# A Distributed Multiplatform Architecture For Traffic Generation

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

## Abstract

*This paper deals with the specification and realization of a synthetic traffic generator for IP traffic, called D-ITG (Distributed Internet Traffic Generator). We point our attention on the innovative architectural choices and the consequent interesting results. Indeed, thanks to the novel proposed distributed architecture, D-ITG reaches the highest performance over all the considered platforms and it shows interesting and innovative features in terms of supported protocols, available traffic patterns and meters type. In a heterogeneous network scenario, we think that D-ITG is an essential tool for network testing and planning activities. We compared our tool with several of the currently available and most widely adopted traffic generators and we found that D-ITG offers several improvements in terms of both functionalities and performance. The presented architecture and the comparative analysis shown in this work confirm our assumption.*

**Keywords:** Internet Traffic Generation, Distributed Architecture, Performance Evaluation.

## 1. Introduction

Over the last twenty years, considerable effort has been made to understand and characterize the behavior of the Internet. The extreme complexity of large topologies and their traffic characteristics make the development of analytical models difficult. Under such conditions, simulation is the most promising technique for understanding network behavior. Simulation modeling of computer networks is an effective technique for evaluating the performance of networks as well as transport and application-level protocols. Traffic generation is one of the key challenges in modeling and simulating the Internet. For a small simulation with a single congested link, simulations are often run with a small number of competing traffic sources. However, for a larger simulation with a more realistic traffic mix, a basic problem is how to introduce different traffic sources into the simulation. For this reason, most of the international researchers move towards simulation environments like *ns* [1] or others. At the same time, simulating how wide area networks behave is complicated by the heterogeneity of these networks and their fast pace of evolution. The interaction between the traffic from the diverse suite of protocols that operate over the Internet and the hierarchical nature of the topologies are a few of the factors contributing to the complexity of such large networks [2]. We refer to [3] for a detailed description of the many difficulties involved in simulating the Internet realistically. In this work we present a contribution to one of the most critical aspects of network architecture analysis: synthetic generation of *realistic traffic over real networks*. The motivations at the base of our choice are the following. Significant progress has been made in the last few years in tools for realistic traffic generation, for both simulations and analysis. An approach of "real simulation" permits to overcome some typical constraint: (i) generally we simulate protocols but we ignore how they are actually implemented in terminals and nodes; (ii) we need to consider computational aspects for applications, nodes, systems; (iii) macro-scale evaluations highly depending on phenomena dynamics; (iv) in a simulation environment like *ns* there is a synchronous coordination among the simulated events.

This paper is organized in five sections. After this introduction, Section 2 presents the motivations at the base of D-ITG and a list of the relevant features. In the Section 3 D-ITG model details, architectural choices and functional modalities are presented. Section 4 shows a complete D-ITG performance analysis and a comparative study over a multiplatform scenario among several traffic generators. Finally, Section 5 presents some conclusions and the issues for future research and applications.

## 2. Distributed Internet Traffic Generator (D-ITG)

For analysis of new applications and network mechanisms over the Internet and for testing Quality of Service (QoS) architectures, a generator of controllable, scalable, synthetic but realistic IP traffic is required. D-ITG [4] has been developed for this purpose. From our point of view, "realistic" traffic is defined as traffic that is statistically similar to traffic generated on a real network from real protocols/applications. D-ITG can generate multiple independent flows with given traffic profiles (in terms of *Inter Departure Time* and *Packet Size* variables). D-ITG works on: (i) PC with Linux Operating System and Windows Operating System as well; (ii) PDAs with Linux Familiar Operating System.

In general, other generators typically produce traffic in a controlled test-bed environment where there are few real users and a corresponding low traffic load. In the field of traffic generators as far as workload granularity, several approaches are possible: (i) Aggregate traffic, (ii) Connection flow or session arrival, (iii) Flow duration or lifetime, (iv) Packet arrival within single connection. D-ITG aims to emulate complex network systems generating traffic on a packet-by-packet basis. A traffic flow is specified through the packet *Inter Departure Time (IDT)* - the time between the transmissions of two successive packets - and the *Packet Size (PS)* - the amount of data being transferred by the packet. By using this information, a per protocol traffic model could be created. Both processes (IDT and PS) are modeled as i.i.d. series of random variables (constant, uniform, exponential, pareto, normal, cauchy, etc…). Due to the architectural approach adopted, for each flow users can define the pattern of packet emission with millisecond resolution. One of the interesting features is the possibility to reproduce exactly the same experiment by choosing the same seed values for IDT and PS random processes. With respect to traffic generators working at session level, flow level or connection level, one of the major advantages of a packet level traffic generator is its simplicity. At opposite side, packet level traffic generators miss some aspect of network behavior. One of these is that the characteristic of one packet could determine the characteristics of successive packets. In this case a per packet traffic generator would require an extremely complex set of rules to model this kind of behavior.

 ISBN: 1-56555-284-9

By using D-ITG it is possible to evaluate a set of QoS performance metrics related to throughput, loss, delay and jitter. In our opinion, D-ITG is a key component for the experiments in a testing or planning phase for IP based networks. In particular, in the context of QoS IP networks is useful to have software architectures able to evaluate the performance of IP traffic control mechanisms supporting QoS. As far as this last point D-ITG provides setting the TOS (Type of Service) field and TTL (Time to live) field too. Statistics related to the generated traffic flow can be collected by analyzing the information stored by both the sender and the receiver. An appropriate utility enables to determine the average values of throughput, delay, jitter and packet loss not only on the whole duration of the experiment, but also on windows of the desired duration. Finally, different protocols may be tested to discover differences in performance.

To demonstrate the applicability, the performance and the usefulness of D-ITG this paper shows an example of a simple scenario created for testing D-ITG performance in an experimental test-bed. The test-bed is an IP based communication platform where a complete and comprehensive comparative analysis is carried out. Indeed after a complete analysis of D-ITG we present a detailed comparative analysis with the following other traffic generators: Mtools [5], Rude/Crude [6], Mgen [7], Iperf [8] and UDPgenerator [9]. The comparative studies in this paper are conducted with CBR (Constant Bit Rate) UDP traffic, even though D-ITG is able to generate stochastic traffic patterns.

In this paper we point our attention on the innovative solutions that we introduced in the field of traffic generators and we analyze the related achieved performance. The motivations at the base of our work are presented in [10] where a complete analysis of related work is present too. In [11] considerations and details on different distributed D-ITG implementations are presented and finally, the use of D-ITG for a comprehensive performance analysis of heterogeneous wireless networks is described in [12] and [13].

# 3. D-ITG Software Architecture

D-ITG platform defines a distributed multi-component architecture for high performance Internet traffic generation in heterogeneous environment. The main components of D-ITG are: (i) Internet Traffic Generator Sender (ITGSend), (ii) Internet Traffic Generator Receiver (ITGRecv), (iii) Internet Traffic Generator Log Server (ITGLog), (iv) ITGSend Manager (ITGManager). Each one of the previous component and in particular ITGSend and ITGRecv present an internal distributed implementation: several kinds of distributed architectures have been carried out (i.e. an MPI version is available). More details of distributed version are reported in [11]. Figure 1 shows a graphical overview on the relationship among the four main bricks of D-ITG platform.

ITGSend and ITGRecv coordinate their activities implementing the *Traffic Specification Protocol (TSP)*, described below, over a separate signaling channel. A similar signaling channel is used by ITGManager to drive ITGSend. Both ITGSend and ITGRecv can use ITGLog to collect information about the generation experiment using a Log channel and a signaling channel. In the next subsections we present the *Traffic Specification Protocol* and each component of the D-ITG platform.

## 3.1. Traffic Specification Protocol (TSP)

In order to set up an innovative and efficient architecture for the traffic generation we introduced a novel protocol for the definition of each experiment requirements: sender and receiver decide the experiment parameters and control the traffic generation by using TSP. In particular, TSP is a protocol that we introduced in order to: (i) create a connection between a sender and a receiver; (ii) authenticate a receiver; (iii) exchange information on a generation process; (iv) close a sender-receiver connection; (v) detect generation events;

Figure 2 and Figure 3 show the TSP state diagram of both the receiver and the sender representing the transitions following the possible events. According to this protocol, the generation of a traffic flow is preceded by: (i) the creation of a connection between the sender and the receiver; (ii) the receiver authentication obtained by a challenge protocol; (iii) the exchange of information on the flow to be generated; (iv) moreover, the sender must communicate the end of a flow generation to the receiver. Figure 4 shows a generic TSP packet. The only mandatory field is *type*; 15 values have been defined for it, the unused values are available for future extensions of the protocol. The type value defines the set of fields included in the packet. Table 1 describes the purpose of each TSP packet, with the indication of the corresponding value for the type field; instead, Table 2 describes the meaning of the different fields when used in different TSP packets.

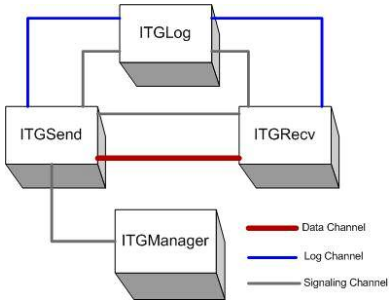



**Figure 1: D-ITG Component Architecture**

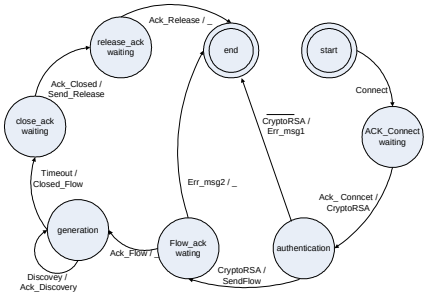



**Figure 2: TSP Sender side**

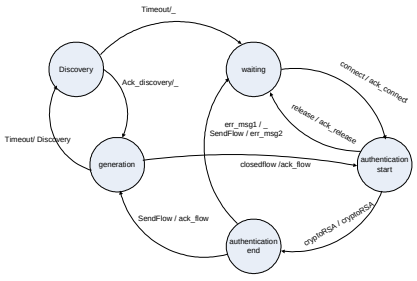



**Figure 3: TSP Receiver side**

 ISBN: 1-56555-284-9

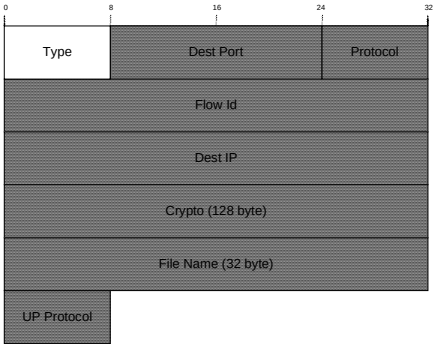



**Figure 4: TSP Packet**

## 3.2. ITGSend Architecture

ITGSend is the sender component of the D-ITG traffic generation platform. ITGSend can operate in three different modes:

- single flow mode (Figure 5): ITGSend generates a single flow; a single thread is responsible for the generation of the flow and the management of the signaling channel through the TSP protocol;
- multiple flows mode (Figure 6): ITGSend generates a set of flows; it operates as a multithreaded application. One of the threads implements the TSP protocol and drives the generation process, while the others generate the simulated flows.
- daemon mode (Figure 7): ITGSend is remotely controlled by ITGManager using the ITGApi. ITGApi is an API that currently provides just one function, which can be used to send a message to ITGSend. This message specifies the parameters (destination IP address and port, inter-departure time characterization,…) of the flow to be generated.

Each flow to be generated by ITGSend 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 those implemented (constant, uniform, normal, cauchy, pareto, exponential,…). Moreover, ITGSend allows sending traffic according to the theoretical models for various protocols (Telnet, DNS, VoIP,…). This means that the user can simply choose one of the implemented protocols, the distributions and the corresponding parameters for the inter-departure and packet size random variables are automatically determined by ITGSend. In this way D-ITG generates "*synthetic but real traffic over real networks*".

To collect statistics on the generation process ITGSend can log detailed information about the generated flows: (1) flow number; (2) sequence number; (3) source address; (4) destination address; (5) transmission time; (6) receiving time; (7) packet size.

This information can be stored either in a local log file or in a remote log file using the log server ITGLog. This log file 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 real traffic generation is 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 ITGSend 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. Also the choice of a multithreaded implementation of ITGSend is tied to the need of limiting the interference among the generations of different simultaneous flows.

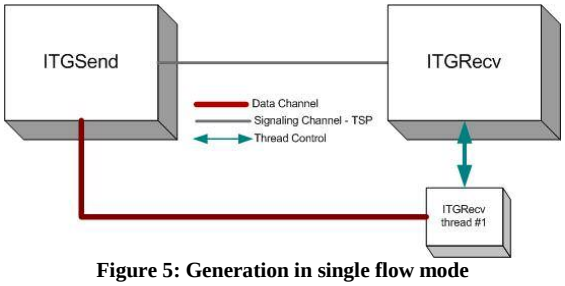



**Figure 5: Generation in single flow mode**

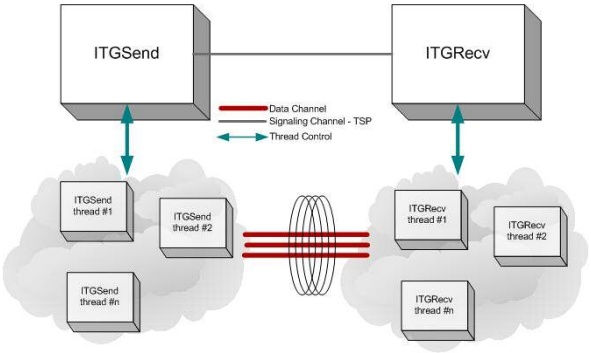



**Figure 6: Generation in multiple flow mode**

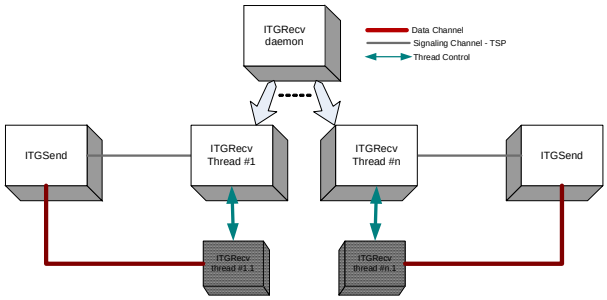



**Figure 7: ITGRecv daemon model.**

## 3.3. ITGRecv Architecture

ITGRecv always works as a concurrent daemon: it listens for new TSP connections on port 9000; when a TSP connection request arrives, ITGRecv generates a new thread that is responsible for the TSP protocol implementation; as shown in Figure 7, and each single flow is received by a separate thread.

Similar to ITGSend, ITGRecv generates a log file that describes at packet level each received flow. This log file can be stored locally or remotely using the log server ITGLog.

## 3.4. The signaling channel

D-ITG implements the TSP protocol over a *TCP signaling channel* between the sender ITGSend and the receiver ITGRecv. Thanks to the multithreaded implementation of both ITGsend and ITGRecv, each signaling channel can be used for multiple flows generation. For each multiple flows generation experiment a TSP connection between an ITGRecv controller thread and an ITGSend controller thread is established.

This thread implements the TSP protocol and it is responsible for instantiating and terminating the threads that generate and receive the simulated flows. The coordination between the controller thread and the threads that are delegated to generate or receive packets is made using the *Inter Process Communication (IPC)* [14] in Unix-like systems, or the *event communication* [15] in Windows system.

### 3.5. ITGRecv authentication

The D-ITG platform has been designed and implemented to simulate Internet traffic at very high bit rate. This feature can be easily used to implement attacks such as ***D**enial **o**f **S**ervice* (Figure 8). To limit this event, and in general to arrange an agreement between sender and receiver, before starting a generation experiment ITGRecv and ITGSend implement a challenge-response authentication protocol [16]. The use of this authentication method allows ITGSend to make sure that the receiving host really wants to receive traffic. Thus, to implement a DoS attack using D-ITG it is necessary that the attacked host has already been under the control of an intruder (the attacker was able to start ITGRecv on it).

### 3.6. ITGLog

ITGLog is a "log server", running on a different host with respect to ITGSend and ITGRecv, which receives and stores the log information from multiple senders and receivers. The logging activities is handled using a signaling protocol.

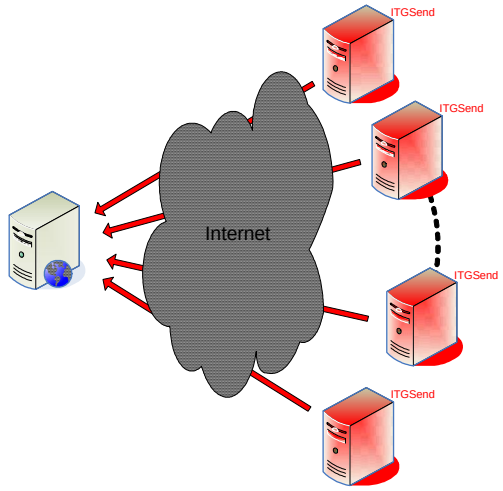



**Figure 8: DoS attack with ITGSend**

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). ITGLog can be used in different scenarios such for example:

1. wide area traffic generation (Figure 9): when D-ITG is used in a wide area distributed scenario ITGLog can be used to easily collect the log file of all the senders/receivers. In this way it is possible to implement a centralized and possibly "on_the_fly" results analysis;
2. device with limited storage resource (Figure 10): if a device with limited storage resource, such as for example a PDA (Personal Digital Assistant), is used to send or receive a traffic flow, ITGLog can be used to collect the log information that can not be stored over a such device.

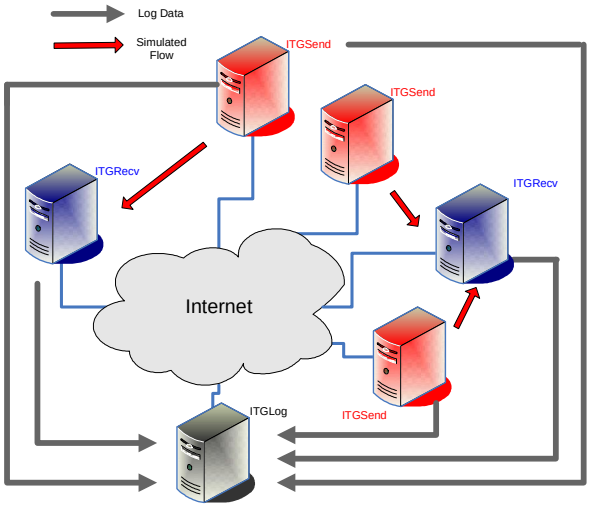
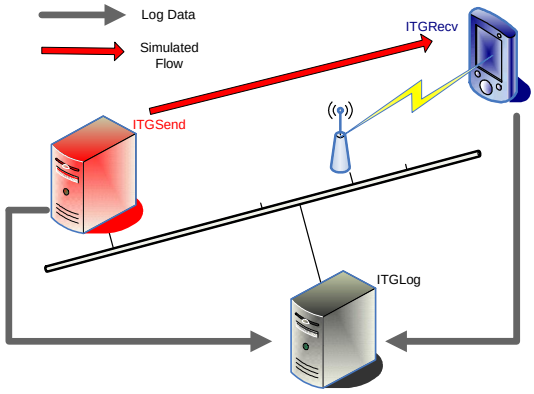



**Figure 9: ITGLog in wide area experiment**




**Figre 10: ITGLog in a wireless experiment**

### 3.7 ITGManager

As described before, ITGSend can be launched in daemon mode and stay idle waiting for commands from ITGManager. ITGManager uses ITGApi to remotely control ITGSend. ITGApi is a C++ API that currently provides just one function, which enables ITGManager to send a message to ITGSend. Through this message, ITGManager can issue the generation of a traffic flow. The syntax of this message is the same as that used to require a flow generation from the command line. ITGManager can remotely control more than one ITGSend, as depicted in
Figure **11**. In this way, ITGManager can control the whole traffic crossing the network. This feature can be used, for example, to test centralized routing algorithms in a real environment. Indeed, we can assume the presence of a "network controller" which receives flow requests and determines the path that the corresponding flow must follow in order to satisfy flow requirements and optimize network resource usage.
We are assuming that the network architecture allows to explicitly routing flows (e.g. MPLS). After the path has been established, ITGManager can issue the generation of the traffic flow. By collecting statistics about average delay, jitter and packet loss related to several flows, we can compare the performance of different traffic engineering algorithms.

## 4. D-ITG: comparative analysis and performance evaluation

In this section before presenting experimental results, we present a complete description of D-ITG features. D-ITG gives the ability to generate network traffic on remote network segments for a "*What If*" analysis during the planning and management of networks. Using D-ITG is useful in order to perform highly specific tests on remote network segments from a central management station.

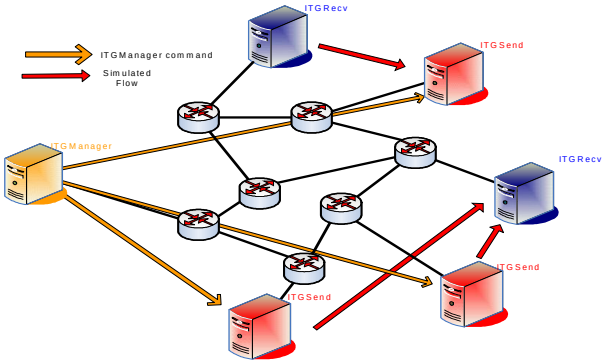



**Figure 11: ITGSend in daemon mode**

This provides a fast and efficient way to perform troubleshooting, stress testing on specific devices and capacity planning Table 3 presents a comparative schema that summarizes D-ITG characteristics and contrasts it with respect to other widely used traffic generators. In [10] a detailed related work analysis is reported. D-ITG is currently available both on Linux and Windows platform. It presents both a multithread and a multitask implementation. The supported protocols are: TCP, UDP, ICMP, DNS, Telnet, VoIP (G.711, G.723, G.729, Voice Activity Detection, Compressed RTP). Currently we are working on SMTP, HTTP, FTP, P2P, SNMP, MPEG protocol implementation. The provided stochastic processes both in the case of PS and IDT are Constant, Uniformly distributed, Exponentially distributed, Pareto distributed, Cauchy distributed, Normal distributed, Poisson distributed, Gamma distributed. Thanks to this wide range of supported stochastic processes it is possible to reproduce a broad range of traffic mixture. D-ITG provides setting of generation seed: this option gives the possibility to repeat different experiments using the same seed. D-ITG can perform both one-way-delay (OWD) measurement and round-trip-time (RTT) measurement, packet loss evaluation, jitter and throughput measurement. Another innovative feature of D-ITG is the possibility to store information both on the receiver and on the sender and, additionally, to remotely store information. As far as this last feature, D-ITG enables the sender and the receiver to delegate the logging operation to a remote log server; this option is useful when the receiver has limited storage capacity - e.g. PDAs, palms, etc. – and when the log information must be analyzed "*on-the-fly*", for example, in case the sender is asked to adapt the transmission rate based on channel congestion and receiver capacity. D-ITG permits the setting of TOS (DS) and TTL packet field. Both experiment duration and delay (initial time of the experiment) can be set. The communication between sender and receiver is made by using a separated signaling channel that implements a protocol for the configuration of traffic generation experiment (Traffic Specification Protocol). Furthermore, the sender can be remotely controlled by using ITGapi. This means that the D-ITG sender can be launched in daemon mode and waiting for commands, so that generation of traffic flows can be remotely controlled. By using this feature is possible to test traffic engineering algorithms in a *real* network.

D-ITG is able to reach high (receiver and sender) data rate. In particular, in a local environment (sender and receiver over the same Linux platform) the maximum data rate is equal to 511 Mbps both at sender and receiver side; in a distributed environment (sender and receiver over two different Linux platforms) the maximum data rate is equal to 612 Mbps at sender side and 611 Mbps at receiver side. As far as this last point in this section in order to demonstrate D-ITG performance we present a performance analysis study both over Linux and Windows platform. We study the D-ITG performance over a local machine too in order to (i) study the interference between sender and receiver process and (ii) isolate the dependencies from network dynamics. In Table 4 and Figure 12 all parameters of our experimentation and the network testbed are summarized.

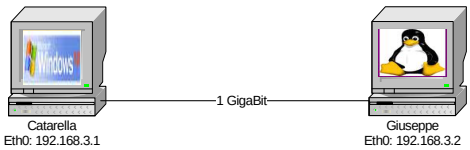



**Figure 12: Testbed**

## 4.1. Linux Platform: local experimentation

In this case we study D-ITG performance at sender and receiver side by storing information at sender and receiver side. This feature is not available in all analyzed traffic generators. Thus we present both the results related to logging phase only at receiver side and results related to logging phase at sender and receiver side.

This "*modus operandi*" has been carried out over the same platform (local experimentation, sender and receiver over the same device) and between two distinct but identical PCs (distributed experimentation, sender and receiver over two distinct devices).

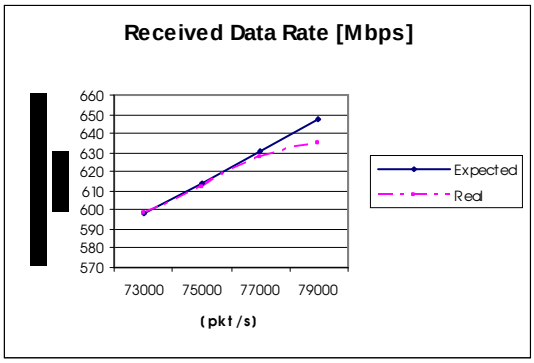



**Figure 13: Local received data rate over Linux platform (log at receiver side)**

### 4.1.1. Local experimentation with logging only at receiver side

Over Linux platform and by using the log file at the receiver side D-ITG presents the performance depicted in Figure 13. In Figure 14 the percentage loss rate (between real and expected data rate) as a function of the packet rate is reported. In the case of C=75000pkt/s, c=1024byte, t=60s D-ITG reaches a received data rate equal to 611Mbit/s with a percentage loss rate equal to 0.4%.

By using the same parameters, in

Figure **15** a comparative analysis between traffic generators that can log at receiver side is reported. As far as experimental results, D-ITG shows the best performance. It is important to underline that Iperf (which presents a percentage loss rate equal to 5,3% with respect to 0,4 % of D-ITG) works in a different way with respect to D-ITG. 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. By using Iperf it is not possible to make a complete performance study, it is only possible to determine average values on the whole duration of the experiment.

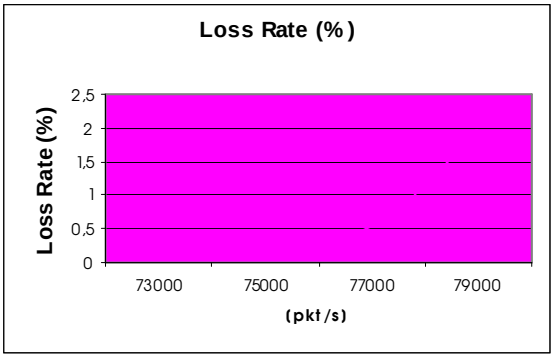



**Figure 14: Local loss rate (%) over Linux platform (log at receiver side)**

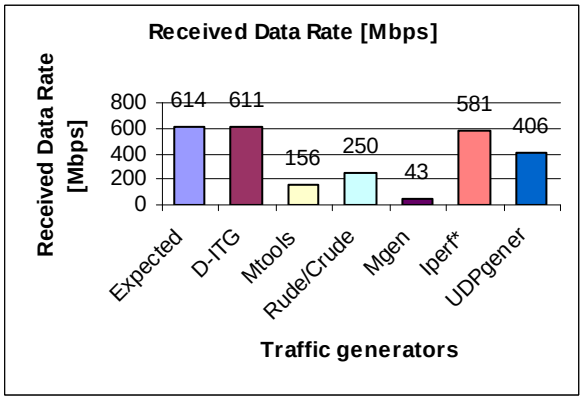



**Figure 15: Comparative analysis of local received data rate over Linux platform (log at receiver side)**

### 4.1.2. Local experimentation with logging both at receiver and sender side

Over a single Linux platform and by using log file both at receiver and sender side D-ITG presents the performance depicted in Figure 16 and Figure 17.

In particular in Figure 16 (Figure 17) a comparative analysis with respect to generated (received) data rate between traffic generators that can log both at receiver and sender side is reported. In the case of C=63000pkt/s, c=1024byte, t=60s D-ITG reaches a received data rate equal to 511Mbps. By observing Figure 16 and Figure 17. it is possible to understand that: (i) by logging both at sender and receiver side D-ITG presents a reduction of 100 Mbps; (ii) both D-ITG and Iperf can receive all sent packets. Other analyzed generators present a high rate of loss packets. Iperf presents a percentage reduction higher than D-ITG when the log process, at both side of the communication, is performed. Finally, in this case Mtools presents a generated data rate higher than that one observed in
Figure **15**: this is due to the different packet rate (C).

## 4.2. Windows Platform: local experimentation

### 4.2.1. Local experimentation with logging only at receiver side

Over Windows platform and by using the log file at the receiver side, D-ITG presents the performance depicted in Figure 18.

In the case of C=30000pkt/s, c=1024byte, t=60s D-ITG reaches a received data rate equal to 241Mbit/s with a percentage loss rate equal to 1.8%. At the same received data rate Iperf and Mtools present respectively a loss rate equal to 38% and 5.3%. In addition, in section 4.1.1 we have already clarified that the Iperf logging process does not permit a complete performance analysis.

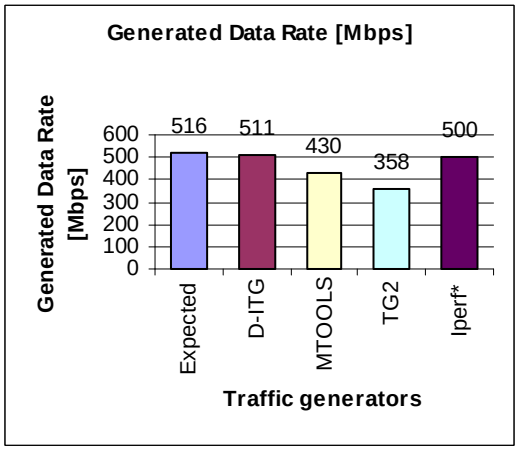



**Figure 16: A comparative analysis with respect to generated data rate over Linux platform (local and log at sender and receiver side)**

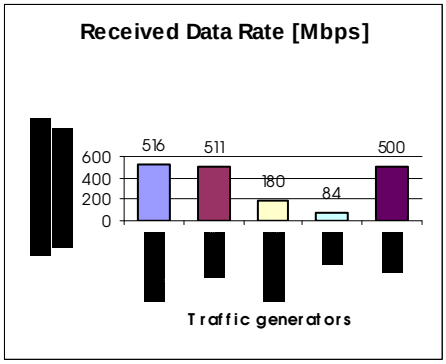



**Figure 17: A comparative analysis with respect to received data rate over Linux platform (local and log at sender and receiver side)**

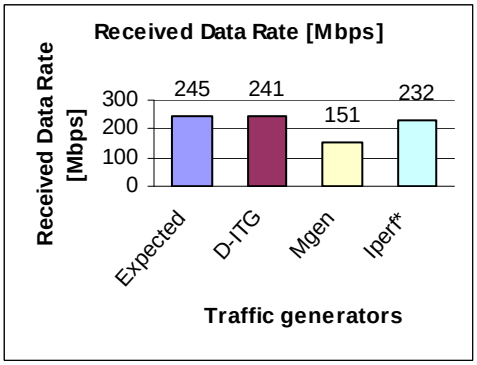



**Figure 18: Comparative analysis of local received data rate over Windows platform (log at receiver side)**

 ISBN: 1-56555-284-9

In Figure 19 the percentage loss rate (between real and expected data rate) as a function of packet rate is reported. In Figure **20** the percentage loss rate trend is depicted.

### 4.2.2. Local experimentation with logging both at receiver and sender side

In this subsection we can analyze only D-ITG because there are not traffic generators, over Windows platform, that can log both at sender and receiver side. In this case, D-ITG presents a reduction of 102 Mbps (Figure 21). Indeed in this case, with a percentage loss rate equal to 1.8%, we had a max data rate equal to 143 Mbps for 17500 pkt/s (in the previous case for the same percentage loss rate we had a max data rate equal to 245 Mbps for 30000 pkt/s). In Figure 22 and in Figure 23 the percentage loss rate respectively at sender and receiver side is depicted.

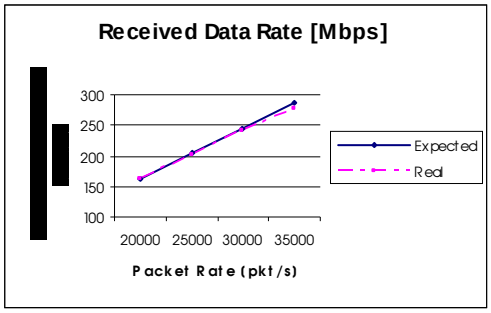



**Figure 19: Local received data rate over Windows**

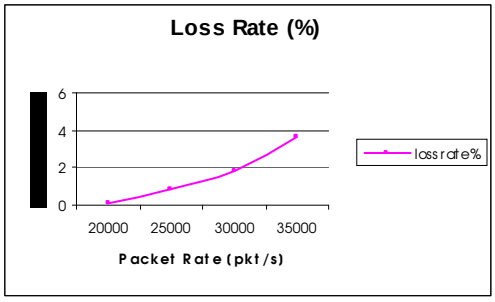



**Figure 20: Local loss rate (%) over Windows platform (log at receiver side)**

Before to study the performance in a distributed testbed, we can conclude that under the same conditions, over Windows platforms the maximum data rate is less than maximum value over Linux platforms.

## 4.3. Linux Platform: distributed experimentation

In this subsection we describe the results of the distributed experimentation over Linux platform. In this case we used the same operational mode of the section 4.1.

### 4.3.1. Distributed experimentation with logging only at receiver side

Between two Linux devices and by using the log file at the receiver side D-ITG present the performance depicted in Figure 24. In Figure 25 a percentage loss rate (between real and expected data rate) as a function of packet rate is reported. In the case of C=77000pkt/s, c=1024byte, t=60s D-ITG reaches a received data rate equal to 627 Mbit/s with a percentage loss rate equal to 0.5%. In Figure 25 the percentage loss rate trend is depicted. It is important to note that also over two distinct devices D-ITG is able to reach high performance. Figure 26 shows a comparative analysis between traffic generators over two distinct devices.

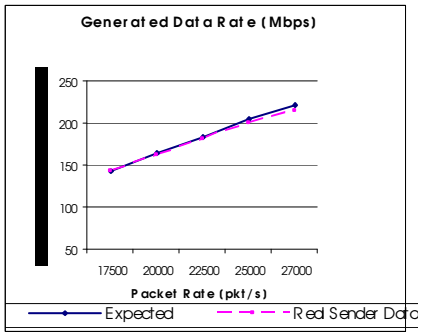



**Figure 21: Local generated data rate over Windows platform (log both sender and receiver side)**

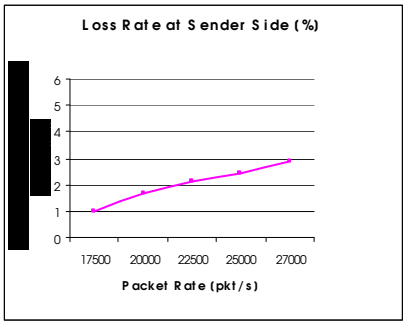



**Figure 22: Local loss rate (%) at sender side over Windows platform (log both sender and receiver side)**

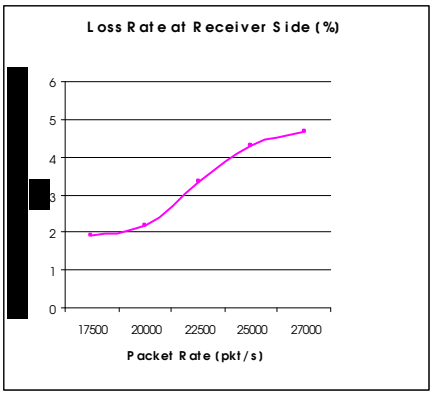



**Figure 23: Local loss rate (%) at receiver side over Windows platform (log both sender and receiver side)**

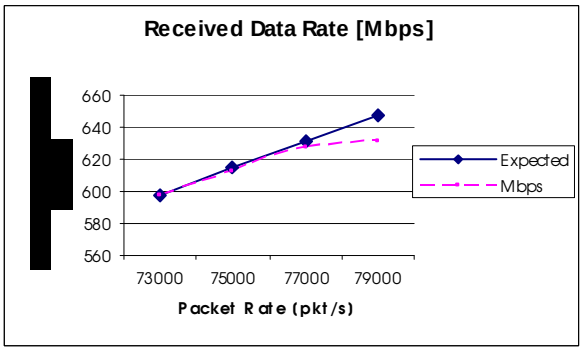



**Figure 24: Distributed received data rate over Linux platform (log at receiver side)**

 ISBN: 1-56555-284-9

Also in this case, D-ITG presents the best performance. It is important to underline that Rude/Crude for 77000 pkt/s presents a very low performance (11 Mbps). Indeed we studied the Rude/Crude performance by varying the packet rate: the results are depicted in Figure 27. Rude/Crude presents the maximum data rate for a packet rate equal to 60000 pkt/s (438 Mbps with an error, between the expected and real value, equal to 0.9%). Thanks to this analysis, we can conclude that: (i) Rude/Crude is able to reach data rate up to 438 Mbps whereas D-ITG reaches 627 Mbps; (ii) Rude/Crude cannot generate traffic with a high packet rate. Regarding Iperf, we must take into account the observation made at the end of the section 4.1.1

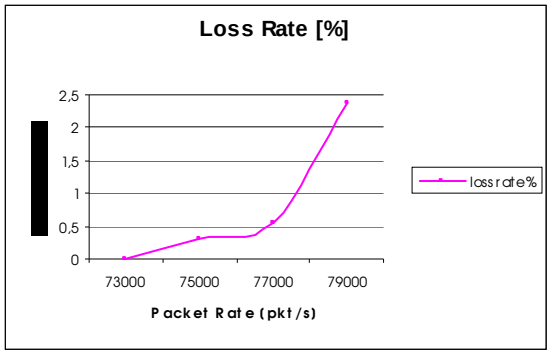



**Figure 25: Distributed loss rate (%) over Linux platform (log at receiver side)**

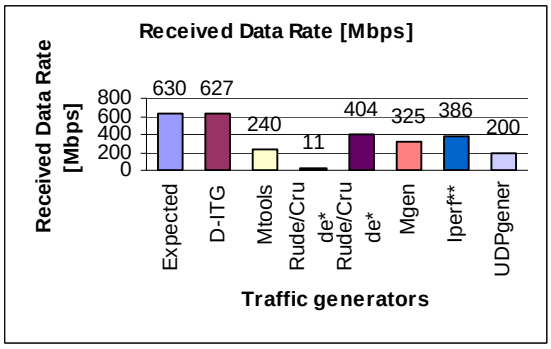



**Figure 26: Distributed received data rate over Linux platform (log at receiver side)**

### 4.3.2. Distributed experimentation with logging both at receiver and sender side

Over two distinct Linux devices and by using log file both at receiver and sender side D-ITG presents the performance depicted in Figure 28. In this figure the data rate (both at sender and receiver side) trend is reported. In the case of C=75000pkt/s, c=1024byte, t=60s D-ITG reaches a generated data rate equal to 612Mbps (with an error equal to 0.5%). The logging process at sender side implies a maximum data rate reduction in the generation phase. Moreover, all generated packets have been received at receiver side. In Figure 29 a comparative analysis among traffic generators over two distinct Linux devices with respect to generated data rate is shown. In the same way in Figure 30 a comparative analysis with respect to received data rate is sketched. Results shown in Figure 29 and in Figure 30 present some interesting properties: (i) D-ITG presents (both at sender and receiver side) high performance; (ii) TG2 presents very high performance. This result is not valid for the local experimentation. Indeed, in that case TG2 reaches respectively 358Mbps at sender side and 84Mbps at receiver side. Instead, D-ITG presents substantially the same performance both in the local and in the distributed experimentation.

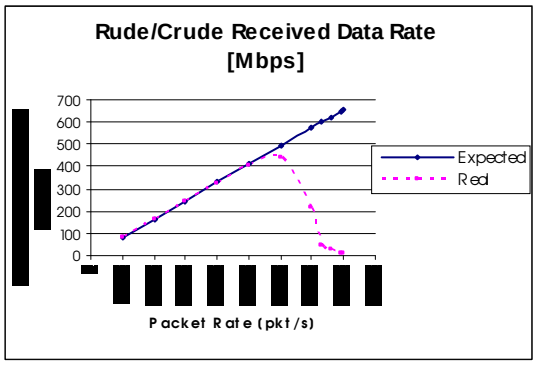



**Figure 27: Distributed loss rate (%) over Linux platform (log at receiver side)**

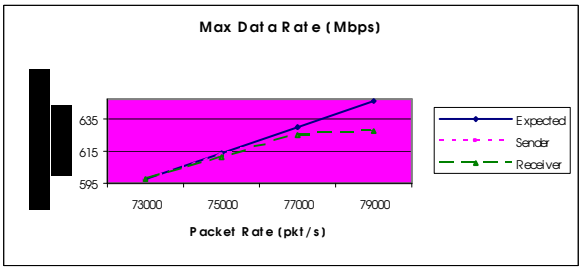



**Figure 28: Generated and received data rate over Linux devices (distributed and log both at sender and receiver side)**

## 4.4. Windows Platform: distributed experimentation

In this subsection we describe the results of the distributed experimentation over Windows platform. In this case we used the same operational mode of the section 4.3.

### 4.4.1. Distributed experimentation with logging only at receiver side

Between two Windows devices and by using the log file at the receiver side D-ITG present the performance depicted in Figure 31. In Figure 32 a percentage loss rate (between real and expected data rate) as a function of packets per second number is reported. In the case of C=30000pkt/s, c=1024byte, t=60s D-ITG reaches a received data rate equal to 242Mbit/s with a percentage loss rate equal to 1.4%.

It is important to note that also over two distinct devices D-ITG is able to reach its higher performance. Figure 33 shows a comparative analysis between traffic generators over two distinct devices: also in this case D-ITG presents the highest performance.

### 4.4.2. Distributed experimentation with logging both at receiver and sender side

D-ITG is the only traffic generator that, over Windows platform, permits to log both at sender and receiver side. Thus, over two distinct Windows devices and by using log file both at receiver and sender side D-ITG presents the performance depicted in Figure 34 (sender) and in Figure 35 (receiver).

In this configuration with C=20000pkt/s, c=1024byte, t=60s D-ITG reaches a generated data rate equal to 161 Mbps (with an error equal to 1.5 %). The logging process at sender side implies a maximum data rate reduction in the generation phase. Moreover, all generated packets have been received at receiver side.

 ISBN: 1-56555-284-9

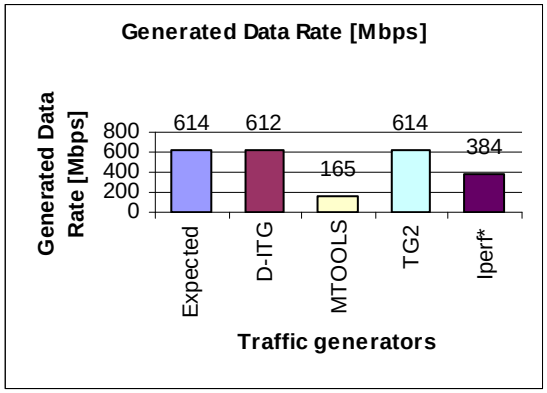



**Figure 29: A comparative analysis with respect to generated data rate over Linux platform (distributed and log at sender and receiver side)**

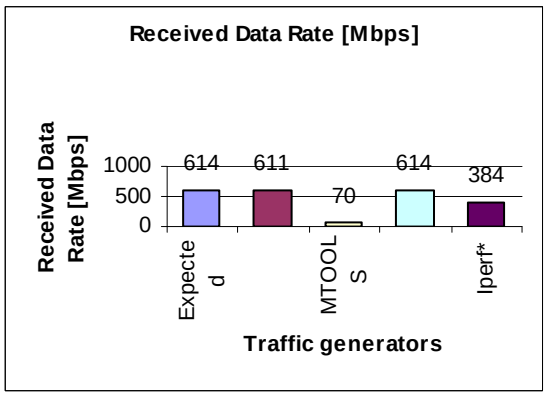



**Figure 30: A comparative analysis with respect to received data rate over Linux platform (distributed and log at sender and receiver side)**


**Figure 31: Received data rate over Windows platform (distributed and log at receiver side)**

## 4.5. A cross platform experimentation

In this subsection we present an interesting study concerning a cross platform performance analysis. We choose traffic generators running both over Windows and Linux platforms and we carried out a comparative analysis interchanging the role of sender and receiver: let $TG_i$ the i-th traffic generator, let W-$TG_i$ the Windows version of the i-th traffic generator and let L-$TG_i$ the Linux version of the i-th traffic generator. We carried out a complete performance measurement in the case of:

- L-$TG_i$ Sender ⇗ W-$TG_i$ Receiver
- W-$TG_i$ Sender ⇗ L-$TG_i$ Receiver

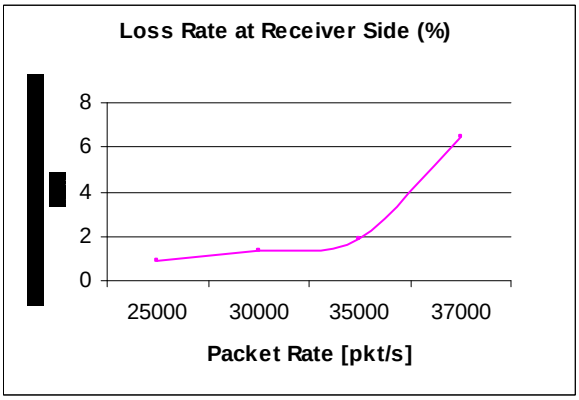



**Figure 32: Loss rate (%) trend over Windows platform (distributed and log at receiver side)**

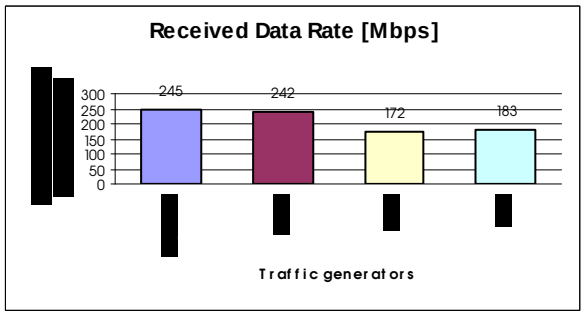



**Figure 33: Comparative analysis of received data rate over Windows devices (distributed and log at receiver side)**

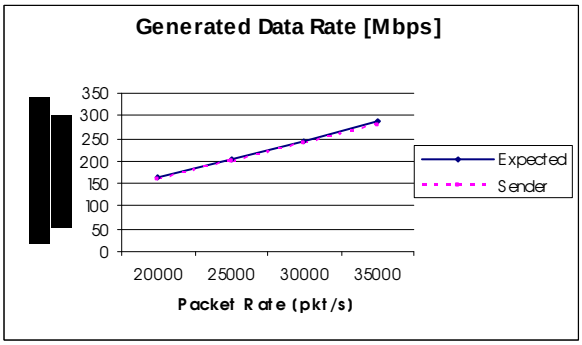



**Figure 34: Generated data rate over Windows platforms (distributed and log at sender and receiver side)**

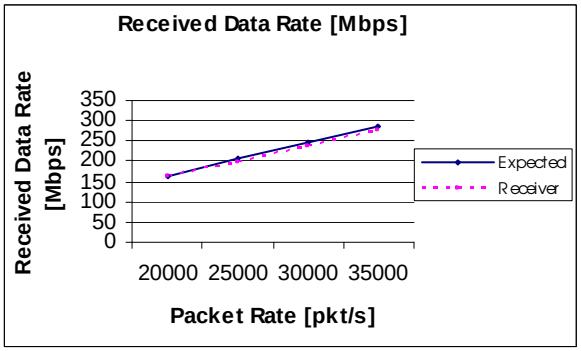



**Figure 35: Received data rate over Windows platforms (distributed and log at sender and receiver side)**

### 4.5.1. Linux Sender and Windows Receiver

Before presenting the comparative study we show the D-ITG performance when the pkt/s growths. Figure 36 shows the Windows D-ITG receiver data rate by using a Linux D-ITG sender. In this case with C=60000pkt/s, c=1024byte, t=60s D-ITG reaches a generated data rate equal to 483Mbps (with an error equal to 1.6%).

After D-ITG multiplatform analysis, Figure 38 shows a comparative analysis among traffic generators. Also in the multiplatform environment D-ITG presents the best performance. In a heterogeneous network scenario where a high number of Operating Systems are present, we believe that this feature is very interesting.

### 4.5.2. Windows Sender and Linux Receiver

Figure 39 shows the Linux D-ITG receiver data rate by using a Windows D-ITG receiver. In this case with C=30000pkt/s, c=1024byte, t=60s D-ITG reaches a generated data rate equal to 483Mbps (with an error equal to 1.6%). Also in this case D-ITG shows the best performance.

In the previous subsections a complete comparative analysis is described. We presented only the results related to a single combination of *packet rate* (C) and *packet size* (c). We carried out a similar experimentation for different combination of C and c: D-ITG showed the best performance in these cases too. In the presented combination of c and C we experimented the highest D-ITG performance.

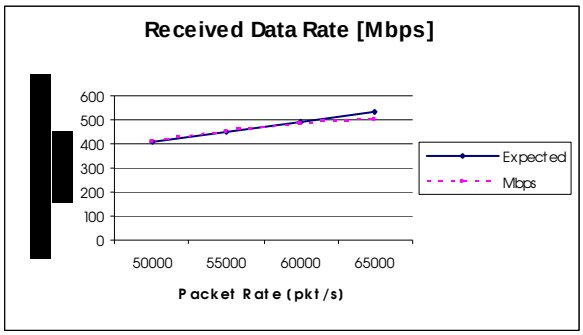



**Figure 36: Received data rate with Linux Sender and Windows Receiver (distributed and log at sender and receiver side)**

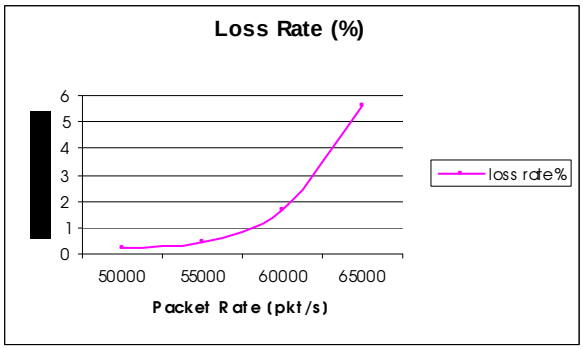



**Figure 37: Loss rate (%) with Linux Sender and Windows Receiver (distributed and log at sender and receiver side)**

## 6. Conclusions and directions for future works

In this paper we presented a traffic generation platform that we called D-ITG and an innovative protocol for traffic specification and experiment control. We called this protocol TSP, Traffic Specification Protocol. The platform architecture and each component of our novel proposal have been presented: ITGSend, ITGRecv, ITGLog and ITGManager. After the architectural details, we presented all D-ITG features and experimental results on generated and received data rate. D-ITG showed innovative characteristics when it is compared with other widely used traffic generators. An experimental analysis has been conducted over Linux and Windows platform: D-ITG showed the highest performance over both platforms and it was able to generate at high transfer rate with high values of packet size. Furthermore, D-ITG Linux implementation showed better performance than D-ITG Windows implementation.

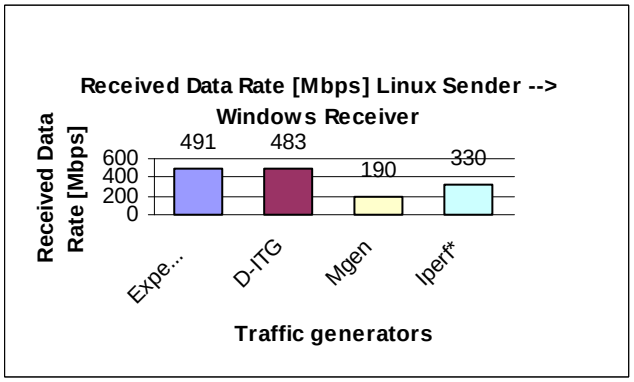



**Figure 38: Comparative analysis of received data rate with Linux Sender and Windows Receiver (distributed and log at sender and receiver side)**

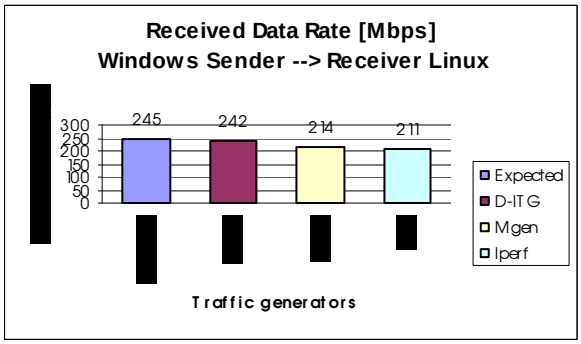



**Figure 39: Comparative analysis of received data rate with Windows Sender and Linux Receiver (distributed and log at sender and receiver side)**

D-ITG showed almost the same performance both in local and distributed environments. Finally, taking into account the received and generated data rate and the comparative analysis reported in Table 3, we believe that D-ITG shows interesting properties with respect to other traffic generators. D-ITG is currently downloadable and freely available at www.grid.unina.it/software/ITG and, to our knowledge, in terms of software architecture, *modus operandi* and results, no other similar platforms are available. Currently D-ITG is running over Linux Familiar platform too. We developed this porting because we believe that in a heterogeneous wireless scenario it is essential to have a tool running over PDA or Palm platforms, in order to understand network behavior. As far as this last point we are working on the porting on Win CE and PocketPC platforms. We developed a mechanism for authentication between sender and receiver. This feature permits to naturally extend our work in a web based scenario. We imagine a framework where people, by downloading our D-ITG receiver over their own devices and by means of a web application, can use our hardware platform and our traffic generation server for

checking networks or devices. Finally, the possibility to remotely control traffic generation enables to conduct experiments in the field of dynamic and automatic network configuration.

## Acknowledgements

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] L. Breslau, D. Estrin, K. Fall, S. Floyd, J. Heidemann, A. Helmy, P. Huang, S. McCanne, K. Varadhan, Y. Xu, and H. Yu. "Advances in Network Simulation". *IEEE Computer*, 33 (5), pp. 59-67, May 2000

[2] V. Paxson, S. Floyd “Why we don’t know how to simulate the internet”, In Proceedings of the 1997 Winter Simulation Conference, SCS (December 1997)

[3] S. Floyd, V. Paxson "Difficulties in simulating the Internet. ACM/IEEE Transactions on Networking, 9(4):392–403, February 2001.

[4] http://www.grid.unina.it/software/ITG

[5] A. Pescapè, M. D'Arienzo, S. P. Romano, M. Esposito, S. Avallone, G. Ventre, ``Mtools" - IEEE Network, Software Tools for Networking 2002, Vol. 16 No. 5 pag. 3. ISSN 0890-80445

[6] http://www.atm.tut.fi/rude

[7] http://mgen.pf.itd.nrl.navy.mil

[8] http://dast.nlanr.net/Projects/Iperf/

[9] http://www.citi.umich.edu/projects/qbone/generator.html

[10] 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

[11] A. Pescapè, D. Emma, G. Ventre, “Analysis and experimentation of an open distributed platform for synthetic traffic generation”, Accepted for publication at 10th IEEE FTDCS 2004 - May 2004 - China

[12] A. Pescapè, S. Avallone, G. Ventre ``Distributed Internet Traffic Generator (D-ITG): analysis and experimentation over heterogeneous networks", accepted poster at ICNP 2003

[13] 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

[14] S. J. Leffler, M. K. McKusick, M. J. Karels and J. S. Quarterman. "The Design and Implementation of the 4.3BSD UNIX Operating System," Addison-Wesley, 1989.

[15] C. Petzold, “Programming Windows, Fifth Edition”, Microsoft Press, published 11/11/1998, ISBN 1-57231-995-X

[16] C. Adams, “The Simple Public-Key GSS-API Mechanism (SPKM)”, rfc 2025, October 1996

| Type | Description | Type | Description | Type | Description |
|---|---|---|---|---|---|
| 1 | ***Connection request***: the sender requests a connection | 6 | ***Flow close acknowledgement***: the receiver acknowledges the end of a flow generation | 11 | ***Connection close request***: the sender requests to close the connection |
| 2 | ***Connection acknowledgement***: the receiver accepts the connection request | 7 | ***Connection close acknowledgement***: the receiver acknowledges closing the connection | 12 | ***Log configuration***: the sender sends to the receiver information on the log server configuration |
| 3 | ***Flow generation request***: the sender requests the permission to generate a flow | 8 | ***Discovery request***: the sender tests the receiver | 13 | ***Log configuration acknowledgement***: the receiver acknowledges the log configuration packet. |
| 4 | ***Flow generation close***: the sender informs the receiver about the end of a flow generation | 9 | ***Discovery reply***: the receiver replies to a sender discovery | 14 | ***Error 1***: unable to accept flow generation request, the specified port is unavailable |
| 5 | ***Flow generation acknowledgement***: the receiver grants the permission to generate a flow | 10 | ***Crypto***: an encrypted information is sent for authentication purpose | 15 | ***Error 2***: receiver authentication failed |

**Table 1: Description of TSP racket**

| | | | |
|---|---|---|---|
| **Dest Port** | **Type** | 3 | 12 |
| | **Description** | Port where the receiver will listen for traffic | Port where the log server listens for log information |
| **Protocol** | **Type** | 3 | 12 |
| | **Description** | Protocol type (UDP, TCP or ICMP) | Transport Protocol used to communicate with the log server |
| **Flow Id** | **Type** | 3, 4, 5, 6 | |
| | **Description** | Identifier of the generated flow | |
| **Dest IP** | **Type** | 3 | 12 |
| | **Description** | Receiver IP address | Log server IP address |
| **Application Layer Protocol** | **Type** | 3 | |
| | **Description** | Simulated application layer protocol | |
| **Crypto** | **Type** | 10 | |
| | **Description** | Encrypted information needed for authentication | |
| **File name** | **Type** | 12 | |
| | **Description** | Log file name | |

**Table 2: Description of fields**

| Traffic Generators | Operating Systems | | | Protocols | | | | | | Options | | | | | | Operative mode | | | Logging Phase | | | Meters Type | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | LINUX | Windows | Unix like | UDP | TCP | ICMP | Telnet | VoIP | DNS | TTL | TOS | Priority | Seed | Duration | Delay | Single Flow | Remote | Multiple Flow | Sender | Receiver | Remote | OWDM | RTTM |
| D-ITG | ✓ | ✓ | | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ |
| RUDE/CRUDE | ✓ | | ✓ | ✓ | | | | | | | | ✓ | | ✓ | | ✓ | | | | ✓ | | ✓ | |
| MGEN | ✓ | ✓ | ✓ | ✓ | | | | | | ✓ | ✓ | | | ✓ | | ✓ | | ✓ | | ✓ | | ✓ | |
| TG2 | ✓ | | ✓ | ✓ | ✓ | | | | | ✓ | ✓ | | | ✓ | | ✓ | | | ✓ | ✓ | | ✓ | |
| Iperf | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | | ✓ | | | | ✓ | |
| NetProbe | ✓ | | ✓ | ✓ | | | | | | | | | | | | ✓ | | ✓ | ✓ | | | | ✓ |
| TfGen | | ✓ | | ✓ | | | | | | | | | | | | ✓ | | | ✓ | | | | |
| Traffic | ✓ | ✓ | ✓ | ✓ | ✓ | | | | | | | | | ✓ | | ✓ | | ✓ | | | | | |
| MTOOLS | ✓ | | ✓ | ✓ | | | | | | | ✓ | | ✓ | ✓ | ✓ | ✓ | | ✓ | ✓ | ✓ | | ✓ | ✓ |
| UDPGenerator | ✓ | | ✓ | ✓ | | | | | | | | | | | | ✓ | | | | ✓ | | ✓ | |

| Traffic Generators | IDT | | | | | | | | PS | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Pareto | Constant | Uniform | Exponential | Cauchy | Normal | Gamma | Poisson | Pareto | Constant | Uniform | Exponential | Cauchy | Normal | Gamma | Poisson | Incremental |
| D-ITG | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | |
| RUDE/CRUDE | | ✓ | | | | | | | | ✓ | | | | | | | |
| MGEN | | ✓ | | | | | | ✓ | | ✓ | | | | | | | |
| TG2 | | ✓ | ✓ | ✓ | | | | | | ✓ | ✓ | ✓ | | | | | |
| Iperf | | ✓ | | | | | | | | ✓ | | | | | | | |
| NetProbe | | ✓ | | | | | | | | ✓ | | | | | | | |
| TfGen | | ✓ | | | | | | | | ✓ | | | | | | | |
| Traffic | | ✓ | ✓ | | | | | | | ✓ | ✓ | | | | | | ✓ |
| MTOOLS | ✓ | ✓ | ✓ | ✓ | | | | | ✓ | ✓ | ✓ | ✓ | | | | | |
| UDPGenerator | | ✓ | | | | | | | | ✓ | | | | | | | |

**Table 3. Traffic Generators: Comparative Schema**

| | |
|---|---|
| **Hardware Details** | Intel Pentium 4 2,6 GHz - CPU Cache 512 |
| | RAM: 1024 MB |
| | Hard Disk: Maxtor 6Y080L0 (Fast ATA/Enhanced IDE Compatible, Ultra ATA/133 Data Transfer Speed, 2MB Cache Buffer, Quiet Drive Technology, 100% FDB motors) |
| **Networks Details** | 2 PCs with a Gigabit Ethernet back-to-back connection |
| | Ethernet Controller: 3Com Gigabit LOM (3c940) |
| **Software Details** | **Linux:** Linux Mandrake 9.1 with kernel 2.4.21-013mdk and Linux Red Hat 9 with kernel 2.4.22.<br>**Windows:** Windows XP Professional 2002, Service Pack 1. |
| **Experiment Duration** | T = 60 s |
| **Traffic Details** | CBR, Constant Bit Rate |
| | Protocol: UDP |
| | C = packets per second (pps or pkt/s) |
| | c = Packet Size (byte) |

**Table 4: Experiment parameters**

 ISBN: 1-56555-284-9