evaluation of storage systems based on network-attached disks. ICPP’99, pages 278 286, 1999.
[18] M. Flouris, E. P. Markatos. The Network RamDisk: Using remote memory on heterogeneous (NOWs). Cluster Computing, 2(4):281 293, 1999.
[19] Francisco Brasileiro and Walfredo Cirne and Erick Passos and Tatiana Stanchi. Using Remote Memory to Stabilise Data Efficiently on an EXT2 Linux File System. In 20Th SBRC.
[20] MPI Forum, “MPI: A Message-Passing Interface Standard”, International Journal of Supercomputer Applications, Vol.3, No.4, pp 165-414, Aug. 1994.
[21] The MPI Forum, The MPI-2: Extensions to the Message Passing Interface. http://www.mpi-forum.org/docs/mpi-20-html/mpi2-report.html, July, 1997.
[22] M. Lauria and A. Chien, ``MPIFM: High performance MPI on workstation clusters'', Journal of Parallel and Distributed Computing, vol. 40(1), January 1997, pp. 4--18.
[23] H. Franke, C.E. Wu, M. Riviere, P. Pattnik, and M. Snir, ``MPI Programming Environment for IBM SP1/SP2'', Procs. of the International Symposium on Computer Architecture, 1995.
[24] Dickens, P. and G. Thiruvathukal, "Performance Prediction for MPI Programs Executing on Workstation Clusters", In the conference of Parallel and Distributed Programming Techniques and Applications 1998, July 13-16, 1998, Las Vegas, NV.
[25] P.H. Carns, W.B. Ligon III, S. P. McMillan and R.B. Ross, “An Evaluation of Message Passing Implementations on Beowulf Workstations”, Proceedings of the 1999 IEEE Aerospace Conference, March, 1999.
[26] Mohak Shroff and Robert A. van de Geijn, “CollMark: MPI Collective Communication Benchmark”, International Conference on Supercomputing 2000, December 1999.
[27] Nicholas T. Karonis, Bronis R. de Supinski, Ian Foster, William Gropp, Ewing Lusk and John Bresnahan, “Exploiting Hierarchy in Parallel Computer Networks to Optimize Collective Operation Performance”
[28] W. Gropp, E. Lusk, N. Doss, and A. Skjellum, ``A High PErformance, Portable Implementation of the MPI Message Passing Interface Standard'', Tech. Rep. Argonne National Laboratories, (http://www.mcs.anl.gov/mpi/mpicharticle/paper.html)
[29] Amit Karwande, Xin Yuan, and David K. Lowenthal, "CC-MPI: A Compiled Communication Capable MPI Prototype for Ethernet Switched Clusters," ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming (PPoPP), pages 95-106, San Diego, California, June 11-13, 2003.
[30] Venkat, R., Dickens, P., and W. Gropp, "Efficient Communication Across the Internet in Wide-Area MPI", To appear: 2001 Conference on Parallel and Distributed Programming Techniques and Applications, 2001
[31] Thilo Kielmann, Rutger F. H. Hofman, Henri E. Bal, Aske Plaat, Raoul A. F. Bhoedjang, "MagPIe: MPI's Collective Communication Operations for Clustered Wide Area Systems", Proceedings of the Seventh ACM SIGPLAN Symposium on Principles and Practice of Parallel Programming, Atlanta, GA, USA, May 4-6, 1999.
[32] E. Lusk. MPI in 2002: has it been ten years already?. In the Proceedings of IEEE International Conference on Cluster Computing, 2002.
[33] Zhixin Ba, Haichang Zhou, Huai Zhang, Zhenxiao Yang. Performance evaluation of some MPI implementations on workstation clusters. In the Proceedings of the Fourth International Conference/Exhibition on High Performance Computing in the Asia-Pacific Region, 2000.
[34] J. Squyres and A. Lumsdaine and W. George and J. Hagedorn and J. Devaney. The Interoperable Message Passing Interface (IMPI) Extensions to LAM/MPI. MPI Developer's Conference, Ithica, NY, 2000
[35] Rajkumar Vinkat and Philip M. Dickens and William Gropp. Efficient Communication Across the Internet in Wide-Area MPI. In the Proceedings of Parallel and Distributed Processing Techniques and Applications, 2001
[36] Saurab Nog and David Kotz. A Performance Comparison of TCP/IP and MPI on FDDI, Fast Ethernet, and Ethernet. Technical Report TR95-273, 1996.
[37] John Doyle and Walter Willinger, “10 Years of Self-Similar Traffic Research: A Circuitous Route Towards a Theoretical Foundation for the Internet”, Tutorial 2 at SIGCOMM 2003 Conference
[38] Petre Dini, “Internet Multimedia Traffic Patterns”, Tutorial 3 at MMNS 2003 Conference
[39] A. Pescapè, S. Avallone, G. Ventre “Distributed Internet Traffic Generator (D-ITG): analysis and experimentation over heterogeneous networks”, poster at ICNP 2003 (http:// icnp03.cc.gatech.edu/)
[40] A. Pescapè, G. Iannello, G. Ventre, L. Vollero, “Experimental analysis of heterogeneous wireless networks”, WWIC 2004, Wired/Wireless Internet Communications 2004, LNCS Vol. 2957 - pp. 153 - 164, ISBN: 3-540-20954-9
[41] R. Manchek, “Design and Implementation of PVM version 3.0”, TR University of Tennessee, Knoxville, 1994.
[42] OpenMosix homepage, http://www.openmosix.org
[43] A. Barak, O. La'adan: “The MOSIX Multicomputer Operating System for High Performance Cluster Computing”, Journal of Future Generation Computer Systems, Vol. 13, March 1998
[44] www.grid.unina.it/software/ITG