Available online at www.sciencedirect.com
SCIENCE DIRECT®
Performance Evaluation 60 (2005) 359–392
PERFORMANCE EVALUATION
An International Journal
www.elsevier.com/locate/peva

Performance evaluation of an open distributed platform for realistic traffic generation

Stefano Avallone, Donato Emma, Antonio Pescapé*, Giorgio Ventre

Dipartimento di Informatica e Sistemistica, Universitá di Napoli "Federico II", Via Claudio 21, 80125 Napoli, Italy

Available online 16 December 2004

Abstract

Network researchers have dedicated a notable part of their efforts to the area of modeling traffic and to the implementation of efficient traffic generators. We feel that there is a strong demand for traffic generators capable to reproduce realistic traffic patterns according to theoretical models and at the same time with high performance. 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) at packet level and of accurately replicating appropriate stochastic processes for both inter departure time (IDT) and packet size (PS) 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. In this work a complete performance comparison among the centralized version and the two distributed versions of D-ITG is presented.

Keywords: Distributed network traffic generator; Network analysis; Performance evaluation; MPI

1. Introduction

As computer networking has become more ubiquitous, researchers are increasingly focused on optimizing computer networks performance and improving network utilization in terms of throughput and offered

* Corresponding author. Tel.: +39 081 7683825; fax: +39 081 7683816.
E-mail addresses: stavallo@unina.it (S. Avallone), demma@napoli.consorzio-cini.it (D. Emma), pescape@unina.it (A. Pescapé), giorgio@unina.it (G. Ventre).

0166-5316/$ – see front matter © 2004 Elsevier B.V. All rights reserved.
doi:10.1016/j.peva.2004.10.012

delay, jitter and packet loss. This process cannot leave the study of traffic patterns and properties out of consideration. In the last 20 years researchers have been looking for the definition of stochastic processes that could be used as accurate and simple models for traffic generation in packet switched networks and in particular in IP networks. In order to be as realistic as possible, traffic models should accurately represent relevant statistical properties of the original traffic [45]. Modeling the Internet traffic is an important and essential task and we think that traffic theory should be increasingly used to guide the design of the future multi-service and integrated Internet. It is unlikely that we will be able to understand the traffic characteristics, predict network performance (*Quality of Service, Service Level Agreement definition, . . .*), or design dimensioning tools without analytical and rigorous models. The successful evolution of the Internet is tightly coupled to the ability of designing simple and accurate models with the property of reproducibility. Traffic theory suggests us the application of mathematical modeling to explain the relationship between traffic performance and network capacity, traffic demand and experimented performance.

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 a network.

For this purpose, we developed a tool, named *Distributed Internet Traffic Generator* (D-ITG) that generates network traffic according to the models proposed for different protocols. We implemented several protocols belonging to network, transport, and application layers. 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 – at transport layer – by using several random distributions to model the *inter departure time* (IDT) and *packet size* (PS) 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 generator. In the first distributed version, a log server is used by senders and receivers to store the information needed to compute statistics about the experiments made. Both communications sender-log server and receiver-log server can be carried out using either UDP or TCP. In the second distributed version, senders and receivers have been implemented using the *message passing interface* (MPI) library [14]. 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 was improved.

The distributed implementations of D-ITG turn out to be advantageous in a heterogeneous mobile scenario made of devices (e.g. PDAs or Palmtops) having a very small storage capacity. Indeed, a mobile device sends or receives packets while the logging activity is delegated to a remote 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.

Finally, another property 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.

To our knowledge, no similar works are available and we believe that D-ITG shows interesting properties when compared to other traffic generators.

The rest of the paper is organized as follows. Section 2 presents a short overview on widely used theoretical traffic models. In Section 3 we present a complete analysis on closely related works. Section 4 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 D-ITG) component. In Section 5 a thorough analysis of our experimentations is illustrated. Section 6 illustrates some examples of simulations of the Internet traffic using our D-ITG. In this section a tool validation analysis is presented. Section 7 provides some conclusion remarks and presents some interesting issues for future research.

2. Theoretical traffic models

Traffic in Internet results from the uncoordinated actions and operations of a very large population both of users and network devices: it should be therefore described in statistical terms. It is important to be able to describe this traffic easily in a manner which is useful for network engineering and administrator. With respect to the traffic composition, in [22] it is reported a survey on analyses of IP traffic observed at a busy Internet exchange. [22] reveals that the large majority of traffic continues to be generated in TCP connections. TCP counts for about 85% of packets and from 90 to 95% of bytes. Of the remaining traffic most uses UDP – 15% of packets, 5% of bytes – with other protocols like ICMP accounting for the rest. A small but increasing proportion of traffic uses generic route encapsulation (GRE) to create tunnels and this masks the underlying transport protocol. For these reasons in our D-ITG we implemented both UDP and TCP generators with the possibility to use several combinations of IDT and PS probability distributions: D-ITG makes use of a random number generator library which makes available a lot of random variable distributions, through which we can model both IDT and PS.

Following the description in [36] a rough breakdown of TCP traffic (in bytes) according to applications is the following [22]: Web (HTTP) 66%, news (NNTP) 11%, file transfer (FTP) 4%, mail (SMTP) 3%, Napster 3%, etc. A similar breakdown of UDP traffic gives the proportions: Real Audio 21%, DNS 20%, games 18%, unidentified 26%. The relative traffic proportions of TCP and UDP transport protocols has varied little over at least the last years and tend to be the same throughout the Internet. It is important to note that nowadays these percentages are changing since in the last two years we have witnessed an explosion in peer-to-peer traffic. New streaming applications are certainly gaining in popularity but the extra UDP traffic is offset by increases in regular data transfers using TCP. The applications driving these document transfers is evolving, however, with the notable impact over recent years first of the Web and then of peer to peer applications like Napster, Gnutella, Kazaa, WinMX, and Morpheus.

We now introduce some of the models proposed for several application layer protocols, in order to provide an overview on theoretical work at the base of the literature in this field. We start by considering telnet and mentioning the studies conducted by Paxson and Floyd [33,31], which are based on the analysis of the Internet Traffic Archive (ITA) tracks. Such studies show that the arrival process of connection requests can be modeled as an homogeneous Poisson process with a fixed rate, while the distribution of the packet inter-arrivals cannot be considered exponential. An interesting conclusion of other related works is the determination of the empirical cumulative distribution function of the packet size, shown in

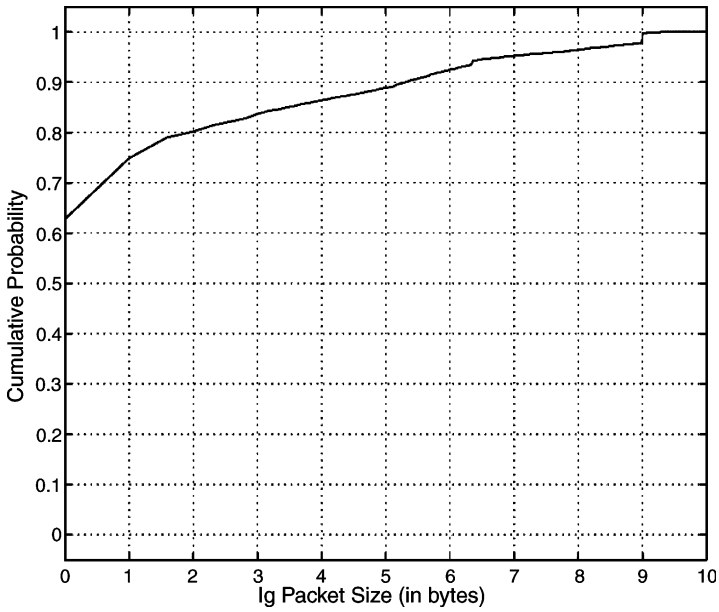


Fig. 1. Telnet packet size distribution.

Fig. 1 [20]. We want to remark that, according to this graphic, 60% of packets has a payload of only one byte.

As for *News Network Transfer Protocol* (NNTP), the proposed models regard the arrival process of connection requests and the total amount of transferred bytes [33,31]. Within a time interval of 60 s, the arrival process can be modeled as a Weibull random variable, while the amount of transferred bytes, which depends on the success of the connection, can be treated using a log 2-normal distribution.

Concerning *Simple Mail Transfer Protocol* (SMTP), Paxson and Floyd observed that, in the determination of the amount of bytes sent, we have to consider a fixed overhead (300 bytes) for all successfully established connections [33,31]. Thus, this overhead is subtracted from the total number of bytes observed and the obtained quantity is modeled again as a log 2-normal random variable.

A great effort has been directed toward the modeling of FTP traffic, since this constitutes a large fraction of WAN traffic. Before delving into the details of the existing models, we point out that an ftp session is made of a sequence of connections, each of which relies on a TCP connection, as indicated in Fig. 2. Typically, ftp connections occur in bursts (a burst is a sequence of connections whose inter-connection times are no longer than 4 s). The arrival process of connections is hard to be modeled since it is influenced by many network-dependent factors, such as bandwidth, congestion level and flow control algorithm [9]. Instead, as for the total amount of bytes transferred during a whole session, a good approximation is that of a log 2-normal random variable. Moreover, the amount of bytes exchanged during a burst can be modeled as a Pareto distribution.

World Wide Web traffic has been obviously the subject of many research studies [1,5,4]. Crovella and Bestavros [5,4] stressed the self-similar nature of this traffic, which came out of their experiments. Such experiments were carried out by using a modified version of MOSAIC, one of the most used earliest

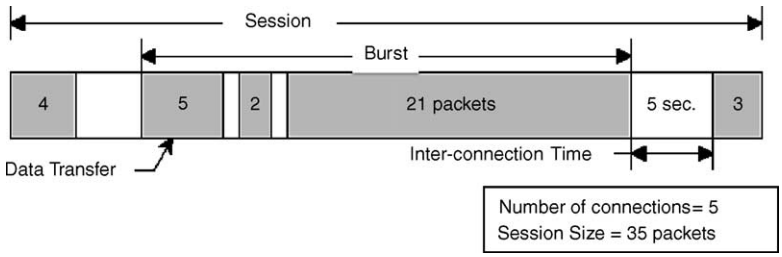


Fig. 2. Example of FTP session.

browsers, which allowed to track statistics like the number of established sessions, the number of file requested, the amount of bytes transferred and so on. Other authors, Arlitt and Williamson, determined the distribution of transferred documents' size [1], paying attention to images (gif, jpeg, bitmap), Postscript, HTML, audio (wav, aiff, aifc) and video (MPEG, QuickTime) files. The results of these analyses indicate that the size of transferred files can be modeled as a Pareto distribution.

Another typology of traffic that is subject to a deep investigation is the *Voice over IP* (VoIP). This is because VoIP traffic will be more and more an important part of Internet traffic. We refer to a careful analysis carried out by Cisco [6], which determines the total bandwidth required in different cases, depending on the encoding algorithm (G.711, G.729, G.723.1, . . .), the number of samples per packet, the usage of *voice activity detection* (VAD) and *real time protocol* (RTP) Header Compression. This analysis provides us with useful information, such as payload size and the number of packets per second. The duration of a call is usually considered exponential.

Lastly, we discuss the case of the video, with reference to MPEG encoding. Garret and Willinger [15] first observed the self-similar nature of variable bit rate MPEG traffic, analyzing an action movie (*Star Wars*) which lasts 2 h. An important result of this test is that the number of bytes per frame can be described using heavy-tailed distributions. In fact, they obtained the curve represented by the consumed bandwidth (shown in Fig. 3) and then calculated its empirical cumulative distribution function. This is well approximated by a mixture of a Gamma and a Pareto random variable. Other works [18,37] aim to exploit the fact that there exist three kinds of MPEG frames (I, P and B) and thus they model separately each one of them.

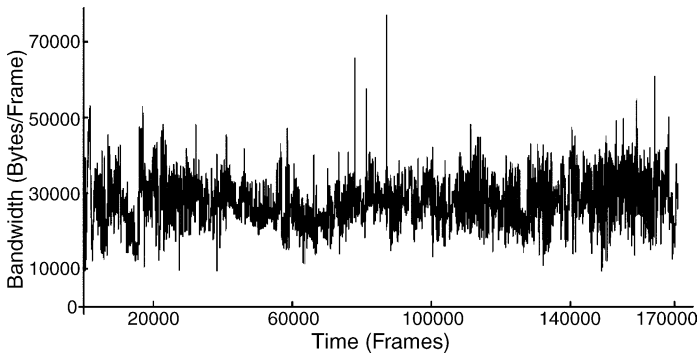


Fig. 3. Time series of entire 2 h VBR video.

3. Related work

This section provides an overview on some of the most widely used traffic generators. TG Traffic Generators [41] runs on Linux, FreeBSD and Solaris SunOS. TG is capable to generate constant, uniform, exponential, on/off UDP or TCP traffic. TG does not offer a rich variety of traffic sources. NetSpec [25] is a traffic generator/emulator that allows the user to define multiple traffic flows from/to multiple PCs. It is capable to emulate TCP, UDP, WWW, FTP, MPEG, VBR and CBR Traffic. Netspec runs on Linux, FreeBSD, Solaris and IRIX. Netperf [24] provides tests for both unidirectional throughput and end-to-end latency. The environments currently measurable by Netperf include TCP and UDP via BSD Sockets, DLPI, Unix Domain Sockets and Fore ATM API. MGEN [21] is both a command line and GUI traffic generator. It runs on Linux, FreeBSD, NetBSD, Solaris, SGI and DEC. MGEN provides programs for sourcing/sinking real-time multicast/unicast UDP/IP traffic flows. The MGEN tools transmit and receive time-stamped, sequence numbered packets. The analysis of the log files can be performed to assess network (or network components) ability to support the given traffic load in terms of packet loss, delay, jitter, etc. Rude/Crude [38] is a command line traffic generator and measurement tool for UDP. RUDE stands for Real-time UDP Data Emitter and CRUDE for Collector for RUDE. Currently these programs can generate and measure only UDP traffic. The operation and configuration might look similar to MGEN. RUDE/CRUDE tools were designed and coded because of the accuracy limitations of MGEN. Rude/crude runs on Linux, Solaris and FreeBSD. UDPgen [43] is a command line UDP traffic generator integrated into the Linux kernel. It aims at maximizing the packet throughput especially for Gigabit Ethernet. To do this, the traffic generator runs completely in the Linux kernel. This allows sending at much higher rates than with an user space program. The toolset also includes a tool which counts UDP packets at the receiver and calculates the packet inter-arrival times. Linux traffic generator [35] (LTG) can generate multiple independent UDP flows with given traffic profiles (i.e. CBR or VBR), with millisecond resolution. LTG works on a common PC with Linux operating system. It is possible to evaluate a set of performance metrics related to throughput, loss and delay. LTG allows generating multiple independent flows of UDP traffic. LTG evaluates the average throughput and enables to log the “instantaneous” throughput. This generator is not publicly available. Traffic Generator (TG) [23] generates and receives one-way packet traffic streams transmitted from the UNIX user level process between traffic source and traffic sink nodes in a network. TG is controlled by a specification language that allows access to different operating modes, protocols, addressing functions, and experimentation with traffic parameters. The specification language allows traffic of several packet lengths and inter-arrival time distributions to be generated. The current implementation supports TCP and UDP transport protocols, with unicast and multicast addressing (UDP only). Traffic [42] generates high volumes of traffic on a network and does not measure throughput or response times. It has a friendly GUI and it runs on Microsoft Win32, FreeBSD and Linux. A limited set of traffic random variables is available. PacGen [28] is an Ethernet IP TCP/UDP packet generating tool for Linux. It generates experimental ARP packets too. This tool enables custom packets with configurable Ethernet, IP, TCP, and UDP layers as well as custom payloads. NTGen [27] (Network Traffic Generator) is a Linux kernel module (supports Linux kernel version 2.4.* and later) that generates network packets. It supports common network protocol packets (ethernet, IP, TCP, UDP, ARP, . . .) and it uses a well-defined meta language (Bison & Lex) for configuring packet generation streams. A user-space application allows the user to configure plans (e.g. streams) of packets generation for the kernel module. Iperf [16] is a tool to measure maximum TCP bandwidth, allowing the tuning of various parameters and UDP characteristics. Iperf reports bandwidth, delay, jitter and, datagram loss. UDPGenerator [44] is a simple UDP traffic

generator. It is a simple unicast traffic generator and per second bandwidth and packet loss data are collected. Mtools [32] is able to send UDP packets to a specific host and measure the transmission time (OWD and RTT) of each packet.

After having used some of the presented traffic generators for our network testing and measurement operations, we experimented the lack of the necessary characteristics in a single traffic generator. Therefore, we decided to implement a traffic generator of our own.

4. Distributed Internet Traffic Generator (D-ITG)

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 models 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 and it is currently downloadable and freely available at http://www.grid.unina.it/software/ITG. 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 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. Table 1 reports all terms and acronyms used in this paper whereas Fig. 4 graphically shows an overview of the D-ITG platform. In the next subsections we present each component of the D-ITG platform.

Table 1

Symbol	Description
D-ITG	Distributed internet traffic generator: distributed platform for traffic generation
ITG-CV	Internet traffic generator: centralized module of D-ITG
ITG-LS	Internet traffic generator with log server: distributed module of D-ITG with log server based either on TCP or UDP channel
ITG-MPI	MPI version of the internet traffic generator: MPI version of D-ITG
ITGSend	Internet traffic generator sender: D-ITG sender
ITGRecv	Internet traffic generator receiver: D-ITG receiver
ITGLog	Internet traffic generator log server: D-ITG log server
OWDM	One way delay meter
RTTM	Rount trip time meter
c	Packet size (byte)
C	Packet per second (pkt/s)
D	Buffer size (packets)

4.1. 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 [7], it is possible 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 "application layer" 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

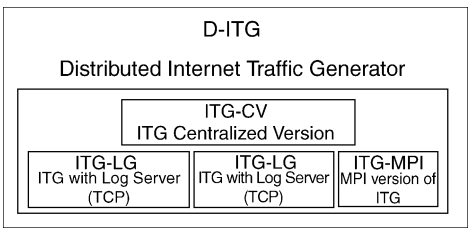


Fig. 4. D-ITG platform architecture.

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), the loss rate experimented by packets, and their jitter. 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 final output is a text file compliant with the format used by MGEN.

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 mechanisms 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 property 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 experiments in order to compare the performance of ITG-CV to those of other traffic generators. Fig. 5 depicts a detailed comparative analysis. These results are related to the generation of 75,000 UDP pkt/s with packet size equal to 1024 bytes and experiment duration equal to 60 s. The experimental testbed is made of two Linux PCs with a Gigabit back-to-back connection. Hardware details are: Intel Pentium IV 2.6 GHz, CPU Cache 512; Controller Ethernet: 3Com Gigabit LOM (3c940); Hard Disk: Maxtor 6Y080L0 (Fast ATA/Enhanced IDE Compatible, Ultra ATA/133 Data Transfer Speed, 2MB Cache Buffer, Quiet Drive Technology, 100% FDB—fluid dynamic bearing-motors). Among the studied

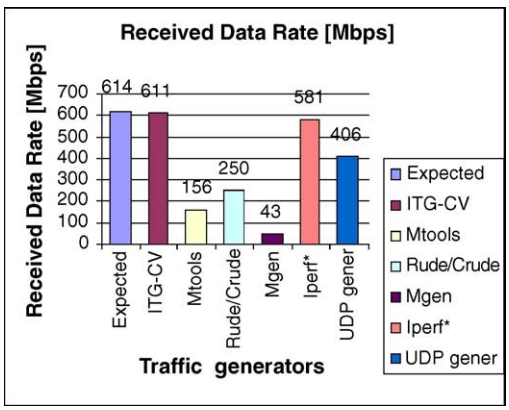


Fig. 5. Data rate analysis.

traffic generators, in this analysis we have taken into account the following traffic generators: Mtools, Rude/Crude, Mgen, Iperf, and finally UDPgenerator. As far as experimental results, ITG-CV shows the best performance. It is important to underline that Iperf works in a different way with respect to ITG-CV. Indeed Iperf does not produce a log file: it provides only an estimation of received and transmitted date rate at the end of the experiment. For a detailed analysis, a comprehensive list of traffic generators can be found at http://www.grid.unina.it/software/ITG/link.html.

Due to data rate comparative analysis, the previous results are carried out using a constant distribution for IDT and PS. In the next section we show some simple example of stochastic synthetic traffic generation. Fig. 5 illustrates 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 expected value. In the sequel we will show that further improvements are gained by using our distributed versions. In this paper we focus only on performance analysis of the distributed implementations. More details and a performance evaluation of the centralized version of our generator can be found in [30].

4.2. 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 [12]. 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 [13,2]. The new element of the ITG-LS, with respect to the ITG-CV, is the ITGLog (see Fig. 6). 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 signaling protocol. This protocol allows each sender/receiver to register on, and to leave, the log server.

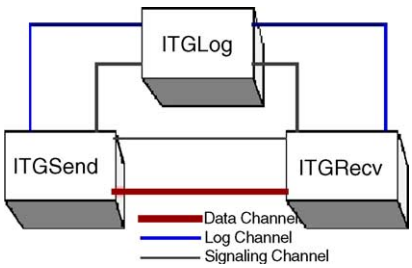


Fig. 6. Architecture of D-ITG with log server.

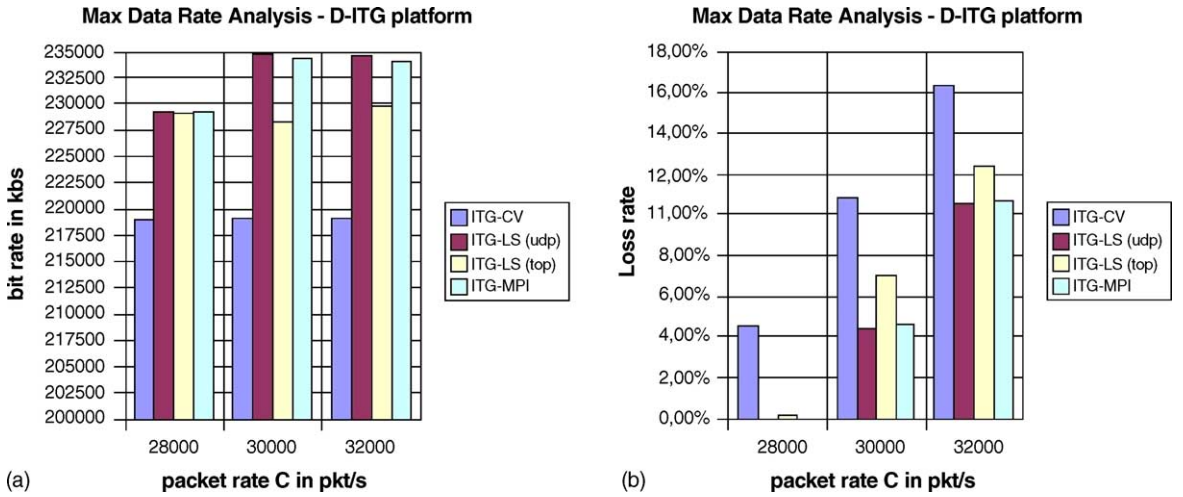


Fig. 7. ITGSend maximum data rate analysis. (a) Maximum data rate for the D-ITG sender; (b) loss rate in the generation process.

The log information can be sent using either a reliable channel (TCP) or an unreliable channel (UDP). The maximum sustainable sending rate of the ITG-LS implementation of the D-ITG platform is substantially greater than that achieved with ITG-CV. Fig. 7(a) and (b) summarize some of the experimental results that are shown in more detail in Section 5. Fig. 11 and Table 2 report a description of the testbed. Note that the testbed used to perform the comparative analysis of the three components of the D-ITG platform is different from the one depicted in Section 4.1. This is reason for the different performance achieved by ITG-CV. Fig. 7(a) shows the bit rate supported by ITGSend for the different component of the D-ITG platform. Fig. 7(b) shows the corresponding loss rate (logged-bit-rate/expected-bit-rate). 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 Fig. 8, where the use of an unreliable channel for the log packet transfer can lead to some information loss.

4.3. 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 might 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

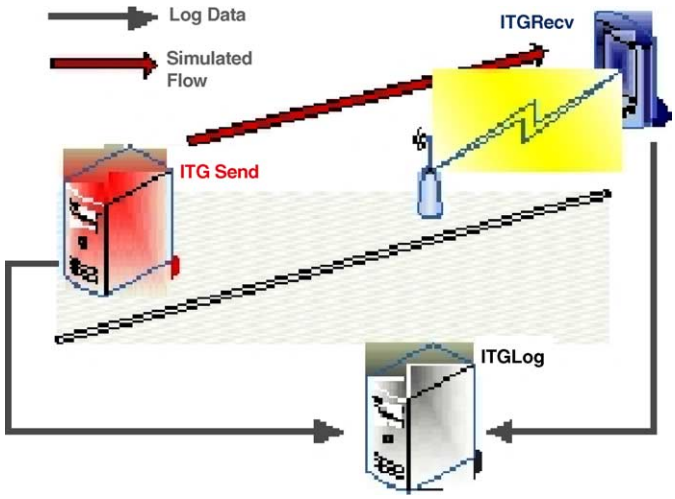


Fig. 8. Network scenario for ITG-LS TCP implementation.

the message passing interface. 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

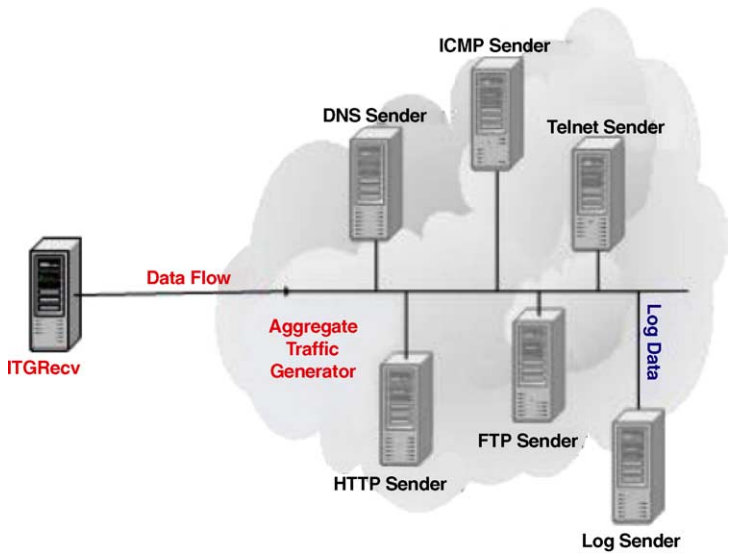


Fig. 9. MPI D-ITG architecture.

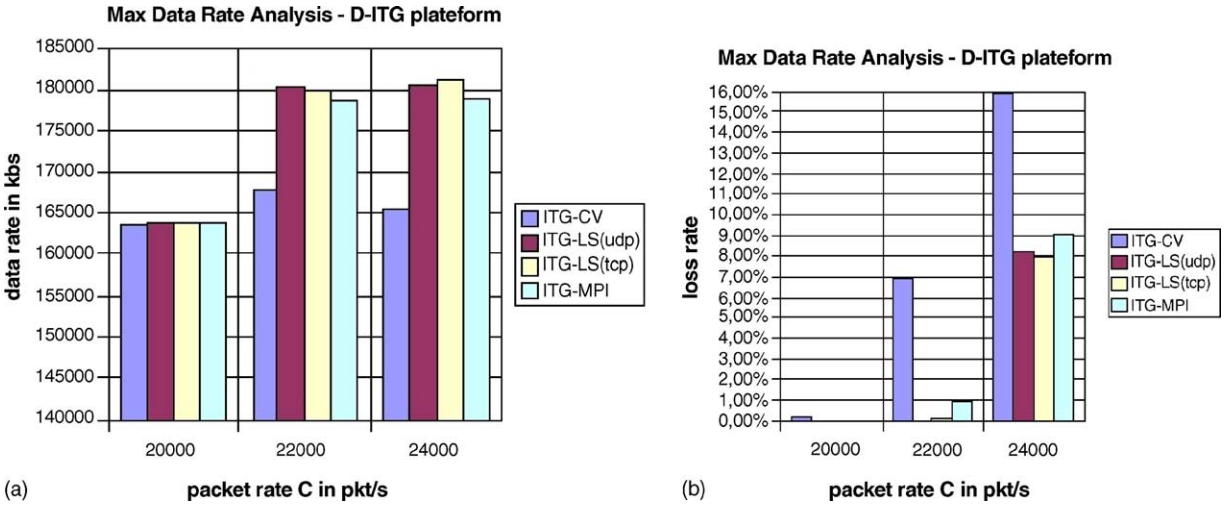


Fig. 10. ITGSend maximum data rate analysis. (a) Maximum data rate for the D-ITG receiver; (b) loss rate in the receiving process.

corresponding MPI receive performed by the process for which the message is bound. Several works have been performed both on performance [19,10] and on the optimization of point-to-point and/or collective communications in MPI [39,17]. There exist different implementations of the MPI library for different computer architectures [3]. ITG-MPI is based on the LAM [40] 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 Fig. 9). This cluster acts as the sender of the generation experiment. If the log of the sender activity is required, to generate n traffic flows ITG-MPI creates $n + 1$ processes, 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 [26]. Experimental results, summarized in Fig. 7(a) and (b) for the sender and in Fig. 10(a) and (b) for the receiver, show that the reduction is negligible and an ITG-MPI sender or receiver is comparable to an UDP ITG-LS sender or receiver.

5. 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 store 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 packets inter-departure time) 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 and 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 20 different trials (trials duration is 60 s). As previously anticipated, this study has been conducted in a different testbed than the one used for the data rate analysis reported in Fig. 5.

5.1. Testbed architecture

The testbed used to carry out the measurements is depicted in Fig. 11. It is a cluster made of four identical PCs having the characteristics shown in Table 2. All PCs have the Linux Red Hat 8.0—kernel 2.4.18.14 Operating System.

5.2. Analysis of the performance of the sender

5.2.1. Optimal size of the log buffer: D

Fig. 25 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

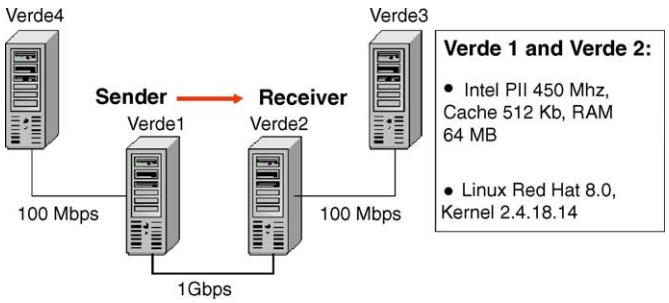


Fig. 11. Testbed architecture.

Table 2
Hardware characteristics

Processor
- Number of CPUs: 2
- Model name: Pentium II
- CPU MHz: 450
- CPU cache: 512KB

PCI bus
- SCSI storage controller: Adaptec AHA-2940U2/U2W/7890/7891 Subsystem: Adaptec 2940U2W SCSI Controller
- Ethernet controller: Intel Corp. 82557/8/9 [Ethernet Pro 100] (rev 04)
- Ethernet controller: Intel Corp. 82542 Gigabit Ethernet (rev 02) (verde1 and verde2)

Hard disk
- Vendor: SEAGATE
- Model: ST39102LW
- Type: Direct-access

Declared performance:
- Average read: 5.4 ms
- Average write: 6.2 ms
- Average latency: 2.99 ms

Synchronous data transfer rate:
- Maximum instantaneous: 40 Mbs (SE mode)
- Maximum instantaneous: 80 Mbs (LVD mode)

Asynchronous data transfer rate:
- Maximum instantaneous: 10.0 Mbs (2 bytes wide)

Multi-segmented cache: 1024KB standard; 4096KB optional

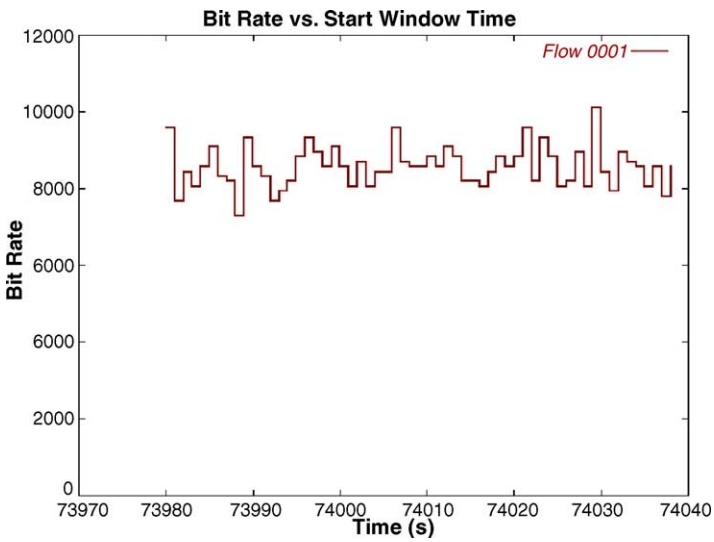
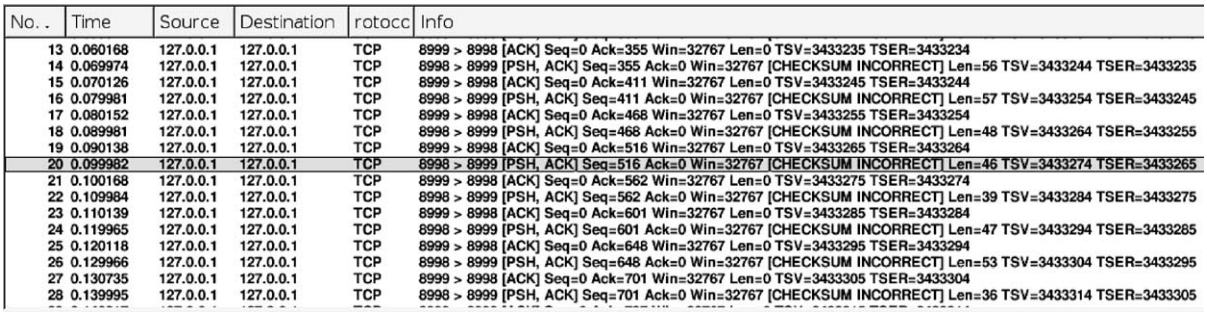


Fig. 12. Resulting plots of TCP generation with IDT = Pareto and PS = constant (window size of 1 s).

```
Num pkts recvd   :  4010
Join delay       :  73979.945 sec
Recv pkt rate    :  66.939 pkt/sec
Recv data rate   :  8.570 kbps
Pkts dropped     :  0
Ave. Tx Delay    :  0.000 sec
Max. Tx Delay    :  0.000 sec
Min. Tx Delay    :  0.000 sec
Delay variation  :  0.000 sec
```

Fig. 13. Output results of TCP generation with IDT = Pareto and PS = constant.



No. .	Time	Source	Destination	rotocc	Info
13	0.060168	127.0.0.1	127.0.0.1	TCP	8999 > 8998 [ACK] Seq=0 Ack=355 Win=32767 Len=0 TSV=3433235 TSER=3433234
14	0.069974	127.0.0.1	127.0.0.1	TCP	8998 > 8999 [PSH, ACK] Seq=355 Ack=0 Win=32767 [CHECKSUM INCORRECT] Len=56 TSV=3433244 TSER=3433235
15	0.070126	127.0.0.1	127.0.0.1	TCP	8999 > 8998 [ACK] Seq=0 Ack=411 Win=32767 Len=0 TSV=3433245 TSER=3433244
16	0.079981	127.0.0.1	127.0.0.1	TCP	8998 > 8999 [PSH, ACK] Seq=411 Ack=0 Win=32767 [CHECKSUM INCORRECT] Len=57 TSV=3433254 TSER=3433245
17	0.080152	127.0.0.1	127.0.0.1	TCP	8999 > 8998 [ACK] Seq=0 Ack=468 Win=32767 Len=0 TSV=3433255 TSER=3433254
18	0.089981	127.0.0.1	127.0.0.1	TCP	8998 > 8999 [PSH, ACK] Seq=468 Ack=0 Win=32767 [CHECKSUM INCORRECT] Len=48 TSV=3433264 TSER=3433255
19	0.090138	127.0.0.1	127.0.0.1	TCP	8999 > 8998 [ACK] Seq=0 Ack=516 Win=32767 Len=0 TSV=3433265 TSER=3433264
20	0.099982	127.0.0.1	127.0.0.1	TCP	8998 > 8999 [PSH, ACK] Seq=516 Ack=0 Win=32767 [CHECKSUM INCORRECT] Len=46 TSV=3433274 TSER=3433265
21	0.100168	127.0.0.1	127.0.0.1	TCP	8999 > 8998 [ACK] Seq=0 Ack=562 Win=32767 Len=0 TSV=3433275 TSER=3433274
22	0.109984	127.0.0.1	127.0.0.1	TCP	8998 > 8999 [PSH, ACK] Seq=562 Ack=0 Win=32767 [CHECKSUM INCORRECT] Len=39 TSV=3433284 TSER=3433275
23	0.110139	127.0.0.1	127.0.0.1	TCP	8999 > 8998 [ACK] Seq=0 Ack=601 Win=32767 Len=0 TSV=3433285 TSER=3433284
24	0.119965	127.0.0.1	127.0.0.1	TCP	8998 > 8999 [PSH, ACK] Seq=601 Ack=0 Win=32767 [CHECKSUM INCORRECT] Len=47 TSV=3433294 TSER=3433285
25	0.120118	127.0.0.1	127.0.0.1	TCP	8999 > 8998 [ACK] Seq=0 Ack=648 Win=32767 Len=0 TSV=3433295 TSER=3433294
26	0.129966	127.0.0.1	127.0.0.1	TCP	8998 > 8999 [PSH, ACK] Seq=648 Ack=0 Win=32767 [CHECKSUM INCORRECT] Len=53 TSV=3433304 TSER=3433295
27	0.130735	127.0.0.1	127.0.0.1	TCP	8999 > 8998 [ACK] Seq=0 Ack=701 Win=32767 Len=0 TSV=3433305 TSER=3433304
28	0.139995	127.0.0.1	127.0.0.1	TCP	8998 > 8999 [PSH, ACK] Seq=701 Ack=0 Win=32767 [CHECKSUM INCORRECT] Len=36 TSV=3433314 TSER=3433305

Fig. 14. Ethereal snapshot.

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 Fig. 26 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 28,000 pkt/s. The error rate is about 5% for a packet rate close to 30,000 pkt/s. We can therefore consider an optimal D value of 30, which is related to a maximum achieved packet rate of 28,000 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 30,000 pkt/s.

```
MCALC : Version 3.3a4
------------------------------------------------------
FLOW: 0001    SOURCE : 127.0.0.1 : 32769    DESTINATION: 127.0.0.1 : 10019
   Num pkts recvd   : 1000
   Join delay       : 57989.328 sec
   Recv  pkt  rate  : 100.017 pkt/sec
   Recv data rate   : 38.254 kbps
   Pkts dropped     : 0
```

Fig. 15. Mcalc output.

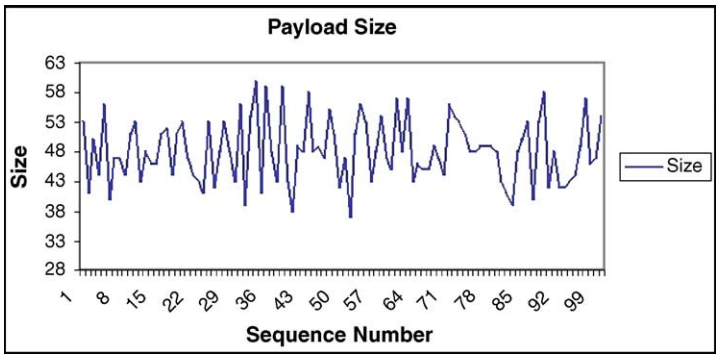


Fig. 16. Poisson packet size trend (D-ITG output).

5.2.2. Optimal packet size: c

Fig. 27 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 greater than the *maximum transfer unit*.

5.2.3. Maximum achieved bit rate: C

Figs. 28 and 29 report 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:

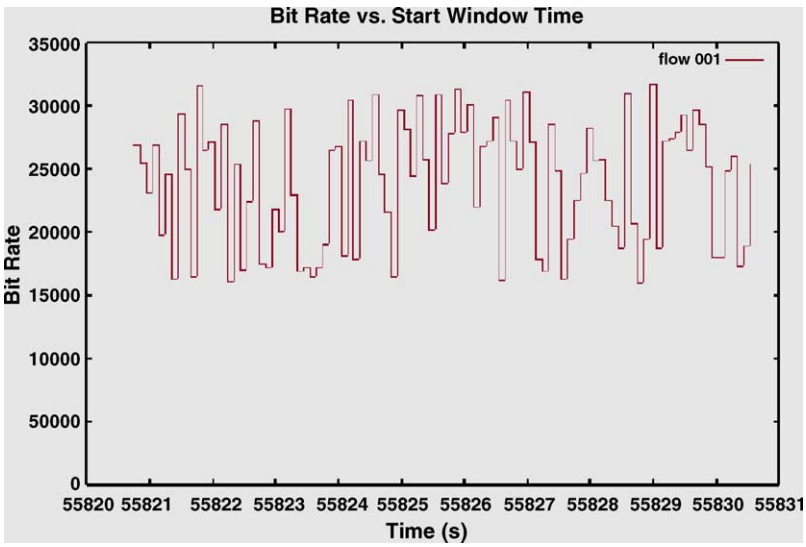


Fig. 17. TCP traffic using uniformly distributed PS.

```
MCALC : Version 3.3a4
---------------------------------------
FLOW: 0001      SOURCE : 127.0.0.1:32769 DE
    Num pkts recvd    :   1000
    Join delay        :   55820.734 sec
    Recv pkt rate     :   100.017 pkt/sec
    Recv data rate    :   240.279 kbps
    Pkts dropped      :   0
    Ave. Tx Delay     :   0.000 sec
    Max. Tx Delay     :   0.000 sec
    Min. Tx Delay     :   0.000 sec
    Delay variation   :   0.000 sec
```

Fig. 18. TCP mcalc snapshot.

- 218,500 kbps for $C = 28{,}000$ in the case of local implementation;
- 230,000 kbps for $C = 28{,}000$ in the case of the implementation with remote log server (both for UDP and TCP implementation);
- 230,000 kbps for $C = 28{,}000$ in the case of the MPI implementation.

The implementation with remote log server allows a gain in terms of maximum achieved bit rate equal to 11,500 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.

5.3. Analysis of the performance of the receiver

5.3.1. Optimal size of the log buffer: D

As for the performance analysis of the sender, Figs. 30 and 31 illustrate the results of the measures performed to determine the optimal size of D. 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 Figs. 30 and 31 it is possible to draw an optimal value of 22,000 pkt/s with $D = 30$ for all the implementations.

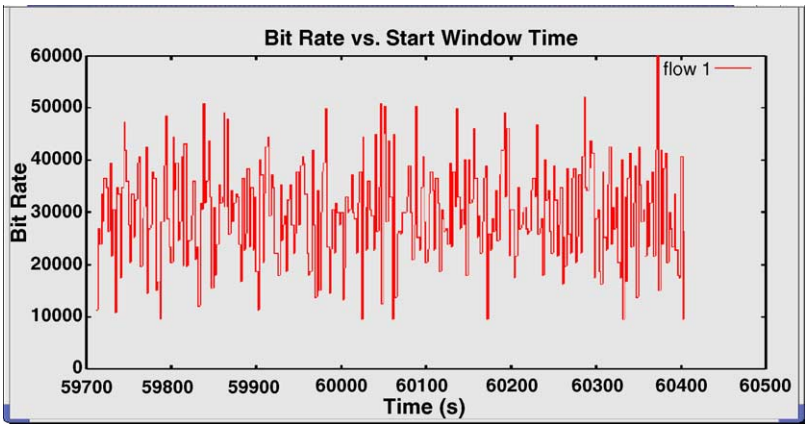


Fig. 19. UDP video traffic (D-ITG output).

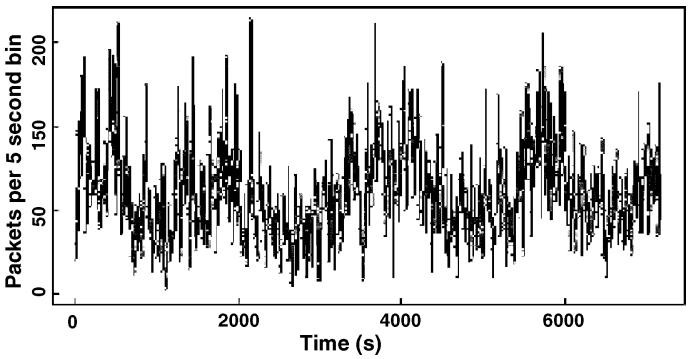


Fig. 20. Two hours of telnet traffic.

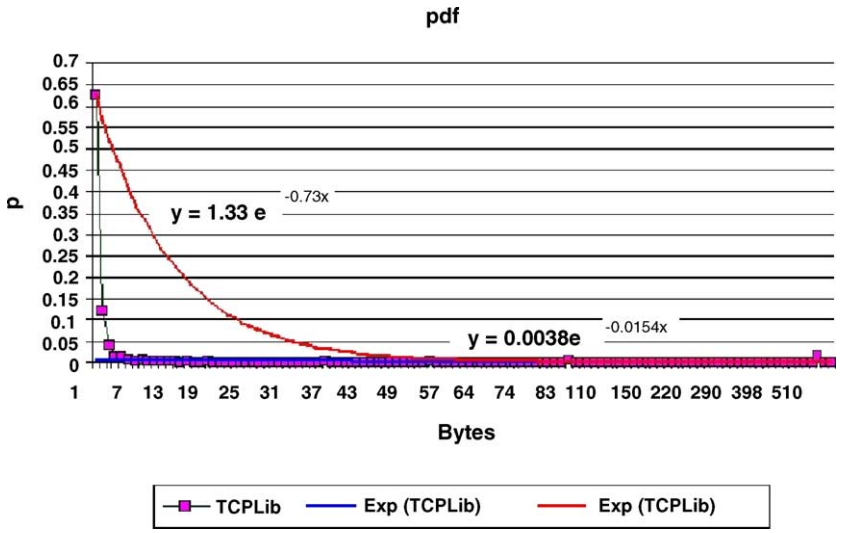


Fig. 21. TCPLib.

5.3.2. *Optimal packet size: c*

Fig. 32 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 the expected one. For values of the packet size above 1024 bytes, we can note a considerable packet loss (about 20% in correspondence of $c = 2048$). These con-

```
MCALC : PACKET RECEPTION STATISTICS
MCALC : Total packets received     : 740 pkts
MCALC : Total recv packet rate     : 74.248 pkt/sec
MCALC : Total recv date rate       : 1.181 kbps
MCALC : Est. num pkts dropped      : 0 pkts
```

Fig. 22. Mcalc telnet output.

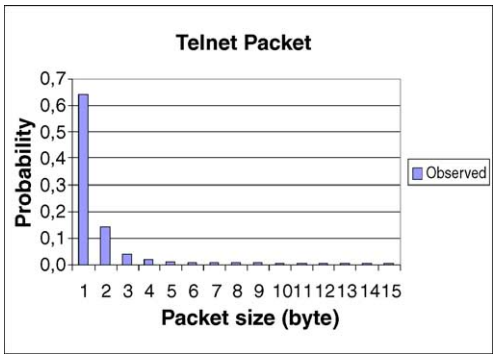


Fig. 23. D-ITG telnet PS.

siderations are true for all the implementations. Therefore, the optimal packet size is again 1024 bytes.

5.3.3. *Maximum achieved bit rate: C*

Figs. 33 and 34 enable to determine the maximum data rate achieved by the D-ITG receiver without loosing packets. The maximum data rate supported by ITG-CV is about 163,500 kbps. ITG-LS (both UDP and TCP) and ITG-MPI present a similar behavior: the maximum supported data rate grow up to about

```
MCALC: PACKET RECEPTION STATISTICS
MCALC: Total packets received  :   1000 pkts
MCALC: Total recv packet rate  :     99.978 pkt/sec
MCALC: Total recv data rate    :     70.455 kbps
MCALC: Est. num pkts dropped   :      0 pkts
```
(a)

```
MCALC: PACKET RECEPTION STATISTICS
MCALC: Total packets received  :   1000 pkts
MCALC: Total recv packet rate  :    100.017 pkt/sec
MCALC: Total recv data rate    :     48.056 kbps
MCALC: Est. num pkts dropped   :      0 pkts
```
(b)

```
MCALC: PACKET RECEPTION STATISTICS
MCALC: Total packets received  :    500 pkts
MCALC: Total recv packet rate  :     49.998 pkt/sec
MCALC: Total recv data rate    :     67.332 kbps
MCALC: Est. num pkts dropped   :      0 pkts
```
(c)

```
MCALC: PACKET RECEPTION STATISTICS
MCALC: Total packets received  :    500 pkts
MCALC: Total recv packet rate  :     49.898 pkt/sec
MCALC: Total recv data rate    :     11.222 kbps
MCALC: Est. num pkts dropped   :      0 pkts
```
(d)

```
MCALC: PACKET RECEPTION STATISTICS
MCALC: Total packets received  :    330 pkts
MCALC: Total recv packet rate  :     32.990 pkt/sec
MCALC: Total recv data rate    :     10.060 kbps
MCALC: Est. num pkts dropped   :      0 pkts
```
(e)

```
MCALC: PACKET RECEPTION STATISTICS
MCALC: Total packets received  :    260 pkts
MCALC: Total recv packet rate  :     26.002 pkt/sec
MCALC: Total recv data rate    :      7.885 kbps
MCALC: Est. num pkts dropped   :      0 pkts
```
(f)

Fig. 24. Mcalc VoIP output. (a) Mcalc VoIP output (Codec G.711 with one voice sample per packet and without VAD); (b) mcalc VoIP output (Codec G.711 with one voice sample per packet and with VAD); (c) mcalc VoIP output (Codec G.711 with two voice samples per packet and without VAD); (d) mcalc VoIP output (Codec G.729 with two voice samples per packet and without VAD); (e) mcalc VoIP output (Codec G.729 with three voice samples per packet and without VAD); (f) mcalc VoIP output (Codec G.723.1 with one voice sample per packet and without VAD).

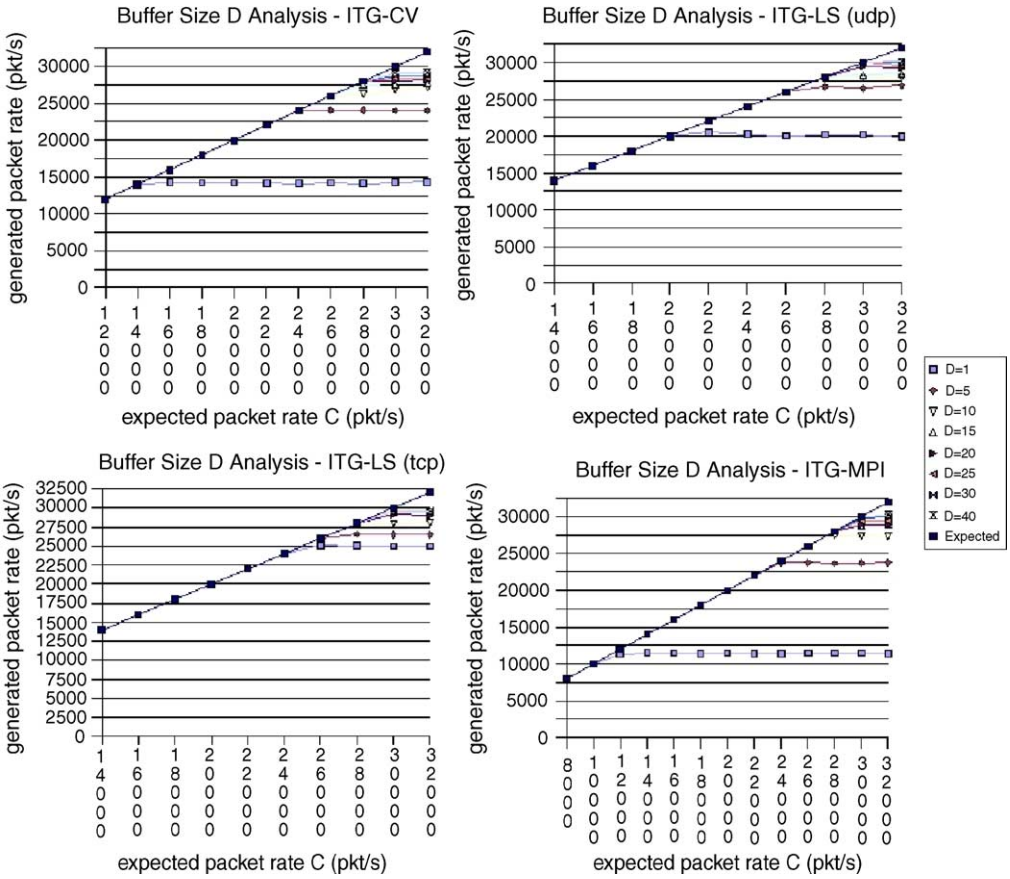


Fig. 25. Sender buffer size analysis—data rate ($c = 512$; $t = 60$ s): this figure depicts the packet rate achieved by the different implementations of ITGSend as a function of the required packet rate C, for different values of the log buffer size D. The achieved maximum packet rate grows as D increases. For every implementation, it is possible to determine an upper bound for the maximum packet rate. ITG-CV achieves the smallest upper bound (about 28,000 pkt/s when D is at least 30) while the other three implementations perform better (about 30,000 pkt/s when D is at least 40).

180,000 kbps. For the receiver, the implementation with remote log, so as the MPI implementation, allows a gain in terms of maximum achieved bit rate equal to 16,500 kbps (about 10%) with respect to the local implementation. This gain is greater than that achieved for the sender both in absolute and relative terms.

6. Simulating internet traffic: analysis and experimentation

This section reports some practical examples of D-ITG with comments on the related results. In particular we show:

- TCP traffic generation:
  - IDT = Pareto distributed, PS = constant.
  - IDT = constant, PS = Poisson distributed.

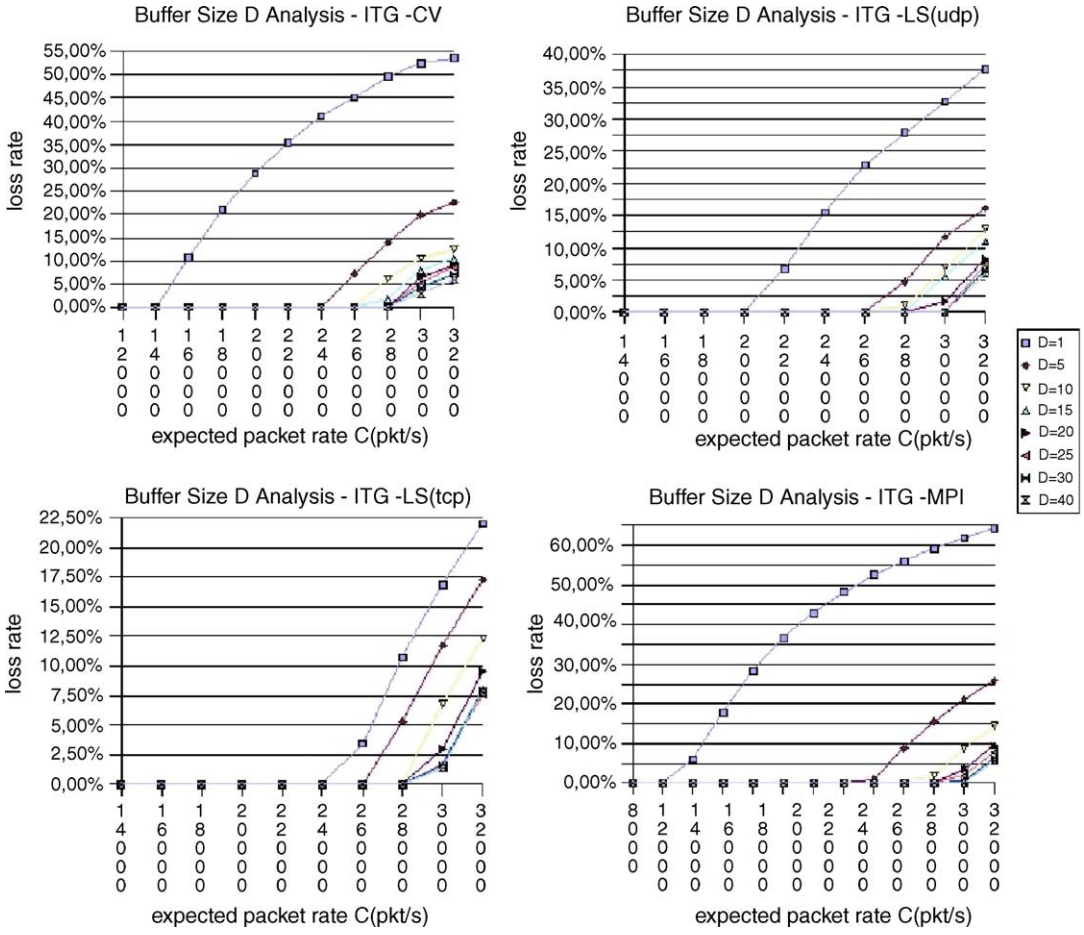


Fig. 26. Sender buffer analysis ($c = 512$; $t = 60$ s): this figure shows the percentage of packets the sender is not able to generate as a function of C, for different values of D. ITG-LS and ITG-MPI exhibit a similar behavior (a percentage error smaller than 2% for $C < 30{,}000$ and $D = 40$) and perform better than ITG-CV. In particular, it is possible to note that ITG-MPI and the UDP version of ITG-LS perform slightly better than the TCP version of ITG-LS (a percentage error almost null for $C = 30{,}000$ and $D = 40$ against the percentage error of 1.5% achieved by the TCP version of ITG-Log).

  - IDT = constant, PS = uniformly distributed.
- UDP traffic generation
  - *Variable bit rate* (VBR) video traffic: IDT = constant, PS = normally distributed.
- Telnet traffic generation
- *Voice over IP* (VoIP) traffic generation.

The former part shows TCP and UDP traffic generation whereas the latter one shows the application layer traffic generation. The first example reports the generation of TCP traffic. The payload size of all packets is constant and equal to 16 bytes. The flow lasts 60 s and the packet generation process is a Pareto process, characterized by *shape* equal to 3 and *scale* equal to 10. We want to calculate now the expected average bit rate, in order to verify the accuracy of our D-ITG. First, we note that *mcalc* and *ez* utilities simply

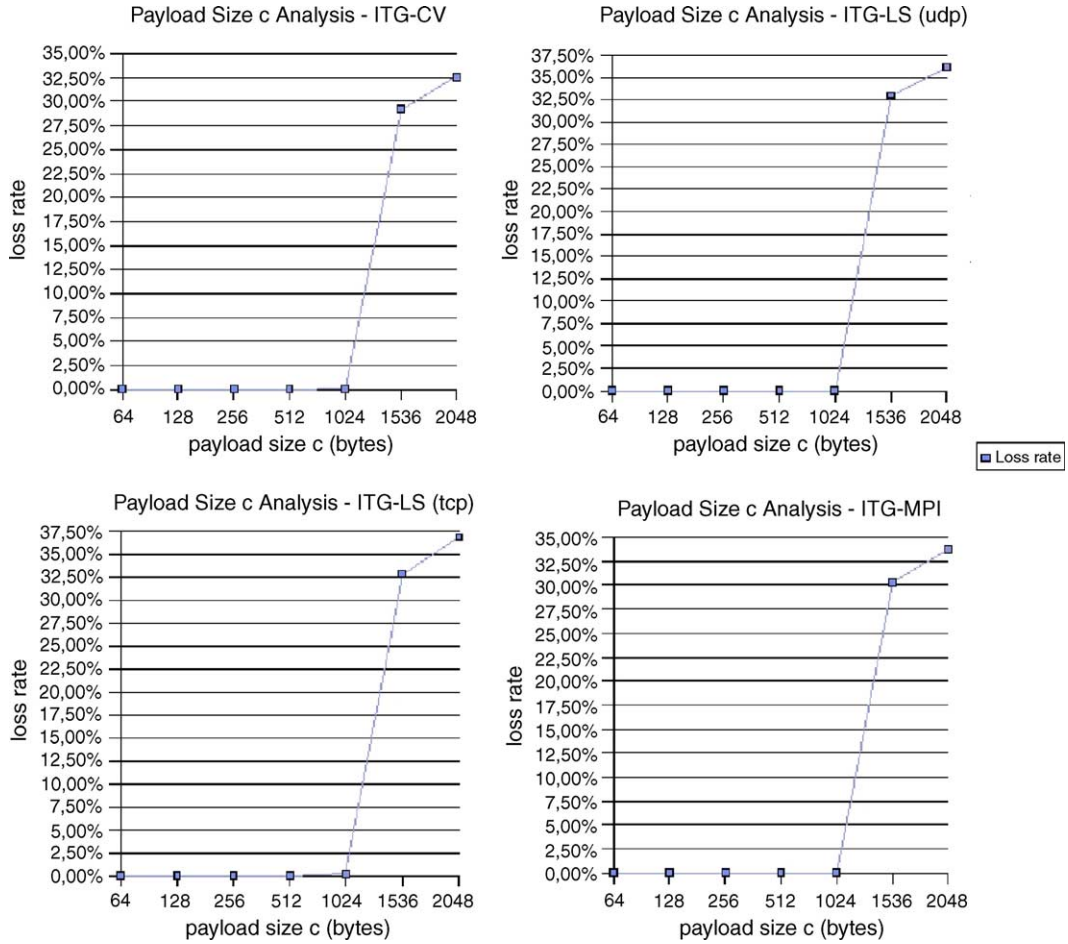


Fig. 27. Sender packet size analysis ($C = 26{,}000$ for ITG, $C = 28{,}000$ for ITG-LS and ITG-MPI; $t = 60$ s): this figure illustrates the percentage error as a function of c; for all the implementations of D-ITG, the generation error is null for $c < 1024$. For values above this threshold, we can observe a considerable packet loss.

consider the payload size of packets (and not their full size) in determining the average bit rate, and we will do so as well. The expected value of a Pareto random variable with the previously defined parameters is 15: this means that the average time interval between the departure of two consecutive packets is 15 ms. As indicated the payload size is equal to 16 bytes (=128 bits). Therefore the average bit rate is equal to the ratio of 128 bits to 15 ms, which yields 8533 Kbps. The resulting plots and mcalc output derived from the log file of sender are shown in Figs. 12 and 13 and confirm our expectations. Notice that bit rate plots (Fig. 12) need the specification of a window size that is the time interval on which the bit rate must be computed. Moreover, notice that all the parameters from mcalc output are related to the generated traffic, even though they are addressed as "received" (this is due to the fact that MGEN does not store sent packets and therefore analyzes only receiver's log files). In the second example we show a TCP traffic generation with a constant IDT (which results in 100 packets per second) and a PS that follows a Poisson

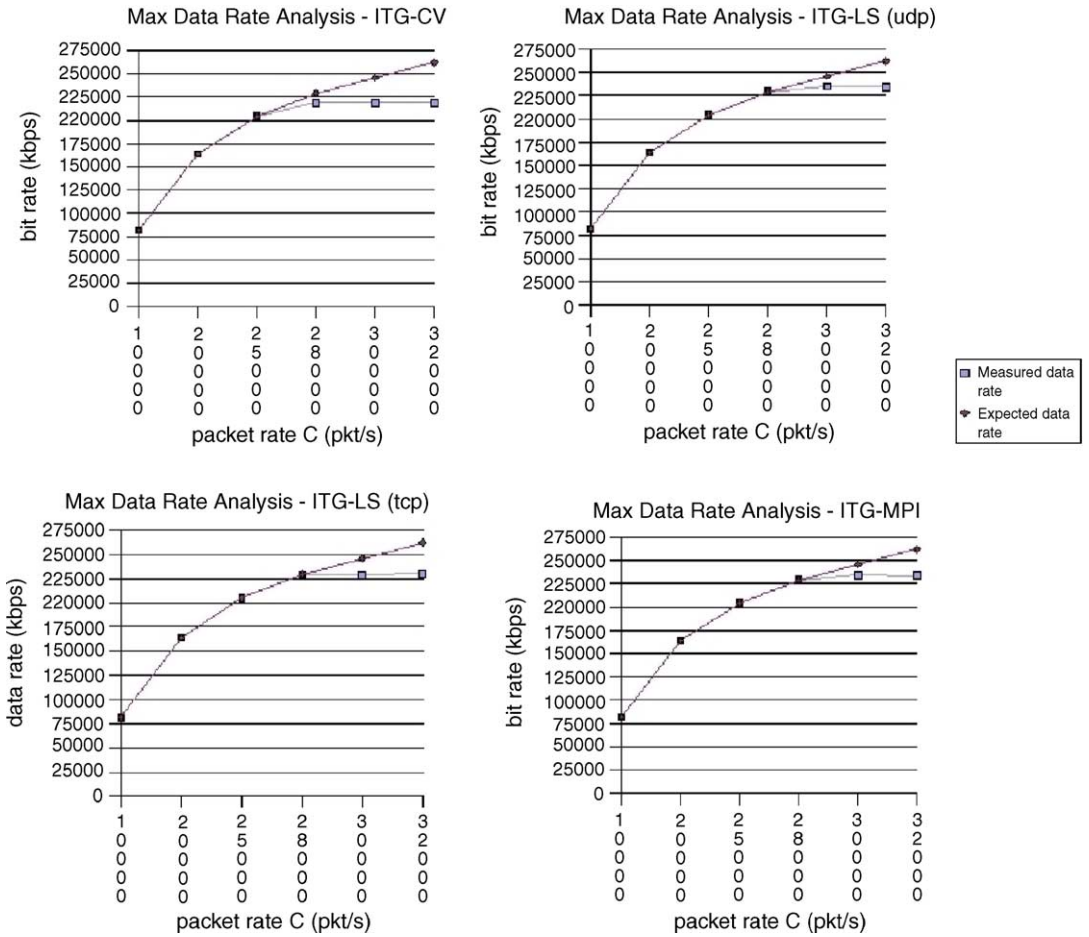


Fig. 28. Sender max data rate analysis ($c = 1024$; $D = 40$): this figure depicts the generated data rate as a function of C, while D and c are equal to their optimal values. For all the implementations of D-ITG, it is possible to determine an upper bound to the generation rate. The smallest upper bound is achieved by ITG-CV and is about 218,500 Kbps. The other implementations perform similarly, having a maximum data rate equal to about 230,000 Kbps (that is, 5% greater).

process with an average value of 48 bytes. Fig. 14 reports the snapshot of a traffic analyzer (Ethereal). This figure allows checking that D-ITG correctly creates TCP packets. The expected data rate is equal to 38 Kbps (48 bytes $\times$ 100 packets $\times$ 8). This value is confirmed by the results shown in Fig. 15, which reports the statistics of the D-ITG generation obtained using mcalc. Furthermore, stemming from the theory, the deviation is equal to 48. Using the real generated values we have a deviation equal to 48.7 bytes and an average value equal to 47,761 bytes with a relative error respect to theoretical value equal to 1.4%. Fig. 16 reports the size of the first 100 packets. In the third example we show a TCP traffic generation with a constant IDT (which results in 100 packets per second) and a uniformly distributed PS between 200 and 400 bytes. The expected average bit rate is 240 Kbps whereas the value achieved by D-ITG is 240.279 Kbps. Fig. 17 reports D-ITG output whereas in the Fig. 18 the mcalc output is sketched. In the fourth example we illustrate the accuracy of D-ITG. Indeed we show how D-ITG is able

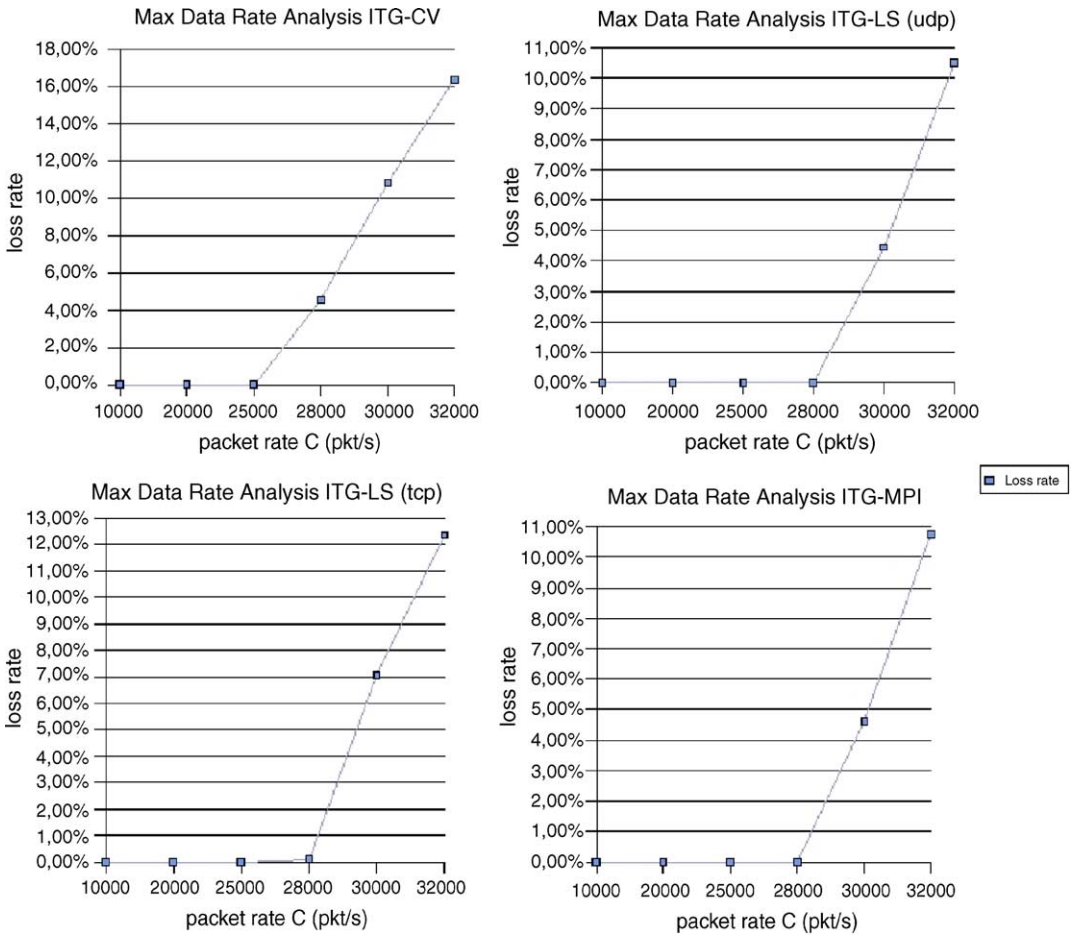


Fig. 29. Sender max data rate analysis ($c = 1024$; $D = 40$): this figure shows the percentage generation error for the three implementations of D-ITG. We can observe that only ITG-MPI and the UDP version of ITG-LS achieve a generation error smaller than 5% when the expected packet rate is 300,000 pkt/s (which corresponds to an actual data rate of 245,760 kbps).

to reproduce theoretical traffic model following the expected results. In particular using the results of Fig. 19 we can perform a comparison between a real traffic trace (Fig. 3) and D-ITG simulated traffic. The compared traffic trace is a real VBR (Variable Bit Rate) video having the following characteristics: 2 h long, 24 frames per second, frame dimensions normally distributed with $m = 27{,}791$ bytes and $s = 6254$ bytes [15]. D-ITG generates UDP traffic with one frame per packet, 12 min of traffic generation, IDT equal to 24 packets per second and PS normally distribution with $m = 27{,}791$ bytes and $s = 6254$ bytes. Comparing the trend in the two figures, we have the opportunity of checking the accuracy of D-ITG. In the next example we show how D-ITG is able to generate application level traffic. In particular we show the generation of Telnet traffic and VoIP traffic. The Telnet traffic model considers a Pareto distributed IDT with shape b = 0.90 and scale a 1, whereas PS follows the TCPLib. In Fig. 20 a 2 h Telnet traffic trace is sketched and in Fig. 1 the cumulative probability function of Telnet packet size is shown. Furthermore,

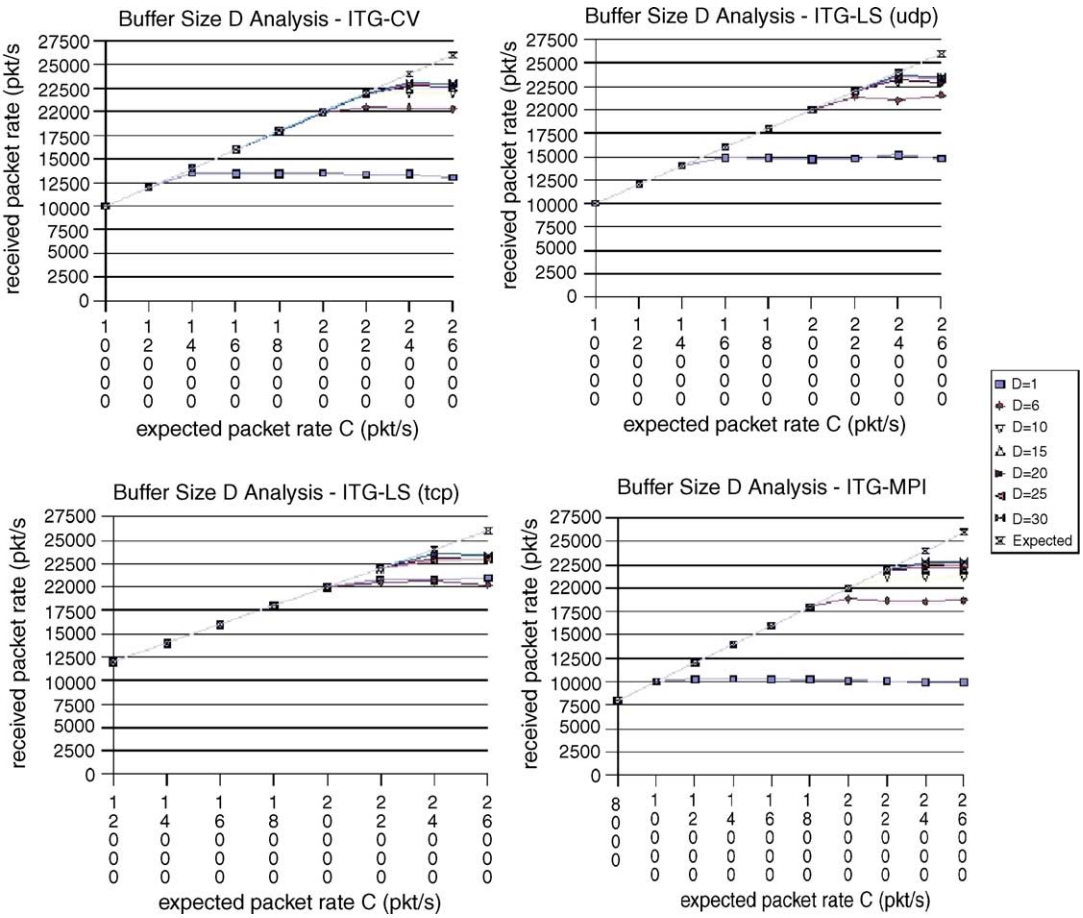


Fig. 30. Receiver buffer size analysis ($c = 512$; $t = 60$ s): this figure represents the number of packets received by the different implementations of ITGRecv as a function of C, while increasing D. The maximum packet rate increases as D grows. For all the implementations, it is possible to calculate an upper bound. This value is similar for all the component of D-ITG and is equal to 22,000 pkt/s.

Fig. 21 depicts the TCPlib trend. This figure reports the comparison among TCPLib distribution and two exponential trend lines: looking at this representation, we can show that the Telnet PS does not follow an exponential distribution. In particular there is a substantial difference especially for the first bytes. As far as D-ITG Telnet traffic generation, we consider a 10 s generation interval. In order to verify the PS distribution we analyze log files using the mcalc utility (Fig. 22): the data rate is equal to 1.181 Kbps (i.e. 147.6 byte per second). In order to calculate the average value of packet dimension we calculate the ratio between the data rate and the packet rate (147.6 bytes per second divided per 74.248 packets per second): the result is 1.98 bytes per packet. In the theoretical model the bytes-per-packet average value is equal to 1 byte in the 62.8% of the total packets whereas it is equal to 2 bytes in the 12.1% of the same total: using D-ITG we can conclude that the theoretical value is respected. Finally, in Fig. 23 the D-ITG Telnet PS is reported.

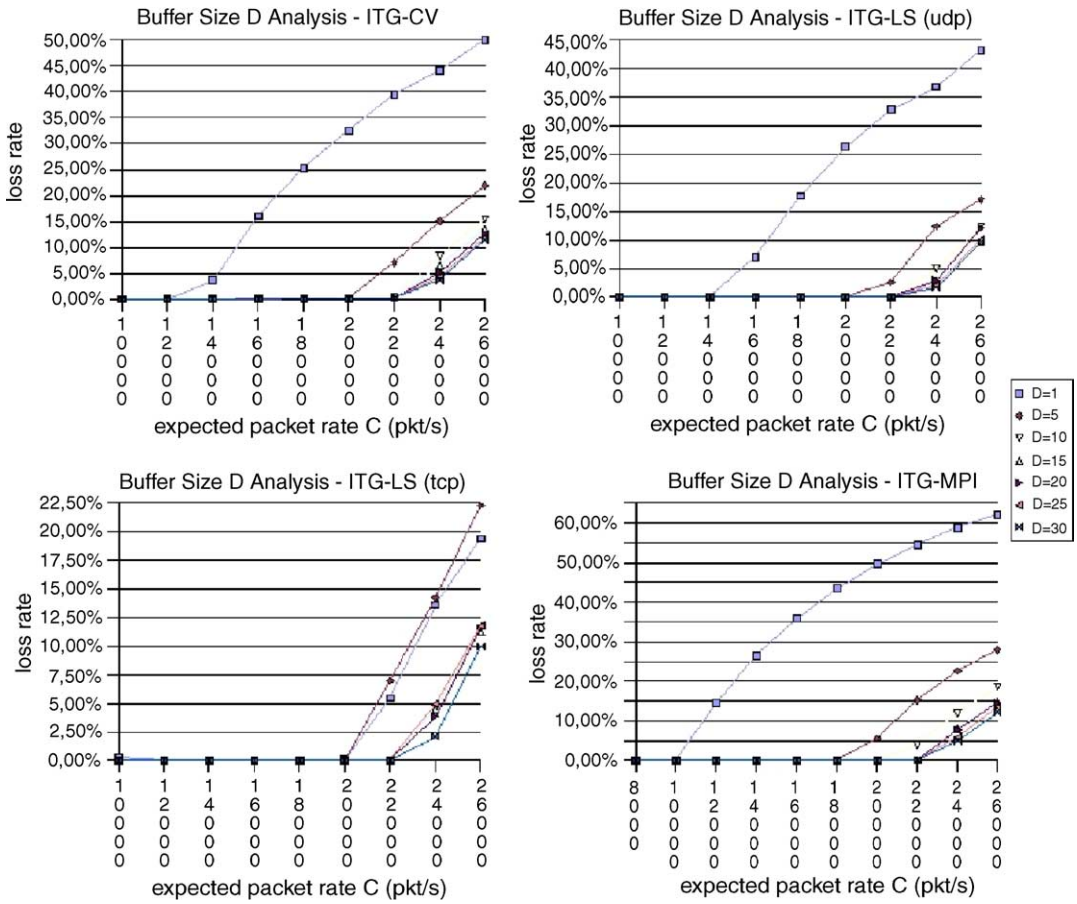


Fig. 31. Receiver buffer analysis ($c = 512$; $t = 60$ s): this figure illustrates the percentage of lost packets as a function of C for different values of D. The different implementations of D-ITG exhibit a similar behavior for $C < 22{,}000$.

Finally, in the last example we show how D-ITG is able to generate different sessions of VoIP traffic. These sessions are different in terms of codecs and for using the voice activity detection (VAD) in conjunction with header compression. Indeed, in the generation phase D-ITG is able to specify the used codec, number of samples per packet, VAD option and finally real time protocol (RTP) header compression. In order to calculate the number of bytes for the payload we use the relation Payload = Codec-Rate × Frame-Time × VAD, where [Codec-Rate] = Kbps and [Frame-Time] = second, while VAD is a pure number. In our real implementation we use an average reduction of the payload equal to 35%. The following figures represent several examples where different combinations of input values are present. The generation interval is equal to 10 s in all trials. Fig. 24(a) reports the mcalc output related to a VoIP traffic generation with Codec G.711 with one voice sample per packet and without VAD. In this case in the theoretical model we have 100 packets per second and the payload equal to 80 bytes. Taking into the account 8 bytes of RTP header the expected data rate is $88 \times 100 \times 8 = 70.4$ Kbps that

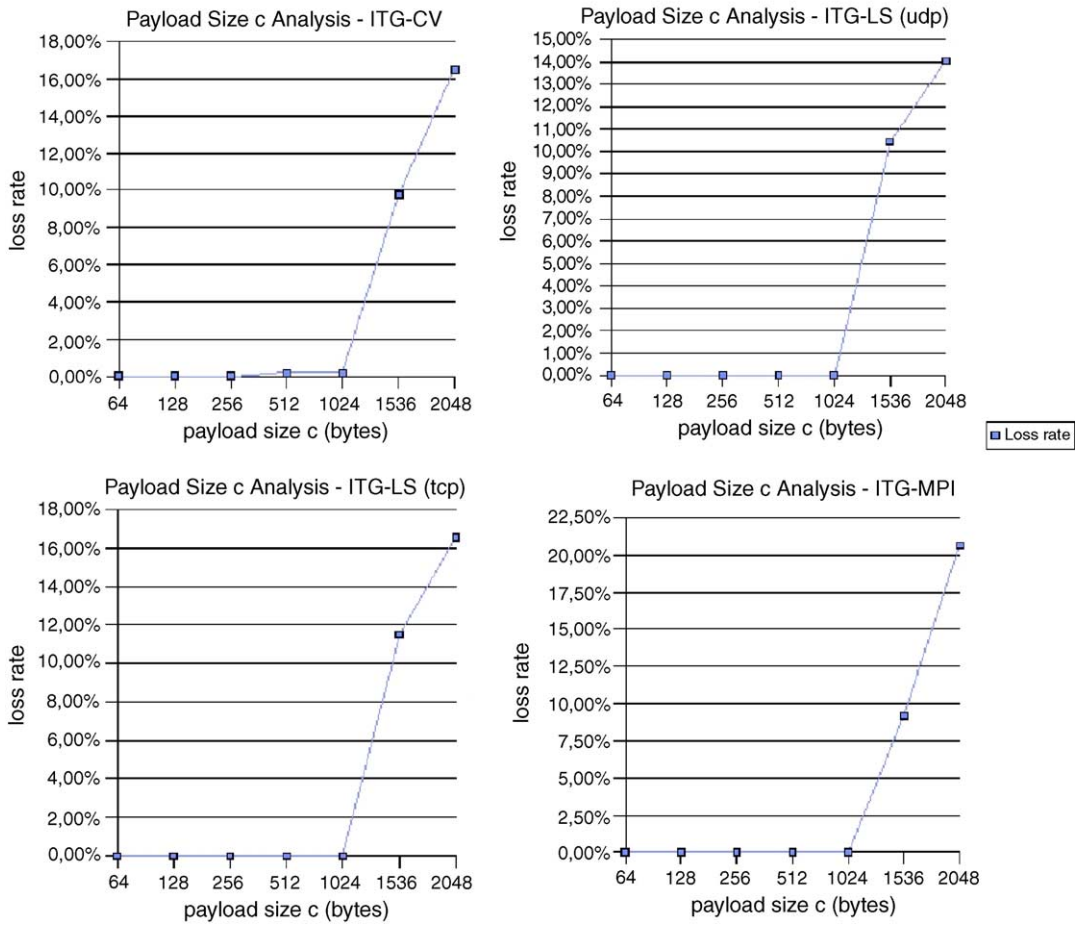


Fig. 32. Receiver packet size analysis ($C = 20{,}000$; $t = 60$ s): this figure shows the percentage error (percentage of packets not received) as a function of c. For all the implementations of D-ITG, the percentage error is null for $c < 1024$. For values above this threshold, we can observe a considerable packet loss.

is the same value of our traffic generation. Fig. 24(b) reports the mcalc output related to a VoIP traffic generation Codec G.711 with one voice sample per packet and with VAD. In this case, while in the theoretical model we have 100 packets per second and the payload equal to 80 bytes, using the VAD we observed a reduction equal to 35%. Taking again into the account 8 bytes of RTP header, the expected data rate is $(80 \times 0.65) \times 100 \times 8 = 48$ Kbps that is the same value of our traffic generation. In Fig. 24(c) the mcalc output related to a simulation with Codec G.711, two voice samples per packet and without VAD is reported. In this case, the theoretical model envisions 50 packets per second and the payload equal to 80 bytes. With 8 bytes of RTP header and the two voice samples per packet, the expected data rate is $(80 \times 2 + 8) \times 50 \times 8 = 67.2$ Kbps that is the same value of our traffic generation. In Fig. 24(d) the mcalc output related to a simulation with Codec G.729, two voice samples per packet and without VAD is reported. In this case in the theoretical model we have 50 packets per second and the

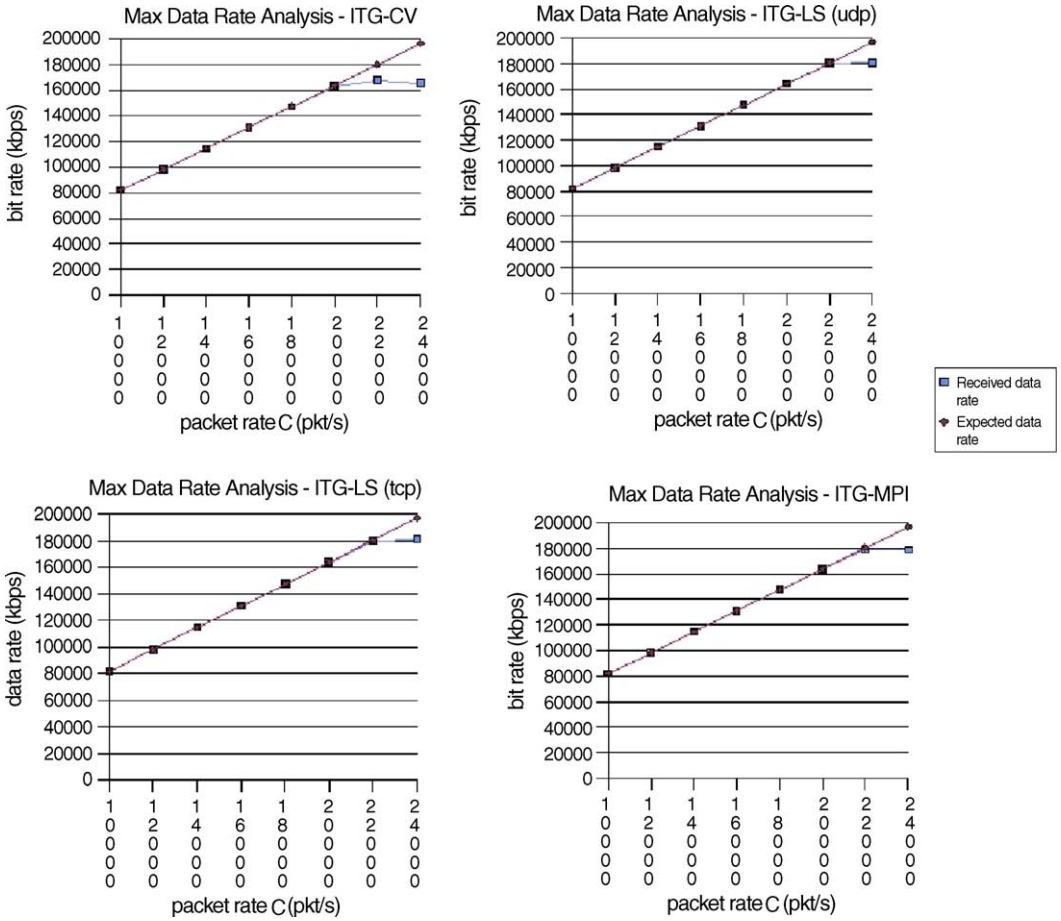


Fig. 33. Receiver max data rate analysis ($c = 1024$; $D = 40$): this figure depicts the received data rate as a function of C, while D and c are equal to their optimal values. For all the implementations of D-ITG, it is possible to determine an upper bound to the received rate. The smallest value is achieved by ITG-CV and is about 165,000 Kbps. The two versions of ITG-LS and ITG-MPI exhibit a similar behavior and their upper bound is about 180,000 Kbps. The gain achieved with respect to ITG-CV is about 10%.

payload equal to 10 bytes. Considering 8 bytes of RTP header and the two voice samples per packet, the expected data rate in this case is $(10 \times 2 + 8) \times 50 \times 8 = 11.2$ Kbps that is the same value of our traffic generation. In Fig. 24(e) the mcalc output related to a simulation with Codec G.729, three voice samples per packet and without VAD is reported. In this case in the theoretical model we have 33 packets per second and the payload equal to 10 bytes. With 8 bytes of RTP header and the three voice samples per packet, the expected data rate is $(10 \times 3 + 8) \times 50 \times 8 = 10.032$ Kbps that is still the same value of our traffic generation. In Fig. 24(f) the mcalc output related to a VoIP traffic generation with Codec G.723.1 with one voice sample per packet and without VAD is reported. In this case in the theoretical model we have 26 packets per second and the payload equal to 30 bytes. Taking into account 8 bytes of RTP header the expected data rate is $(30 + 8) \times 26 \times 8 = 7.9$ Kbps that is the same value of our traffic generation.

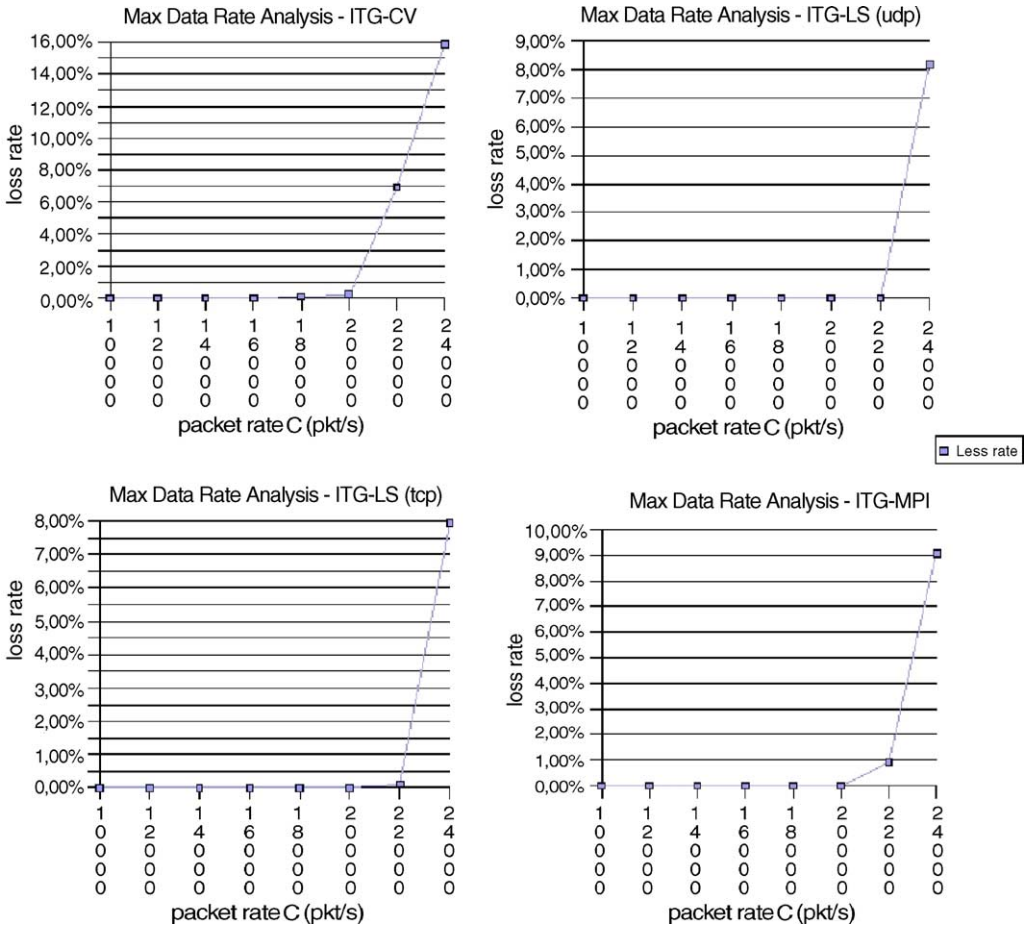


Fig. 34. Receiver max data rate analysis ($c = 1024$; $D = 40$): this figure shows the percentage of lost packets for the three implementations of D-ITG. The two versions of ITG-LS and ITG-MPI exhibit a null error rate for $C < 22{,}000$. ITG-LS presents a loss rate of 7% for $C = 22{,}000$. For C above this value, the error rate grows, but it is still below 8% for $C = 24{,}000$ for the two versions of ITG-LS.

7. Conclusion and future work

In this work we presented a general framework for traffic generation and performance characterization of our distributed platform named distributed internet traffic generator. It is able to reproduce real "Internet Traffic" according to theoretical models and with high performance with respect to existing and widely used traffic generators. Our work steps from the assumption that currently the Internet traffic generation is an important research task. Indeed, both the tutorials presented at SIGCOMM 2003 [11] and MMNS 2003 [8] have shown that Internet traffic patterns and models are particularly interesting for networking research community. D-ITG has been planned for generating network traffic (ICMP), transport layer traffic (TCP and UDP) and "application layer" traffic (HTTP, FTP, TELNET, SMTP, DNS, VoIP, Video, NNTP, . . .). In this paper we presented only some example of UDP and TCP generation and, as far as the application

level traffic, Telnet and VoIP traffic generation. D-ITG implements traffic generation according to several statistical distributions (exponential, uniform, constant, Pareto, cauchy, normal, . . .) both for IDT and PS random variables. It enables to simulate various network conditions under different traffic loads and network configurations. D-ITG is based on theoretical traffic models and represents a way for analyzing network performance through the measurement of the typical network parameters (delay, throughput, jitter, and packet loss). The basic idea of creating a new traffic generator arose from the lacks of existing ones (MGEN, Rude/Crude, etc.), emerged when we used them to analyze the different behavior of the network when some strategies that provide QoS were employed.

In this work 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 sender and receiver processes.

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 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 real heterogeneous mobile networks [29,34].

Acknowledgement

A preliminary short version of this work was presented in “Analysis and experimentation of an open distributed platform for synthetic traffic generation”, 10th IEEE International Workshop on Future Trends of Distributed Computing Systems (FTDCS 2004). We would like to thank the anonymous reviewers for their suggestions for improving our work and Salvatore Guadagno for its incredible work on the D-ITG platform. This work has been partially supported by the Italian Ministry for Education, University and Research (MIUR) in the framework of the FIRB Project “Middleware for advanced services over large-scale, wired-wireless distributed systems” (WEB-MINDS), by Regione Campania in the framework of “Centro di Competenza Regionale ICT”, and finally by the E-NEXT IST European project.

References

[1] M.F. Arlitt, C.L. Williamson, Web server workload characterization: the search for invariants, in: Measurement and Modeling of Computer Systems, 1996, pp. 126–137.
[2] F. Brasileiro, W. Cirne, E.B. Passos, T.S. Stanchi, Using remote memory to stabilise data efficiently on an ext2 linux file system, in: Proceedings of the SBRC 2002, May 2002.
[3] Z. Ba, H. Zhou, H. Zhang, Z. Yang, Performance evaluation of some MPI implementations on workstation clusters, Technical report, October 2000.
[4] M. Crovella, A. Bestavros, Explaining world wide web traffic self-similarity, vol. 29, Technical Report 1995–015, 1995.
[5] M. Crovella, A. Bestavros, Self-similarity in world wide web traffic: evidence and possible causes, in: Proceedings of ACM SIGMETRICS'96, Philadelphia, Pennsylvania, May 1996.
[6] Cisco, Traffic analysis for voice over ip, online white paper.
[7] R. Davies, Newran02a a random number generator library. http://webnz.com/robert/nr02doc.htm.
[8] Petre Dini, Internet multimedia traffic patterns, 2003.
[9] P. Danzig, S. Jamin, R. Caceres, D. Mitzel, D. Estrin, An empirical workload model for driving wide-area tcp/ip network simulations, 1992.
[10] P. Dickens, G. Thiruvathukal, Performance prediction for mpi programs executing on workstation clusters, in: Proceedings of the Conference of Parallel and Distributed Programming Techniques and Applications 1998 (PDPTA'98), Las Vegas, NV, 1998.
[11] J. Doyle, W. Willinger, 10 years of self-similar traffic research: a circuitous route towards a theoretical foundation for the internet, 2003.
[12] 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, 1994, pp. 267–280.
[13] M. Flouris, E.P. Markatos, The network ramdisk: using remote memory on heterogeneous NOWs, Cluster Computing 2 (4) (1999) 281–293.
[14] MPI Forum, Mpi: A message-passing interface standard, Technical report, 1994.
[15] M.W. Garrett, W. Willinger, Analysis, modeling and generation of self-similar VBR video traffic, in: SIGCOMM, 1994, pp. 269–280.
[16] Iperf, http://dast.nlanr.net/Projects/Iperf/.
[17] N.T. Karonis, B.R. de Supinski, I. Foster, W. Gropp, E. Lusk, J. Bresnahan, Exploiting hierarchy in parallel computer networks to optimize collective operation performance, in: 14th International Parallel and Distributed Processing Symposium, pp. 377–386.
[18] M. Krunz, H. Hughes, A traffic model for MPEG-coded VBR streams, in: Measurement and Modeling of Computer Systems, 1995, pp. 47–55.
[19] M. Lauria, A. Chien, MPI-FM: high performance MPI on workstation clusters, J. Parallel Distributed Computing 40 (1) (1997) 4–18.
[20] Beng O. Lee, Victor S. Frost, R. Jonkman, Netspec 3.0 source models for telnet, ftp, voice, video and www traffic, Technical Report 10980-19, ITTC, 1997.
[21] Mgen, http://mgen.pf.itd.nrl.navy.mil.
[22] S. McCreary, Kc claffy, Trends in wide area ip traffic patterns — a view from ames internet exchange, in: ITC Specialist Seminar, Monterey, CA, September 2000, pp. 18–20.
[23] P.E. McKenney, D.Y. Lee, B.A. Denny, Traffic generator software, In SRI International and USC/ISI Postel Center for Experimental, January 2002.
[24] Netperf, http://www.netperf.org/.
[25] Netspec, http://www.ittc.ku.edu/netspec/.
[26] Saurab Nog, David Kotz, A performance comparison of TCP/IP and MPI on FDDI, fast ethernet, and ethernet, Technical Report TR95–273, 1996.
[27] Ntgen, http://tochna.technion.ac.il/project/NTGen/html/ntgen.htm.
[28] Pacgen, http://pacgen.sourceforge.net/.
[29] A. Pescapé, S. Avallone, G. Ventre, Distributed internet traffic generator (d-itg): analysis and experimentation over heterogeneous networks.

[30] A. Pescapé, S. Avallone, G. Ventre, Analysis and experimentation of internet traffic generator, in: New2an'04, St. Petersburg, Russia, February 2004, pp. 70–75.
[31] V. Paxson, Empirically derived analytic models of wide-area tcp connections, IEEE/ACM Trans. Netw. 2 (4) (1994) 316–336.
[32] A. Pescapé, M. D'Arienzo, S.P. Romano, M. Esposito, S. Avallone, G. Ventre, Mtools, IEEE Netw. 16 (15) (2002) 3.
[33] V. Paxson, S. Floyd, Why we don't know how to simulate the internet, in: Winter Simulation Conference, SCS, 1997, pp. 1037–1044.
[34] A. Pescapé, G. Iannello, L. Vollero G. Ventre, Experimental analysis of heterogeneous wireless networks, in: WWIC 2004, Frankfurt, Germany, vol. 2957, February 2004, pp. 153–164.
[35] D. Papaleo, S. Salsano, The linux traffic generator, Technical Report 003–004-1999, University of Rome La Sapienza, 1999.
[36] J.W. Roberts, Traffic theory and internet traffic management, France Telecom R&D.
[37] O. Rose, Statistical properties of MPEG video traffic and their impact on traffic modeling in ATM systems, vol. 20, pp. 397–406, 1995.
[38] Rudecrude, http://www.atm.tut.fi/rude.
[39] M. Shro, R. Geijn, Collmark mpi collective communication benchmark, 2001.
[40] J. Squyres, A. Lumsdaine, W. George, J. Hagedorn, J. Devaney, The interoperable message passing interface (impi) extensions to lam/mpi, 2000.
[41] Tg traffic generators, http://www.caip.rutgers.edu/arni/linux/tg1.html.
[42] Traffic, http://rsandila.ezfish.net/traffic.html.
[43] Udpgen, http://www.fokus.fhg.de/usr/sebastian.zander/private/udpgen.
[44] Udpgenerator, http://www.citi.umich.edu/projects/qbone/generator.html.
[45] M. Zukerman, T.D. Neame., R.G. Addie, Internet traffic modeling and future technology implications, vol. 1, in: INFOCOM 2003, 2003, pp. 587–596.

Stefano Avallone received the MS degree in Telecommunications Engineering from Università di Napoli "Federico II", Italy in 2001. He is currently pursuing his Doctoral studies at the same University under the advisement of Prof. Giorgio Ventre. His research interests include computer and communication networks, traffic modelling and generation, traffic engineering, QoS routing. In 2004 he has been awarded a research funding from the European Doctoral School of Advanced Topics in Networking (SATIN), the instrument employed by E-NEXT (an EU FP6 Network of Excellence) to invest in education of researchers for the European Research Area.

Donato Emma received the MS degree in Computer Science at the University of Napoli Federico II in December 2002. Its graduation thesis regarded the definition and simulation of an architecture for contents distribution in VPNs. Currently he is a member of the COMputers for Interaction and CommunicationS (COMICS) group and he is involved in research projects in the area of Network Modelling and Simulation.

Antonio Pescapé is a PhD Student at Computer Engineering and Systems Department of the University of Napoli Federico II. He earned a Laurea Degree in Computer Engineering from the same institution. His research interests are in the networking field with focus on models and infrastructure for QoS over IP networks, models and algorithms for Internet Traffic, and Network Measuring and Management. He is IEEE member.

Giorgio Ventre is Professor of Computer Networks in the Computer Engineering and Systems Department of the University of Napoli Federico II. He earned a Laurea Degree in Electronic Engineering and a PhD in Computer Engineering, both from University of Napoli Federico II. As leader of the networking research group, he is the principal investigator for a number of national and international research projects. He has coauthored more than 100 publications and is a member of the IEEE Computer Society and of the ACM.