Live Traffic Monitoring with Tstat: Capabilities and Experiences

A. Finamore[1], M. Mellia[1], M. Meo[1], M.M. Munafò[1], and D. Rossi[2]

[1] Politecnico di Torino
`lastname@tlc.polito.it`
[2] TELECOM ParisTech
`dario.rossi@enst.fr`

Abstract. Network monitoring has always played a key role in understanding telecommunication networks since the pioneering time of the Internet. Today, monitoring traffic has become a key element to characterize network usage and users' activities, to understand how complex applications work, to identify anomalous or malicious behaviors, etc. In this paper we present our experience in engineering and deploying Tstat, a passive monitoring tool that has been developed in the past ten years. Started as a scalable tool to continuously monitor packets that flow on a link, Tstat has evolved into a complex application that gives to network researchers and operators the possibility to derive extended and complex measurements. Tstat offers the capability to track traffic flows, it integrates advanced behavioral classifiers that identify the application that has generated a flow, and automatically derives performance indexes that allow to easily characterize both network usage and users' activity. After describing Tstat capabilities and internal design, in this paper we present some examples of measurements collected deploying Tstat at the edge of our campus network for the past years.

1 Introduction

The importance of traffic monitoring and analysis has always played a key role in understanding telecommunication networks. Since the early days of the Internet, several methodologies and tools have been engineered to perform traffic monitoring, to detect problems and to understand the network behavior and users' usage. Due to the availability of cheap, high-performing PCs and network interfaces, and driven by the explosive growth of the Internet, nowadays traffic monitoring is considered a crucial component of the network and a lot of advanced tools and methodologies are available to "monitor the Internet".

At a very high level, there exist two antagonist approaches to traffic measurement: *passive* and *active*. The *active* approach aims at interfering with a network to induce a measurable effect. For example, active approaches inject ad-hoc probe traffic or modify the network state, e.g., enforcing artificial network conditions. Active approaches induce a cause-effect relationship between the network and the traffic, triggering a measurable effect which is the goal of the

measurement itself. Active approaches are the core for a number of tasks such as operation and management (e.g., ICMP), network tomography (e.g., traceroute), delay and capacity measurement (e.g., ping, capprobe[1], pathchar[2]), controlled empirical studies (e.g., netem, dummynet[3]). In the *passive* approach the traffic exchanged on the network is merely observed as to infer some of its root properties, but taking care not to interfere with the observation. In this case, a passive "sniffer" is used to observe normal traffic; for example, at the physical-level this is done by means of passive optical splitter, at higher layers by means of replication of layer-2 frames. Copies of the traffic are then sent to an "analyzer machine" for processing. Passive monitoring applications are diverse and range from user characterization to intrusion detection, from traffic classification to policing, just to mention a few.

Over the years, several tools have been engineered to passively analyze traffic, each of which has some type of specialization. Wireshark[4], for instance, is a well-known protocol analyzer with several advanced analysis capabilities, which allow the user to interact with the collected packets through a GUI interface. Other tools like Snort[5], an intrusion detection system, or CoralReef[6], an advanced classification tool, are designed to work in real time and minimize the human interaction.

Tstat [7] is one of such passive analysis tools, developed in the past ten years by the Politecnico di Torino networking research group, and nowadays used by several researchers and network operator worldwide. Tstat started as evolution of TCPTRACE[8], which originally allowed to track single TCP flows and offered the user detailed statistics about each flow. After years of development, Tstat is now a scalable application that gives network researchers and operators the ability to derive an extended set of advanced measurements, focusing on an IP network. Tstat offers live and scalable traffic monitoring up to Gbps on off-the-shelf hardware. It implements traffic classification capabilities, including advanced behavioral classifiers, while presenting advanced performance characterization of both network usage and users' activities. In this paper, we report on the Tstat architecture and functionalities, illustrating through several examples its capabilities.

2   Tstat Features

Tstat is a traffic analyzer tool with a very flexible design: it consists of different modules that produce a wide set of statistics in different formats. Tstat is highly configurable at runtime, so that many features can be activated and deactivated on the fly, without interrupting the program. In the following, we first overview Tstat architecture, we then present its capabilities.

2.1   Data Structures and Workflow

Tstat is an IP network passive sniffer that processes packet-level traffic to produce both per packet and per flow statistics. The basic entity in Tstat is a
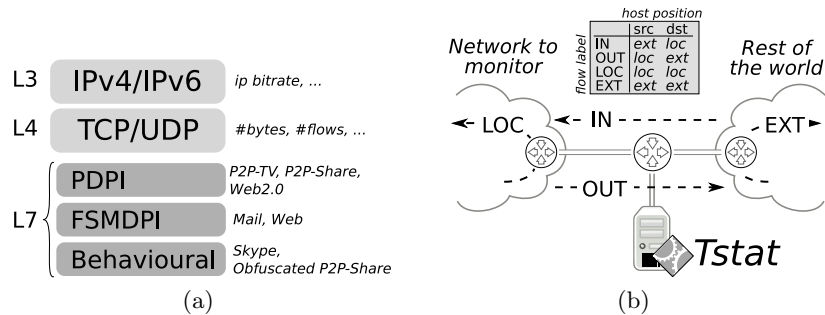
Fig. 1. Tstat modules (a) and monitoring probe setup (b)

flows, i.e., a single communication considering all IP packets having the same tuple $flow_{ID} = (ipProtoType, ipSrcAddr, srcPort, ipDstAddr, dstPort)$. Since services in the Internet typically require bidirectional conversations, *connections* are defined by grouping together the two flows going in opposite directions, i.e., in which src and dst addresses are exchanged. Connections are identified and tracked for both TCP and UDP traffic. Tstat discriminates client-to-server and server-to-client flows, where the client is conventionally identified as the sender of the first packet seen of the connection. Flow end is detected due to either proper connection tear-down (possible only in the TCP case), or a flow inactivity timeout (i.e., in case no new packet is seen for a connection for more than 120 seconds).

A common data structure is used to store some basic connections information, like the connection id, the total number of packets and bytes seen up to a given time, the timestamp of the first and last packet. Given a connection, two independent data structures collect statistics for each direction while advanced information related to a specific transport-layer (e.g., TCP, UDP, RTP), or to the application-layer (e.g., BitTorrent, Skype, ...), are grouped in specific structures linked to the flow data structures.

Since flow and connection structures must be accessed and updated for each packet, pointers to the structures are stored in two hash-tables, one for TCP and one for UDP connections. A simple hash function $hash(flow_{ID}) = (ipSrcAddr + srcPort + ipdstAddr + dstPort)\ mod\ N$ is used, being N the number of entries of the hash-table. Collisions are handled using a linked list.

Fig. 1(a) reports a simplified version of the analysis steps performed on each packet. After layer-2 (L2) framing de-encapsulation which is not subject of statistics, Tstat processes the L3 (network-layer) header: since Tstat is explicitly designed to process only IPv4 and IPv6 packets, possible non-IP packets are immediately discarded. Given the datagram service offered by IP networks, at the L3 only some per-datagram statistics are collected, like bitrate, packet length, etc.

The control is then handed over to the L4 (transport-layer) analysis. For both TCP and UDP, Tstat maintains a set of common statistics generally applicable (like the volume expressed in terms of bytes, packets or flows) and performs

further analysis like the identification of duplicated packets. For a TCP segment, further statistics are computed, like congestion window size, number of retransmissions [9], interrupted flows [10], etc.

At the L7 (application-layer), the main goal is to classify the traffic. A set of classification engines are triggered to identify which application has generated the flow under analysis. More in details, Tstat implements three different engines, based on different technologies:

Pure Deep Packet Inspection (PDPI): each application is uniquely identified by an expression of L7 payload bytes, called *signatures*. All the applications signatures are collected in a dictionary, defining a set of classification rules. Using a *pattern matching* approach, the signatures are then checked against the current payload until either a match is found, or all the signatures have been tested. In the first case, the flow is labeled according to the application associated to the rule that has matched, while in the second case the flow is labeled as "unknown". This approach is used to identify *Web 2.0* (Megaupload, RapidShare, YouTube, Vimeo, Facebook, Flickr), *P2P-TV* (PPLive, TvAnts, Sopcast), and *P2P-file-sharing* (Bittorrent, eMule, Kad, DC++, Gnutella) partially derived from *IPP2P*[11].

Finite State Machine Deep Packet Inspection (FSMDPI): the mechanism is similar to PDPI but involves the inspection of more than one packet of a flow. Finite State Machines (FSM) are used to verify the application protocol so that, to have a positive match, a specific sequence of matching rules have to be triggered. For example, if the first packet contains '`GET http://`' and response carries '`HTTP/1.0 OK`', the flow can be considered as HTTP. Using this approach, Tstat identifies *Mail* (IMAP, POP3, SMTP), *Chat* (MSN, XMPP/Jabber, Yahoo), *Web* (HTTP), *Encripted traffic* (SSL/TLS, SSH) and *VoIP* (RTP/RTCP).

Behavioral: given that the previous classifiers are based on payload inspection, in many cases they fail to correctly classify applications which rely on encryption mechanisms to protect the payload. In this case, behavioral classifiers can be used. They exploit *statical properties* like the packet size or inter arrival time to distinguish among applications. Using this approach, Tstat identifies *Skype* [12] and *Obfuscated P2P-file-sharing* (BitTorrent, eMule) traffic as well.

All these techniques are designed to work even if the complete packet payload is not available due to privacy or disk space concerns.

Once the traffic has been classified, further analysis can be carried out by application-specific plugins. For instance, once an RTP flow has been recognized, specific VoIP metrics are computed, like packet loss probability, loss burst length, jitter, etc.

Computational complexity of the above operations has been extensively analyzed in [13,14], showing that with off-the-shelf hardware it is possible to process several Gbps with no particular problems.

2.2 Input

Tstat offers the capability to both (i) monitor in real time operational networks and (ii) offline analyze packet level traces. In the first case, the software runs on a *probe*, i.e., a dedicated PC that "sniffs" traffic flowing on an operative link, as shown in Fig. 1(b). In this context, Tstat processes data packets that are captured in real time from a network interface. The standard *libpcap* library is supported to capture packets from standard Ethernet cards, but dedicated hi-end capture devices such as Endace DAG [15] or AITIA S1GED [16] cards are also supported. When used to process already captured packet level traces, Tstat reads from data files that store the packet level trace. This allows to inspect specific traffic for post-mortem analysis, to develop more complex statistical analysis for advanced performance evaluation (e.g., testing several algorithms, or checking the impact of some parameter over the same trace multiple times). A variety of dump file formats are supported, like pcap, erf, etherpeek, snoop, ns2 and netmetrix, and files can also be provided in compressed form, derived from 7zip, gzip and bzip.

Finally, it is also possible to compile Tstat as a *library*, called *libTstat*, so that its analysis capabilities can be easily linked to other traffic analysis tools. In this case, Tstat runs as a "plugin" of the other application and a simple API is used to pass packets to Tstat. In this case Tstat no longer has control on the measurement interface and the main application is free to tune the amount of payload handed over to Tstat or to filter out some packets, or to anonymize IP addresses and ports for privacy purposes. In our experience, this approach has revealed very successful, facilitating the integration of Tstat with the monitoring tools that operators may already have developed. For example, Tstat has been successfully integrated into TIE, a analysis tool developed by the university Federico II di Napoli, and METAWIN.

When characterizing a network, such as a campus LAN or an ISP Point-of-Presence (PoP), as reported in Fig. 1(b), it is often very useful to distinguish the traffic transmitted by "local" hosts from traffic sent by "external" hosts. By simply providing a set of IP network addresses used by local hosts, Tstat label flows according to theirs source. Four cases are possible:

- *incoming* traffic (in), if the source is external and the destination is local;
- *outgoing* traffic (out), if the source is local and the destination is external;
- *local* traffic (loc), if both source and destination are local;
- *external* traffic (ext), if both source and destination are external;

If the set of local network addresses is not provided, Tstat labels all the flows as local. Since the external flows contain traffic which does not involve the local network, statistics are collected only for the first three directions. Tstat allows to configure different sets of statistics for different classes, e.g. collect the incoming bitrate and the outgoing number of flows.

2.3   Output

Output statistics are reported in different formats, providing different levels of information:

Connection Logs: A set of text files which provide detailed information for each monitored connection. Each log is arranged as a simple table where each column is associated to a specific information while each line contains statistics related to the two flows of a connection. Tstat generates several different log files: UDP log, which collects all statistics related to UDP connections; multimedia log, which collects statistics for RTP/RTCP traffic; Skype and chat log, which collects statistics about Skype and Chat services; finally, TCP log, which collects statistics related to TCP traffic, and is split into two separate logs, discriminating "complete" connections (i.e., TCP connections properly started by a three-way handshake) from "incomplete" connections (i.e., TCP connections with partial three-way-handshake, due to routing asymmetry, packet loss, or malicious traffic, like port-scan). When a connection is closed, a line in the corresponding log is added.

Histograms: A set of histograms are used to collect empirical frequency distributions of parameters such as IP packet length, number of times a L4 port has been used in a flow, probability of an interrupted TCP flow, per-protocols bitrates, or number of flows active in a time slot. For each measured metric, the corresponding histogram is stored on separate file, using a two column table reporting the indication of the bins and the number of samples observed in that bin. Histograms can be generated periodically, i.e., every 5 minutes by default the current histogram is flushed to disk and a new collection starts, or a unique histogram for each feature can be saved at the end of the analysis.

Round Robin Database (RRD): A Round Robin Database (RRD)[17] is created for continuous monitoring a given statistics. RRD has been designed as a scalable mechanism to store historical data, by aggregating them with different granularity: newer samples are stored with higher frequencies, while oldest data are averaged in coarser time scales. This dramatically reduces the requirements in terms of disk space and, thanks to the tools provided by the RRD technology, it is possible to easily inspect the results. Indeed, it is possible to associate each Tstat probe to a website template using a simple *CGI* interface to query the RRD database [18], and obtain plots of historical measurements.

Packet traces: At the end of the packet analysis, it is possible to dump it into pcap traces. The traffic can be saved in an aggregated IP trace but it is also possible to separate TCP from UDP. Moreover, by exploiting its classification capabilities, Tstat can also split the UDP traffic according to the application layer. For example, RTP connections can be saved on a different file with respect to eMule without the need for processing the original traffic more than once.

The chosen output configuration anyway is very flexible thanks to the *runtime* engine. With this module in fact is possible to change the output configuration

(a) IP bitrate.

(b) IP flows.
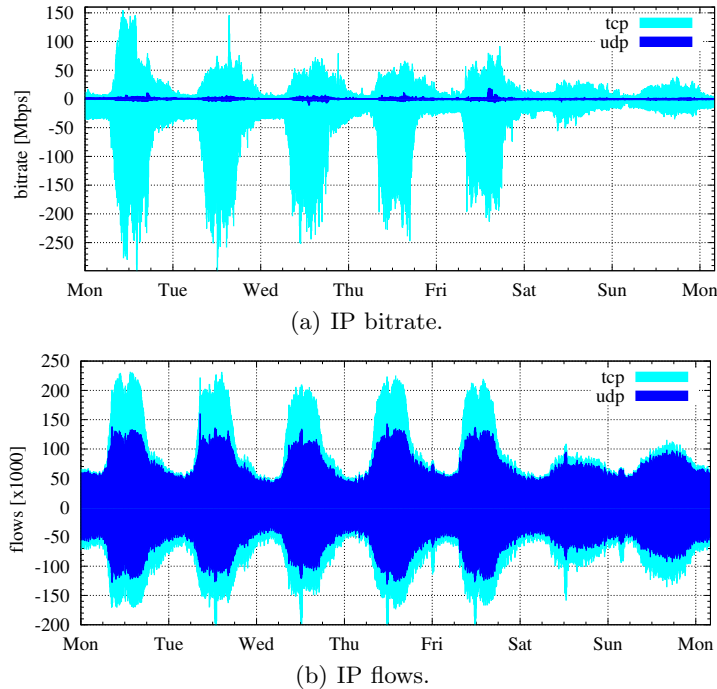
Fig. 2. One week of outgoing(+) and incoming(-) traffic volume

at runtime i.e. without the need to restart the program. In this way, is possible to configure a Tstat probe with a base configuration while specific statistics can be enable/disabled when needed.

3 Gallery of Tstat Capabilities

3.1 Campus Network

Politecnico di Torino institution is the second largest technical University in Italy, with about 1800 staff members and 28,000 students. The campus network is connected to the GARR [19] network, the Italian and European research network, with a 1 Gbps link. It is subdivided in more than 14 subnetworks, with approximatively 9,000 PCs and 50 servers accessing the network during a typical working day.

The Politecnico network has been constantly monitored using a Tstat probe since 2000. The probe runs on common PC hardware, i.e., dual core Intel Xeon 2.40GHz, 1 Gbyte of RAM and a simple Intel based Ethernet card. Linux is used as OS. The probe is configured to use the *runtime* engine. RRD are continuously collected and can also publicly browsed through the Web interface provided in the Tstat website [18].
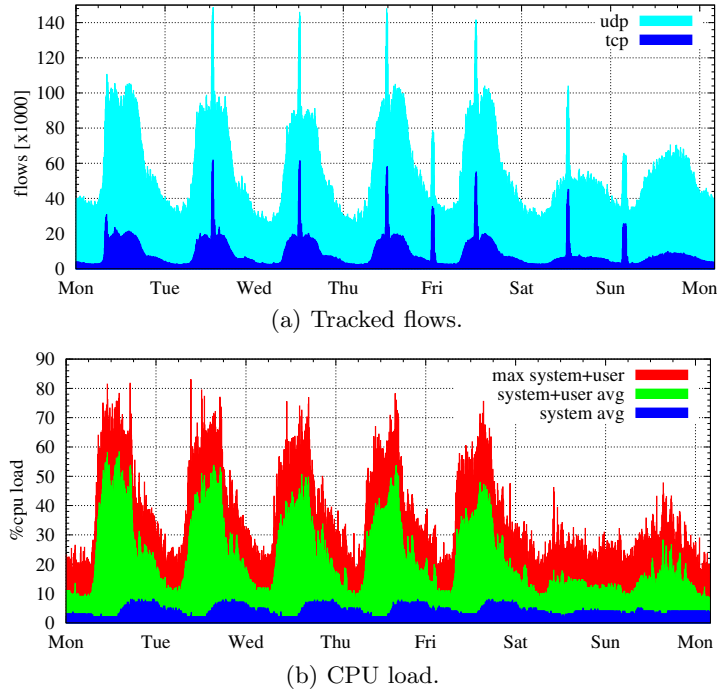
(a) Tracked flows.

(b) CPU load.

Fig. 3. One week of load sustained by the probe

3.2   RRD Examples

In this section, we report some examples of RRD graphs that can be obtained using a Tstat probe. Results reported cover a period of time between January 2009 to March 2010. One week of traffic, the first of February 2010, has also been chosen and analyzed as representative of the traffic of the campus network.

The plots show the variation over time of given measures in the considered periods, and each point corresponds to a 5 minute time-window. Positive and negative y-axis values (when they both are present) are used to distinguish outgoing and incoming traffic directions, respectively. Curves are stacked, so that the measures are breakdown in classes each associated to a different colour.

Traffic volumes. Fig. 2 reports the bitrate and the number of new connections over a one week time period. As expected, we observe that the traffic is highly asymmetric in terms of bitrate, e.g., in normal condition, the incoming traffic volume is 4-5 times larger than the outgoing traffic. The difference in terms of connections is only about 25% of the total. This is due to the intrinsic characteristics of our campus network, where most of the users require information to the outside world, i.e., they open outgoing connections that produce incoming volume. It can be seen that traffic volume evolution follows a well known day/night pattern, with extremely low values during nights and week-end days: a sustained

(a) TCP bitrate per application.


(b) HTTP bitrate per application.
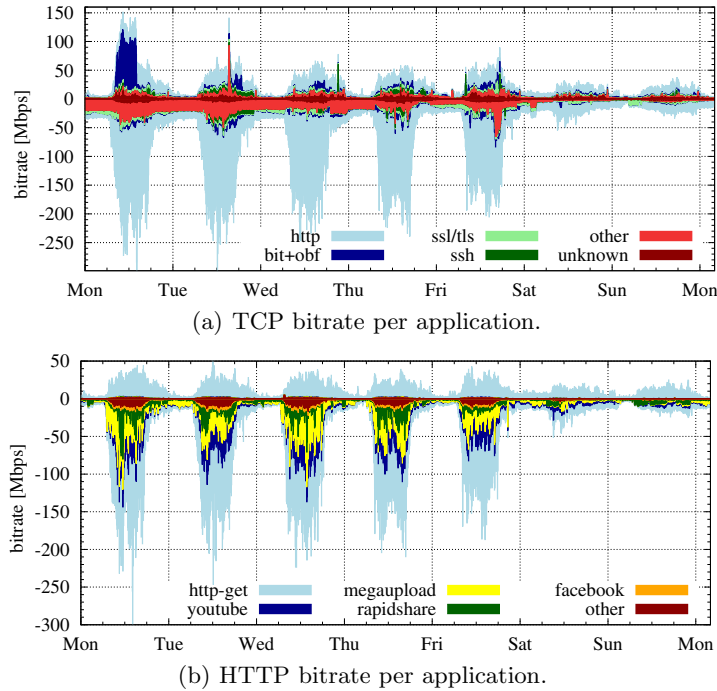
Fig. 4. One week TCP traffic outgoing(+) and incoming(-) classified

activity is present between 8am and 6pm (i.e., during office hours), with a drop around 12am corresponding to lunch time. Similarly, during the week end traffic is much smaller given fewer users are in the University premises.

By comparing Fig. 2(a) with Fig. 2(b), we can see that TCP accounts for more than 90% of the traffic volume, but UDP accounts for a very significant amount of connections. Indeed, UDP represents nearly half of the connections during the day, and practically all of the connections during the night. This higher number of UDP connections is due to the connectionless service of UDP, that is used by applications such as DNS which exchange very little amount of traffic. Later on, we will focus more deeply on traffic classification.

As a last example, Fig. 3(a) reports the number of TCP and UDP connections Tstat tracked. Since connections are bidirectional, we do not distinguish incoming and outgoing flows. As expected, the number of connections follows the daily pattern already noticed in Fig. 2(a) and Fig. 2(b). Interestingly, some very regularly spaced peaks in the number of TCP connections are present, possibly corresponding to some experiments, network or port-scan activities.

Finally, Fig. 3(b) reports the average CPU load of the probe machine, separately showing the system space CPU time and the user space CPU time, i.e., the time spent in kernel mode by OS calls, and the time spent in running Tstat. Measurements refer to the average CPU utilization in a time window of 5 minutes. To identify critical situations, the total maximum CPU load sustained in
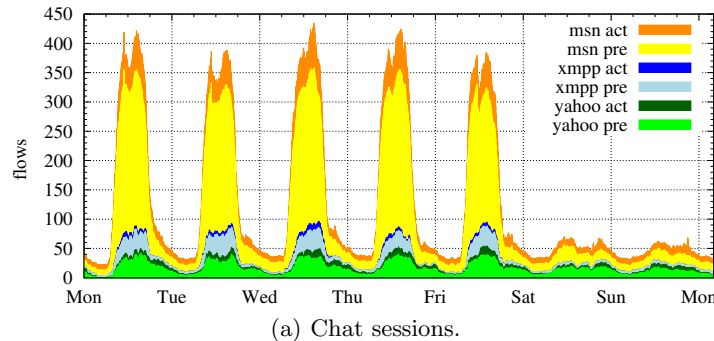
(a) Chat sessions.

Fig. 5. One week chat session classified

the 5 min time window is shown. The maximum load always smaller than 100% confirms that the Politecnico di Torino Tstat probe was able to process an aggregate amount of traffic of up to 400 Mbps, composed by more than 2,000 new connections per second using common hardware. Based on our experience [13], the major bottleneck of the probe is due to the Ethernet NIC communications, which, not being optimized for packet capturing, can overload the PC system bus by generating too many IRQ per seconds. The use of dedicated traffic capturing devices like [15,16] solves this problem.

Traffic classification. As briefly described in the previous section, Tstat offers advanced traffic classification capabilities. Fig. 4, Fig. 5 and Fig. 6 report some examples of such capability. Fig. 4(a) starts showing the TCP traffic volume breakdown per application. Among all the protocols identified by Tstat, we select the topmost 4 applications, namely: HTTP, SSL/TSL, SSH and BitTorrent (both plain and obfuscated). Other correctly classified protocols are aggregated in the "other" class, while unclassified traffic is labeled as "unknown". As always, outgoing/incoming traffic is reported using positive/negative values. Results shows, that the large majority of incoming traffic volume is due to HTTP traffic, with a small fraction of SSH traffic, and little Bittorrent traffic. Indeed, the campus network traffic is regulated by a firewall which limits the possible protocols; for example, it blocks eMule traffic which is very common in other scenarios. Considering outgoing traffic, HTTP traffic represents only half of the total volume, which is due to Web service asymmetry. On the 1st day, it is however possible to see an "abnormal" activity, represented by some local Bit-Torrent client that was transmitting a lot of data, possibly seeding some content. Some other strange activities are also present. For example, there is a moderate constant and large amount of unclassified traffic which is due to the PlanetLab nodes present in the campus network that were likely conducting experiments using some unknown protocol.

Given the predominance of HTTP traffic, it is interesting to observe which application is actually used to generate traffic. Fig. 4(b) reports the breakdown
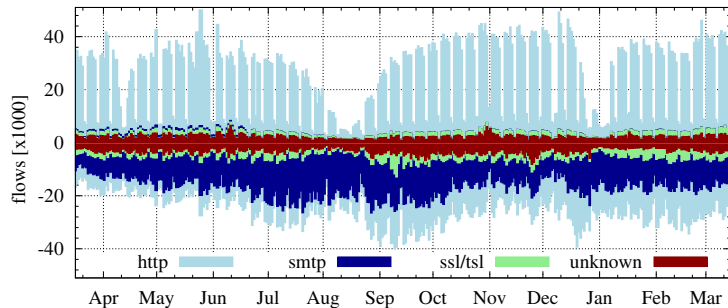
Fig. 6. One year of TCP flows classified

of the HTTP volume of traffic per application. It shows that popular streaming services, like YouTube, or social networks, like Facebook, generate about 30% of HTTP traffic. More interestingly, about 30% of the incoming volume is due file hosting services (like Megaupload or RapidShare), which are becoming alternatives for downloading contents when P2P file sharing applications are not available or strictly regulated as in our campus network.

We now focus on chat services, to show another capability of Tstat. Fig. 5 shows users chatting habits during the week. It considers Windows Live Messenger (MSN), XMPP based chat services (like Google Talk) and the proprietary Yahoo Messenger. Chat traffic is further classified as *presence* and *activity*, respectively referring to traffic generated by the client to simply use the service, and actual chat sessions generated by the user. Besides showing the different popularity of the three chat services, Fig. 5 shows that the number of users logged, but inactive, is much larger than the number of users that actually have ongoing chat sessions. This is especially evident for MSN, which is by far the most popular chat service in the campus network.

Finally, Fig. 6 reports the number of TCP connections seen every 5 minutes, averaged over a one day log interval. The plot covers a period of time starting from April '09 to March '10. As already observed for Fig. 2(b), the number of outgoing flows is higher than the incoming one (reported using negative values on the y-axis, as usual). The year-long plot allows to appreciate the weekly periodicity of traffic, with weekend days showing much less activity. The summer and Christmas periods show also a clear drop in traffic, being those a typical vacation period. Finally, as already noticed, the large majority of traffic is due to HTTP based services in our network. In more depth, more than the 50% of the incoming connections are due to SMTP traffic, i.e., emails received by the campus mail server. For the large part, this traffic corresponds to spam messages that the network has to carry, which is later on tagged as SPAM by the email server. The amount of email connections received on average is indeed equal to more than 20,000 every 5 minutes, and it is not correlated to user activity, e.g., during the summer the volume of SMTP connections is constantly high despite a few users only are presents.

4   Conclusions

This paper reported our experience with the use of Tstat in live traffic monitoring of operational networks: Tstat capabilities have been quickly described and, as an example, practically used in our campus network.

Started as a scalable flow-level logger, Tstat capabilities evolved toward upper layer analysis and classification capabilities. The flexible input/output interface makes it worth using for researchers (with a great deal of information available for post-processing) but also simple to deploy and use as-is (with a nice browsable Web interface). Among other interesting features of Tstat, we point out its high configurability, possibly at runtime. Moreover, Tstat is also extensible by writing new specified plugins. Finally, Tstat can also be embedded in current monitoring applications and infrastructures, for which it can run as a plugin.

References

1. Kapoor, R., Chen, L.-J., Lao, L., Gerla, M., Sanadidi, M.Y.: CapProbe: A Simple and Accurate Capacity Estimation Technique. In: ACM SIGCOMM'04, Portland, USA (2004)
2. Downey, A.B.: Using pathchar to estimate Internet link characteristics. ACM SIG-COMM Computer Communication Review (1999)
3. Rizzo, L.: Dummynet: a simple approach to the evaluation of network protocols. ACM Computer Communication Review (January 1997)
4. Wireshark Homepage, `http://www.wireshark.org/`
5. Roesch, M.: Snort - Lightweight Intrusion Detection for Networks. In: 13th USENIX LISA Conference (1999)
6. Moore, D., Keys, K., Koga, R., Lagache, E., Claffy, K.: The CoralReef Software Suite as a Tool for System and Network Administrators. In: 15th USENIX Conference on System Administration, San Diego, CA (December 2001)
7. TSTAT Homepage, `http://tstat.tlc.polito.it`
8. TCPTrace Homepage, `http://www.tcptrace.org`
9. Mellia, M., Meo, M., Muscariello, L., Rossi, D.: Passive analysis of TCP anomalies. Elsevier Computer Networks 52(14) (October 2008)
10. Rossi, D., Casetti, C., Mellia, M.: User Patience and the Web: a Hands-on Investigation. In: IEEE Globecom'03, San Francisco, CA, USA (December 2003)
11. IPP2P Homepage, `http://www.ipp2p.org`
12. Bonfiglio, D., Mellia, M., Meo, M., Rossi, D., Tofanelli, P.: Revealing Skype Traffic: When Randomness Plays with You. ACM SIGCOMM Computer Communication Review 37(4), 37–48 (2007)
13. Rossi, D., Mellia, M.: Real-Time TCP/IP Analysis with Common Hardware. In: IEEE International Conference of Communication (ICC'06), Istanbul, Turkey (June 2006)
14. Rossi, D., Valenti, S., Veglia, P., Bonfiglio, D., Mellia, M., Meo, M.: Pictures from the Skype. ACM Performance Evaluation Review (PER) 36(2), 83–86 (2008)
15. Endace Homepage, `http://www.endace.com`
16. AITIA Homepage, `http://www.aitia.ai`
17. RRDtool Homepage, `http://oss.oetiker.ch/rrdtool/`
18. TSTAT RRD Web interface, `http://tstat.tlc.polito.it/web.shtml`
19. GARR Homepage, `http://www.garr.it/reteGARR/index.php`