Real-Time TCP/IP Analysis with Common Hardware

Dario Rossi
Politecnico di Torino – Dipartimento di Elettronica
dario.rossi@polito.it

Marco Mellia
Politecnico di Torino – Dipartimento di Elettronica
marco.mellia@polito.it

***Abstract*— Traffic measurement represents an indispensable and valuable tool for the analysis of nowadays telecommunication networks. Moreover, it is desirable for traffic measurement and analysis to be both continuous and persistent, since only these joint requirements allow to track important changes on the traffic pattern. On the other hand, transmission links bandwidth keep improving, at a seemingly inexorable rate: therefore, the analysis of the traffic is becoming more complex than ever. This paper focuses on the description and the benchmarking of a network traffic analyzer, called `Tstat`, able to process real-time traffic further providing i) several advanced measurement indexes of transport layer protocols and ii) ever-lasting monitoring capabilities. Particularly, our aim is to assess what kind of links, and under which load, can be continuously and persistently monitored without compromising the complexity of the traffic analysis that has to be performed.**

I. Introduction

Starting from the works of Danzig [1], and Paxons [2], the interest in data collection, measurement and analysis of Internet traffic, aimed at characterizing either the network or the user behavior, increased steadily. More recently, several research groups [3], [4], [5] have focused on joint measurements of IP packets and TCP flows, and on the network monitoring in itself. Generally speaking, traffic characterization can be performed by means of active measurements (i.e., by sending and receiving probe traffic), or by means of passive measurements (i.e., by collecting packets flowing through networks links) or possibly by combining both these methodologies. While active probing possibly allows to obtain very detailed information on specific performance index, passive measurements allow to characterize network and user behavior without modifying their activities and status, allowing researchers to acquire deeper insight into TCP/IP traffic. Among the various passive traffic characterization project, the two most famous are probably OCX-mon [6] from CAIDA, the Cooperative Association for Internet Data Analysis, and IPMON [7] from SprintLabs.

The layered structure of the TCP/IP protocol suite requires the analysis of traffic at least at the IP (network), TCP/UDP (transport), and possibly Application/User–behavior (application) layers. Given today traffic pattern in the Internet, which accounts for a large majority of the traffic being transfered by TCP flows, it is particularly interesting to analyze and measure transport layer indexes, ofter termed also "layer-4" indexes. Given the complexity of layer-4 protocols, and of TCP in particular, this requires a lot of computational power and memory space. Indeed, a single measurement at the transport layer usually requires to build and maintain the flow status for the whole flow lifetime, with a great deal of additional complexity compared to pure packet layer analysis.

In this paper, we consider *open source* and *passive* analysis tools able to produce statistics at the transport layer. Among the available software, we select `Tstat` [8], which has been developed at the Politecnico di Torino in the past years, and whose development has been motivated by the lack of automatic tools able to produce statistical data from passive network measurement. Though we select a specific software tool, we argue that the insight gathered are valid to a more general extent: indeed, `Tstat` performs typical operations (such as flow status tracking, measurement computation, and statistical data generation) that every layer-4 measurement tool must perform. Furthermore, several other applications actually require to track transport layer status, such as, e.g., Intrusion Detection Systems (IDS). For example, tools such as `Bro` [9] and `snort` [10], are very similar in their operative details despite having rather different goals: indeed, to detect misbehaving or intruding users, IDSs need to keep track and correlate packets to identify streams that are not bound to a permitted network application. Similarly, "layer-4 switching" [11] architectures (such as load balancers, firewalls, NAT boxes or caching system) must perform per-flow tracking operation, which are very similar to the one a "layer-4 measurement box" performs.

The goal of this paper is therefore to assess the performance of `Tstat`, but also to derive some useful indication on scalability problems, and bottleneck identification when using software-based tools to perform transport layer operations. By running several benchmarking tests using different real traffic traces, we show that off-the-shelf hardware can easily be used to perform real-time layer-4 traffic analysis at Gigabit speed. After briefly describing `Tstat` capabilities and the internal operation performed in Section II, we present in Section III the results of a thorough benchmarking involving parameters such as the compiler, the I/O generation, the type of link under analysis. The analysis of the performance, mainly described in terms of memory usage, CPU utilization and packet processing rate, will allow to gather the conclusion drawn in Section IV.

1-4244-0355-3/06/$20.00 (c) 2006 IEEE

This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the IEEE ICC 2006 proceedings.

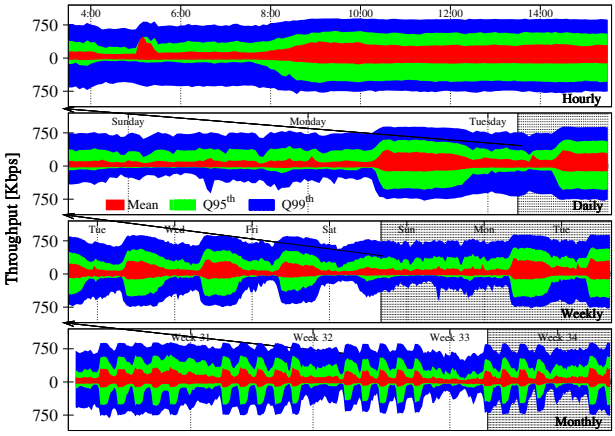

Fig. 1. RRD Output: Time-Varying Throughput Distribution

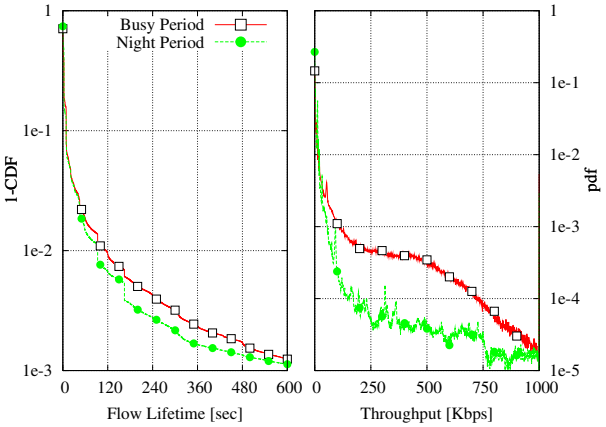

Fig. 2. Histogram Output: Throughput and Flow Lifetime Distributions

II. TSTAT

Started as evolution of `TCPtrace` [12], `Tstat` analyzes either real-time captured packet traces, using common PC hardware or more sophisticated ad-hoc cards such as the DAG cards [13]. The tool provides several measurements at the network layer, but it mainly performs analysis of transport layer protocols. Assuming that both forward and backward stream of packets are observed, `Tstat` can correlate them to infer advanced measurement indexes. If both TCP data and ACK segments can be analyzed, `Tstat` rebuilds each TCP flow status by looking at the TCP header in the forward and backward packet flows. The TCP flow analysis allows the derivation of novel statistics (such as the congestion window size, out-of-sequence and duplicated segments classification, etc.) which are collected distinguishing both between clients and servers, (i.e., hosts that actively open a connection and hosts that reply to the connection request) and also identifying internal and external hosts (i.e., hosts located inside or outside the measurement point). This methodology has been applied to infer traffic characteristics at the transport layer, and in particular to the TCP and RTP/RTCP protocols. A detailed presentation of `Tstat` and list of all monitored performance indexes is available from [14].

A. Tstat at a Glance

`Tstat` builds histograms of measured indexes, dumping the collected distribution periodically, rather than dumping each single measured datum. The data produced by the on-line statistical analysis is ready to be visualized as either time plots or aggregated plots over different time spans. A complete transport layer log, which is useful for post-processing purposes, tracks all analyzed layer-4 flows by including all performance indexes for later post-processing. Finally, `Tstat` has been integrated with `RRDtool` [15]: the whole measurement indexes can be stored as a Round Robin Database (RRD). Since RRD has fixed size, this allows ever-lasting live-capture without compromising the statistical relevance of the data, as it can be gathered by browsing [14].

In order to quickly overview `Tstat` processing capabilities, Figure 1 shows the time evolution of the TCP-flow throughput distribution on a backbone link that is persistently monitored. In order to make plots readable, only the average, 95^{th} and 99^{th} percentiles are explicitly reported; moreover, both link directions are displayed in a single plot, using either positive or negative values for a given direction. Measurements are evaluated with different granularities over different timescales: each point corresponds to a 5 minute window in the hourly plot (at the top), a 30 minute window for both daily and weekly plots and a 2 hour interval in the monthly plot (at the bottom). The plot of the entire month clearly shows a night-and-day trend, which is mainly driven by the link load; the trend is less evident at finer time scales, allowing to identify stationary periods during which `Tstat` allows to perform advanced statistical characterization.

The entire monthly dataset has been used to produce results reported in Figure 2, which reports the survival function (1-CDF) of the flow lifetime (left plot) and the probability distribution function (pdf) of the per-flow throughput (right plot). The dataset has been discriminated on busy (10:00–15:00) versus off-peak (22:00–6:00) periods, which roughly correspond to the concave and convex portions of Figure 1. Let us notice from the left plot of Figure 2 that 99% of TCP flows lasts less than 2 minutes, whereas this amount increases to 99.7% when considering flows shorter than 5 minutes. This observation has important impact on the workload a layer-4 box has to face.

B. Anatomy of a Traffic Analyzer

`Tstat` has been designed in a very modular way, so that it is easy to integrate modules for new measurements; it has also been designed to efficiently run on common hardware, allowing both off-line and real-time traffic measurements. Multi-threading programming is used to split the program execution in several tasks, that may run in parallel, exploiting Operating System multitasking capabilities as well as multiprocessor hardware. Figure 3 shows a schematic diagram highlighting the main building blocks implemented in `Tstat`, that we

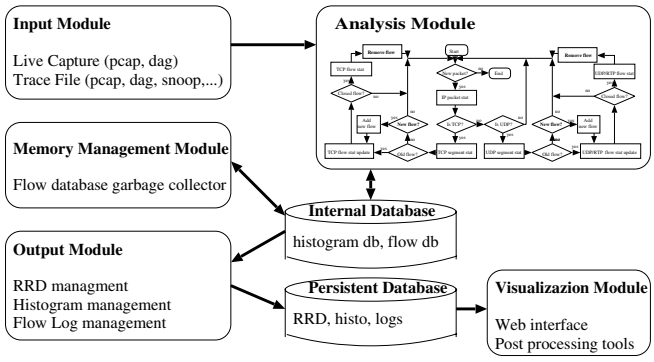

Fig. 3. Tstat Functional Modules and Databases.

describe in the following.

Input Module. The Input Module allows to either capture packets in real-time, or to read trace files already stored on disk, where by trace we mean an uninterrupted bidirectional flow of dumped packet headers. The real-time capture supports classic `pcap` [16] interface with filtering support, as well as the DAG capture API [13]; the latter allows to use dedicated hardware, designed to perform live trace capture on high speed links with minimum CPU overhead. Conversely, the trace file routines are directly derived from `TCPtrace`, and allow to read traces in several formats.

Analysis Module. The core features of Tstat are implemented in the Analysis Module, whose basic assumption of is that both the forward and backward packets are exposed, so that bidirectional TCP connections can be analyzed. Each packet is first analyzed by the IP module, which takes care of all statistics at the packet layer; then, UDP or TCP per-segment statistics are possibly computed. In case the packet is a TCP segment, it goes through the TCP flow analyzer module, which is the most complex part of `Tstat`. First, the module decides if the segment belongs to an already identified flow, using the classic 5-tuple, i.e., IP source/destination addresses, IP protocol type and TCP source/destination ports. Otherwise, a new flow is identified only provided that the `SYN` flag is set, as `Tstat` processes only *complete* flows. Once the segment is successfully identified, all state variables related to the flow are updated. Finally, in case the segment correctly closes a flows, all flow-layer statistics are updated and the flow data structures are released.

Similarly, UDP segments go through the UDP flow analyzer module where, given that UDP is a connectionless protocol, no signaling can be exploited to identify new flows. Therefore, if no already tracked flow can be associated to the UDP segment under analysis, a new flow is created and tracked. Besides, additional statistics are available for RTP/RTCP flows, and other specific-traffic (e.g., multimedia, peer-2-peer) modules can easily be integrated into the architecture.

Internal Database. Statistics collected by the Analysis Module are stored internally using the Internal Database. Two main databases are used: the "histogram database" stores all collected statistics, and the "flow database" stores temporary data used to perform transport layer measurement. Each histogram is a simple static array of integer that counts the number of times a given measured index assumes values in a given range. The flow database is the core of the layer-4 measurements, as it stores all the state variable variables related to every ongoing flow. A very efficient memory management core has been implemented, based on reuse techniques that optimize memory usage.

Memory Management Module. The Memory Management Module is a separate thread that is periodically activated to perform garbage collection of flows stored in the flow database. A garbage collection procedure is required to free up memory allocated to incomplete flows: if no segments of a given flow are observed in between two garbage collections, the flow is considered complete, and the flow status memory used is put back in the reuse list. This is the only possible flow termination definition when considering UDP/RTP flows, given their connectionless properties; but this is also possible considering TCP flows, when for example no connection tear-down sequence is performed by the hosts (e.g., due to host/network failure, or due to misbehaving hosts). The inactivity timer is set by default to 5 minutes, which has been shown to be a safe choice according to [17].

Output Module. The Output Module is a separate thread, periodically activated to store measurements in the persistent database. It manages the round robin database by converting information stored in the internal histogram database, and optionally dumps the histograms status on files before resetting them. Finally, an asynchronous routine is used to update the flow-level log, where the computed measurement indexes of successfully tracked flows are added at each flow end.

Persistent Database. The Persistent Database is a collection of files storing all measured indexes. RRD, histogram database and layer-4 log database are then available to offer off-line post-processing of information and to derive more complex measurement indexes.

Visualization Module. The Visualization Module, a part of the `Tstat` design, is a collection of post-processing scripts that allow to easily extract information from the Persistent Database; the most notable visualization module is a CGI Web interface that allows to present historical data exploiting the RRD capabilities.

As previously stated, `Tstat` has been engineered to run in real-time: therefore both memory and CPU constraints have been carefully addressed in its design. In particular, `Tstat` avoid to relies to the OS memory management library, in particular considering the `free()` function. Indeed, it is well known that memory management operation are rather expensive, and that it is not guaranteed that chunks of memory freed by the `free()` function are actually made available to the OS for later reuse. Therefore `Tstat` implements internal memory reuse techniques that optimize memory usage and avoid expensive library calls for memory allocation and deallocation. Therefore, after the initial transient period during which structures are allocated on the fly, the memory usage

becomes stable.

Considering flow identification, Tstat uses hash function to identify if the packet currently under analysis belongs to any of the tracked flows. The hash key is derived from the 5-tuple that identifies the flow itself, and collisions are handled by chaining (i.e., through the use of linked lists). While this structure does not guarantee good worst case performance, it represents a good balance in term of average performance and simplicity, as we will show later. Indeed, while balanced binary search trees do guarantee better worst case performance, nevertheless we believe that i) the complexity in managing them may actually overcome the worst-case benefits and ii) such structures may be unnecessary for the average case, where the maintenance overhead may further limit the performance.

III. Performance Evaluation

This section deals with the evaluation of the processing capabilities of layer-4 analysis tools, considering Tstat as an example. In the benchmarking setup we inspect the impact of several parameters on the performance. Controlled parameters range from the compiler flavor and optimization options to the traffic measurement point, load and storage trace format. Performance indexes are described in terms of both user-centric indexes (such as CPU and memory usage, packets and flows processing rates) as well as internal algorithm state variable. In order to stress Tstat to the limit of its traffic analysis capabilities we consider off-line processing only: as we will show later, this choice will enable us to gather useful, otherwise unknowable, insights on the limit of on-line performance.

A. Testbed Description

All experiments have been performed on a PC, running a GNU-Linux operating system with kernel version 2.4.29, sporting two Intel Xeon CPUs clocked at 2.40GHz with 512 KB cache, equipped with 3 GB of RAM memory and several IDE 7200-rpm hard-disks.

Performance Metrics

For each test execution, we report the following performance indexes:

- *general-purpose:* elapsed execution time, instantaneous CPU utilization, RAM memory occupation;
- *output-oriented:* packet processing rate π, flow processing rate ϕ and output storage size;
- *internal-state:* number of iterations spent in the hash search routine and average length of a TCP flow lookup.

These metrics allows us to inspect different aspects of the traffic processing task: we will not only i) assess which kind of networks, and under which level of congestion it is possible to painlessly monitor with common hardware, but also ii) state some software design guidelines, such as the impact of data structure choice.

Off-line Performance

Though, as previously mentioned, Tstat performs on-line analysis of traffic, our benchmarking will only involve *off-line* analysis of real packet-level traces. This is due, first, to allow running batch of tests on the same input, while inspecting input parameters to the software. Second, but not less important, our experience with on-line analysis showed us that the incoming traffic rate, even when considering backbone traffic, never saturated Tstat processing capabilities.

Therefore, if τ off-line seconds are needed to process T seconds worth of captured traffic, we may conclude that the on-line processing could accommodate about T/τ times more traffic. While we acknowledge that this is a very rough estimate, nevertheless we point out that artificially generating a T/τ more traffic would yield even rougher estimates, since, given the intrinsic difficulty of the traffic generation, the resulting traffic pattern would be hardly representative of a real situation.

Input Network Traffic

We present results obtained through real traffic traces collected from different networks. The first one is a packet-level trace gathered from the Abilene Internet backbone, publicly available on the NLANR Web site [18]. We will refer to this dataset as ABILENE throughout the paper. The trace has been collected on June 1st, 2004 at the OC192c Packet-over-SONET link from Internet2 Indianapolis node toward Kansas City. Peculiar of this trace is the large presence of very high-speed transfers of long files, which generates very short spikes of intense load above 2.5 Gbps. Similarly, quite anomalous traffic patters, e.g., large presence of port scan attempt, are present. These are due to the experimental traffic ABILENE carries. Both previously described characteristics form a stress scenario when considering layer-4 processing.

The second measurement point is located on the GARR backbone network [19], the nation-wide ISP for research and educational centers, that we permanently monitor using Tstat. Results shown early in Figure 1 and Figure 2 refer to this measurement point. The monitored link is a OC48 Packet-over-SONET from Milano-1 to Milano-2 nodes. We selected a trace collected the 15th of April 2005, in the following denoted by GARR. GARR traffic trace is representative of typical today traffic in which business, home and research traffic share the same infrastructure.

Both GARR and ABILENE traces correspond to about two-hours long period and include 80 Bytes of packet headers only; GARR traces occupy 46 GB of storage whereas ABILENE amount to 36 GB of compressed data (141 GB uncompressed[1]). Moreover, traces show different *routing symmetry*: while the GARR trace reflects almost 100% of traffic symmetry, only 46% of ABILENE traffic is symmetric; this is of particular relevance, since Tstat relies on the observation of both DATA and ACK segments to track TCP flow evolution. But if only "half" flow is observed, Tstat starts tracking the flow anyway, consuming internal database resources until the

[1]Due to the size of ABILENE dataset, we were able to manage the trace only in compressed format.

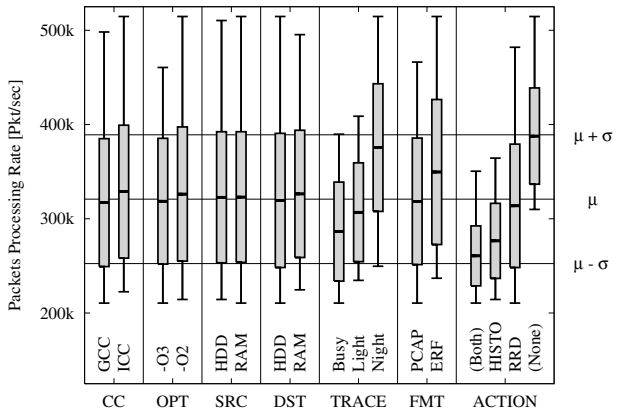

Fig. 4. Tstat Performance: Sensitivity to Different Parameters

flow is discarded by a garbage collection operation. Finally, the average load is quite different for the two traces: on average, GARR traffic rate is about 36 Kpps or 177 Mbps. ABILENE traffic rate is more than 6 times higher, resulting in an average rate slightly above 225 Kpps or 1.12 Gbps.

The third measurement point is located at the edge of the of Politecnico campus LAN, which behaves thus as an Internet stub. It is an OC4 AAL5 ATM link from Politecnico to the GARR POP in Torino. We selected three different two-hour long traces, starting namely at:

- POLINIGHT: Sunday the 23th of May 2004, at 4:00
- POLILIGHT: Monday the 19th of April 2004, at 18:00
- POLIBUSY: Monday the 10th of May 2004, at 10:00

These three traces –which correspond to a night, an after-work and a busy-time period respectively– account for increasing levels of congestion, allowing to gather the impact of the different kind of traffic flowing through the same network.

B. Sensitivity Analysis

As a first experiment, we inspected the *sensitivity* of the traffic processing time to several external conditions. Specifically, considering as performance metric the packet processing rate π, we performed a batch of tests considering the following parameters:

TABLE I
PARAMETERS SENSITIVITY

Parameter	Meaning	Inspected Values
CC	C Compiler	{ GCC, ICC }
OPT	Compiler Options	{ O2, O3 }
SRC	Input Device	{ RAM, HDD }
DST	Output Device	{ RAM, HDD }
TRACE	Trace Load	{ NIGHT, LIGHT, BUSY }
FMT	Trace Format	{ `pcap`, `erf` }
ACTION	Tstat Output	{ None, Histo, RRD, Both }

Each test has been repeated three times to get average results, for a total number of 1152 tests. As reported in Table II, we varied the compiler (CC), choosing either the GNU C Compiler version 3.2.2 (GCC) as well as the Intel C Compiler version 8.1 (ICC). For either compiler, we tested several optimization options (OPT) beside the raw optimization level (i.e., O2, O3): after several preliminary tests we ended up using the following compiling options `-finline-functions -funroll-all-loops -march=pentium4 -mfpmath=sse`, which yielded the best performance. Moreover, we investigated the impact of the trace storage *device* (SRC, DST): in our tests the input was either stored on a non-volatile `ext3` filesystem (HDD), or on a `raw` device created using a Ramdisk (RAM). Similarly, trace storage *format* (i.e., `pcap` or `erf`) has been also considered (TRACE). Only POLI traffic traces have been taken into account to reflect increasing levels of network congestion (i.e., NIGHT, LIGHT, BUSY). Finally, the various combinations of processing output (ACTION) (i.e., None, Histogram, RRD, Both) were directed to either a `ext3`-formatted hard-disk or to an `ext3` RAM device.

Figure 4 reports the sensitivity results in terms of the achieved packet processing rate for a given parameter value, obtained aggregating the tests irrespectively of all other parameter values. Denoting by μ and σ the mean and standard deviation of the packet processing rate π respectively, the line inside the box reports μ, the boxes delimit $\mu \pm \sigma$, and the lines outside the box indicate the maximum and minimum values achieved. On the right axis, the average μ and bounding box $\mu \pm \sigma$ values achieved over all the tests are reported as reference values.

In a glance, `Tstat` performance ranges from a minimum of 210 Kpps up to 530 Kpps. Considering typical packet size, this is equivalent to about 1.7 Gbps - 4.2 Gbps range. This is in accordance with the previously mentioned fact that actual load of OC48 backbone traffic poses no problem to real-time layer-4 measurement.

In more details, considering compiler version and optimization options (CC, OPT), Figure 4 allows to conclude that they play a marginal (though non negligible) role: the use of ICC with optimization level two (-O2) may bring a 3.6% gain with respect to the other compiler settings.

Considering Ramdisk vs HDD storage (SRC, DST), the use of Ramdisk brings almost negligible advantages in most cases, but in some tests performance actually suffered rather than benefiting of its use. On the input side, this can be explained taking into account Linux aggressive caching policy: it is evident that the operating system is already performing everything possible to leverage the input task, thus an explicit caching by the use of RAM input storage is actually useless. Considering the output generation, the output data structure produced by `Tstat` are stored on files. This implies the need for a filesystem, rather than a raw device (i.e., a device merely storing binary data), which introduce an additional overhead. Second, the amount of output is much smaller than the original packet trace. Therefore the O.S. caching algorithms almost overcome the possible output bottleneck. However, while the cache consolidation operations are performed via DMA by the Hard Disk Controller, the same operations are carried over by the CPU when Ramdisk is involved, thus stealing CPU cycles

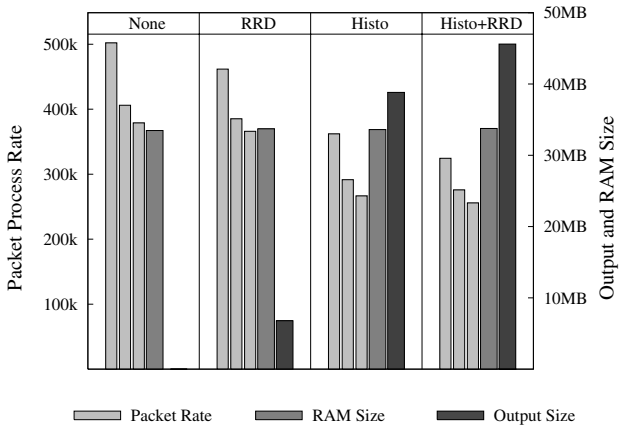

Fig. 5. Best-Case Performance for Different Input Traffic and Output Types

to the packet processing task.

Considering the trace type (TRACE), we observe that as intuitively expected, the impact of the specific traffic is also very important: NIGHT traces are processed 1.26 and 1.32 times faster than LIGHT and BUSY traces respectively: indeed, the higher the traffic rate, the larger the amount of flows have to be analyzed, the larger the internal data structures as well.

Considering the trace file format (FMT), we observe that it deeply affects the performance: the use of the `erf` format bring a gain of about 9.7% with respect to `pcap`, which is essentially due to smaller library overhead.

Finally, considering the output produced by `Tstat` (ACTION), it clearly impacts the processing performance. This is mainly tied to the amount of data that has to be stored: when no-output is required, the traffic is processed at the highest possible rate, while producing both RRD and histogram databases limits the overall processing rate.

C. The Cost of Edge Traffic Analysis

Based on the sensitivity results, we simplify the reminder of the analysis by restricting our attention to the best combination of options. We consider from now on ICC compiler, with -O2 optimization option, `erf` trace file format, and HDD storage for both input and output. We further investigate the impact of the TRACE type still considering traces collected at the edge node of our campus LAN. Additional results in terms of RAM utilization and output storage requirements will be investigated.

The results referring to experiments that comply to above selection criteria are reported on Figure 5: the picture shows the packet processing rate achieved averaging three different runs over the same trace (left y-axis) using light-gray, thin bars; bars refer to NIGHT, LIGHT, BUSY trace respectively. Total memory utilization and persistent database size averaged over all traces and runs (right y-axis) are reported using gray and black bars respectively. Finally, results are grouped to distinguish them according to the persistent database information recorded (ACTION).

Interesting indications can be gathered about the packet processing rates. First, it should be noticed that the overhead introduced by RRD is marginal, since performance of RRD vs Null output are very similar (the same holds for periodical Histogram only vs Both output). This is a very important observation, since it suggests that the RRD module leverages the storage issue, allowing ever-lasting monitoring, and it also scales very well with the traffic load. Periodically dumping the histogram database and logging each flow measurement has instead a larger impact on the performance, which decrease by about 20%. This is related to the overhead due to persistent storage maintenance on files, as previously noticed. Second, let us consider the case where all the processing computations are performed but the output is discarded (leftmost bars): there is a significant (i.e., on the order of 100 Kpps) difference between nightly and daily traffic, but the relative difference between nightly and heavily loaded network is limited, as it is still possible to process about 400 Kpps, or about 2 Gbps. This shows that the kind of traffic under analysis has a large impact, but the performance impact is limited.

Considering memory requirements, there is marginal difference respected to output type, since the same internal database is maintained. Indeed, memory usage depends on the traffic load, since the internal database size increase with the number of contemporary active flows: the lowest memory usage, corresponding to the NIGHTLY traffic is about 29 MB compared to the 38 MB of the BUSY period. We can then state that there is no particular memory requirement to track layer-4 measurement indexes on edge nodes.

Finally, considering the output size, we notice that it is unrelated to the specific input trace: indeed, while RRD storage amount is fixed *a priori*, the histogram output size depends on the actual trace *duration* rather than traffic mixture, given the periodical nature of the dump process. Therefore it is possible to easily predict the size of generated output.

D. The Cost of Backbone Traffic Analysis

While the previous section focused on the impact of different levels of congestion on the same edge-network traffic, in this section we extend our analysis by inspecting the processing performance of two different backbone traces, namely ABILENE and GARR. To simplify the analysis, we further limit the type of processing to either no-output, focusing on pure-processing performance, or RRD output, investigating what kind of link/traffic can be persistently monitored.

Table II reports detailed performance figures relative to both ABILENE and GARR traces: total elapsed time, pure CPU time, flow and packet processing rate (as usual, measured according to the total elapsed time), memory and CPU usage are reported. We define the processing *speed-up* as the ratio of the trace duration over the CPU time required to complete the analysis. Results confirm that `Tstat` painlessly processes GARR trace and that persistent ever-lasting monitoring is feasible, as in our personal experience `Tstat` never suffered performance problems during real-time monitoring of the GARR link. Indeed, we point out only 18 minutes were needed

TABLE II

FEASIBILITY ANALYSIS OF PERSISTENT BACKBONE LINK MONITORING

Backbone Trace	Output Type	π [Kpps]	ϕ [Kfps]	Elapsed [h:m:s]	CPU Time [h:m:s]	Speed-up	Memory [MB]	CPU %
GARR	Null	252.35	11.00	0:18:33	0:16:16	8.06	476	96.7
	RRD	233.86	10.20	0:20:01	0:16:32	7.93	478	98.0
ABILENE	Null	255.79	5.94	2:00:42	1:26:30	1.49	1047	74.0
	RRD	255.01	5.92	2:01:04	1:26:37	1.49	1049	74.0

to process more than two-hour worth of traffic, thus the processing was much faster than the real-time traffic arrival. Even in the ABILENE case, though the gain margin is scarcer since a significant portion of the CPU power has been devoted to the input *decompression*, `Tstat` processing is still faster than real-time traffic arrival. Indeed, by considering the pure CPU time devoted to the user-space process rather than the total elapsed time, a speed speed-up of about 1.5 can be envisioned. Notice that the use of dedicated capture cards (mandatory to monitor OC192 links) would further lessen the CPU from IO input processing, allowing for even more traffic to be processed.

Considering internal parameters, we are interested to double check the effectiveness of the simple hash routines used to manage the internal flow database. We measured therefore the average lookup rate, considering a hash table of 1,000,000 entry. Results show that the average lookup rate is about 1.32 considering GARR trace (which account for about 300,000 tracked flows per unit of time) and 3.27 considering ABILENE trace (which accounts for about 800,000 tracked flows); the former result implies that the use of balanced search trees would prove useless. Note that, in the ABILENE case, it could be sufficient to use larger hash table to reduce the average number of collisions, which has not been done in order to allow for fair cross-trace comparison.

IV. CONCLUSIONS

In this paper we presented a thorough performance evaluation of `Tstat`, a measurement tool which allows to derive performance indexes at the transport layer by tracking TCP/UDP flows during their lifetime.

By using real traffic traces captured from simple edge node up to backbone OC192 links, we evaluated the impact of several parameters, such as software design choice, compilation option, output generation overhead, etc. Results show that with common hardware it is possible to perform real-time measurement up to 2.5 Gbps traffic without performance issues, which opens the possibility of performing real-time traffic analysis on today commonly used backbone links.

We believe that the insight gathered can be extended to more general "layer-4" problems, such as the ones faced by intrusion detection systems, firewalls, load balancers, etc., given that operations required to perform flow tracking are very similar.

V. ACKNOWLEDGMENTS

This work was partly funded by the European Community "EuroNGI" Network of Excellence, and partly by the Italian MIUR PRIN project "MIMOSA". We also would like to thank the GARR for allowing us to monitor some of their backbone links, the Politecnico di Torino network facilities CESIT for their patience in supporting the monitoring at our campus LAN, and finally the people from NLANR for providing the research community their invaluable traces.

REFERENCES

[1] R. Caceres, P. Danzig, S. Jamin and D. Mitzel, "Characteristics of wide-area TCP/IP conversations," *In Proc. of ACM SIGCOMM'91*, Zurich, Switzerland, 1991, pp. 101–112.

[2] V. Paxons, "Empirically Derived Analytic Models of Wide-Area TCP Connections," *IEEE/ACM Transactions on Networking*, V. 2, NO. 4, 1994, pp: 316-336.

[3] S. Jaiswal, G. Iannaccone, C. Diot, J. Kurose and D. Towsley. "Inferring TCP Connection Characteristics Through Passive Measurements," *In Proc. of IEEE INFOCOM'04*, Hong Kong, March 2004, pp. 1582–1592.

[4] L. Li, M. Thottan, B. Yao and S. Paul, "Distributed Network Monitoring with Bounded Link Utilization in IP Networks," *In Proc. of IEEE INFOCOM'03*, San Francisco, Ca, March 2003, pp. 1189-1198.

[5] M. Jain and C. Dovrolis, "End-to-end Estimation of the Available Bandwidth Variation Range," *In Proc. of ACM SIGMETRICS'05*, Banff, Canada, June 2005.

[6] NLANR Measurement Group, "Towards a systemic understanding of the internet organism: a framework for the creation of a network analysis infrastructure", `http://moat.nlanr.net/`, 1998.

[7] S. A. T. Laboratories, Ip monitoring project (ipmon), `http://ipmon.sprintlabs.com/`, 2002.

[8] M. Mellia, R. Lo Cigno and F. Neri, "Measuring IP and TCP behavior on edge nodes with Tstat", *Computer Networks*, Vol. 47, No. 1, pp. 1–21, Jan. 2005.

[9] V. Paxson, "Bro: A system for detecting network intruders in real-time", *Computer Networks*, Vol. 31, No. 23, pp. 2435–2463, 1999.

[10] M. Roesch. "Snort: lightweight intrusion detection for network," *In Proc. of the 13th USENIX Systems Administration Conference (LISA'99)*, Seattle, WA, November 1999.

[11] V. Srinivasan, G. Varghese, S. Suri, M. Waldvogel, "Fast and scalable layer four switching", *In Proc. of ACM SIGCOMM'98*, Vancouver, BC, Canada, pp. 191-202, August 1998.

[12] S. Ostermann, Tcptrace Web site `http://www.tcptrace.org`, 2005.

[13] Endace Web site, `http://www.endace.com`, 2005.

[14] M. Mellia and D. Rossi, Tstat Web Site, `http://tstat.tlc.polito.it`, 2005.

[15] T. Oetiker, RRDtools Web site, `http://people.ee.ethz.ch/~oetiker/webtools/rrdtool`, 2005.

[16] S. McCanne, C. Leres, V. Jacobson, `pcap` Web site, `http://www.tcpdump.org`, 2005.

[17] G. Iannaccone, C. Diot, I. Graham, N. McKeown, "Monitoring very high speed links", *In Proc. of ACM Internet Measurement Workshop*, San Francisco, CA, 2001, pp.267-271.

[18] AbileneIII packet trace, `http://pma.nlanr.net/Special/ipls3.html`

[19] GARR Network, `http://www.noc.garr.it/mrtg/RT.TO1.garr.net`