Topnet: A Network-Aware top(1)

Antonis Theocharides, Demetres Antoniades, Michalis Polychronakis, Elias Athanasopoulos, and Evangelos P. Markatos – Foundation for Research and Technology; Hellas, Greece[1]

ABSTRACT

System administrators regularly use the top utility for understanding the resource consumption of the processes running on UNIX computers. Top provides an accurate and real-time display of the computing and memory capacity of the system among the running processes, but it provides no information about the network traffic sent and received by the processes running on the system.

Although we've seen a proliferation of network monitoring tools that help system administrators understand the traffic flowing through their networks, most of these tools have been designed for network deployment and can not easily, if at all, provide real-time attribution of network resources to individual processes running on end hosts.

In this paper, we describe the design and implementation of *Topnet*, an extension of the top UNIX utility that provides a *process-centric* approach to traffic monitoring. *Topnet* presents users with an intuitive real-time attribution of network resources to individual processes. Our evaluation suggests that *Topnet* through (i) the familiar user interface of top and (ii) a reasonable performance overhead, provides an accurate way to attribute network traffic to individual processes, enabling users to have a more comprehensive process-aware understanding of network resource consumption in their systems.

Introduction

The UNIX top utility [34] is used daily by users and system administrators for real time monitoring of system and process information. Top provides a continuously updated breakdown of system resources such as CPU and memory of the running processes, as well as process-specific information such as the state, priority, size, total CPU time, and so on. However, top does not provide any information about how the network resources of the system are utilized by the different processes. Although there exist several system utilities that display information about the network configuration, connection state, or statistics per network interface, currently there is no convenient way to attribute the system's network traffic to the individual processes that send or receive it.

With the proliferation of network services and systems, it is increasingly difficult for users to keep track of which applications communicate through the network, when this is happening, and how much traffic each application sends and receives. For example, besides the operating system, popular applications such as browsers, media players, and productivity suites, among many others, communicate with remote servers for automatic updates. In many cases the user is unaware of this activity. Similarly, the network activity of peer-to-peer software like file sharing or video conferencing applications is not always directly related to the user's actions and thus the exact network behavior is unclear to the end user.

Consider, for example, a Voice-over-IP (VoIP) application, such as Skype. Even when the user is not directly using the application, Skype may still consume processor and network resources to relay other users' traffic and keep the overlay network well connected. Thus, in the presence of such applications generating network traffic in the background, administrators would like to be able (i) to understand when such traffic is generated, and (ii) to pinpoint the processes that are responsible for the generation of this traffic.

In this paper, we present the design and implementation of *Topnet*, an extension of the top UNIX utility that introduces a *process-centric* approach to network traffic monitoring. *Topnet* provides users with a simple and intuitive real-time attribution of the system's incoming and outgoing network traffic to the running processes in additional columns of the familiar console output of top. By treating incoming and outgoing traffic as additional process properties, the user can instantly spot the running processes with current network activity, and by sorting according to the relevant column, identify which applications send or receive most of the traffic.

Our work suggests that *Topnet* through (i) a familiar top user interface and (ii) a reasonable performance overhead, provides an accurate way to attribute network traffic to individual processes, enabling users to have a better understanding of the consumption of resources in their systems.

The contributions of this paper are:

- We provide a *process-centric approach* to processor, memory, and network resource monitoring, that is implemented completely in user-level without requiring any kernel modifications as previous approaches do.

[1] The authors are also with the University of Crete.

- We *demonstrate the feasibility* of our approach by providing an open source implementation of *Topnet*.
- We *evaluate Topnet* and show that it provides high measurement accuracy at a reasonable computational cost for most of the expected operation range.

Related Work

In this section we present related efforts. We start with exploring modern home networks and the arising need for better administration, even by the end user and then we proceed in more complex tasks like monitoring and classifying aggregated network traffic. We finally, present visualization techniques for similar applications to *Topnet*.

Home Networks

The evolution of home networks and the complexity they inhabit lately has driven the community to seek algorithms and technologies for a better administration of a host by the end user. Broadband technologies and their characteristics have been studied by Dischinger, et al. [19]. According to this study, broadband technologies that appear often in home networks experience high jitter rates, affecting the expected performance.

Papagiannaki, et al. [42], has shown how small configuration changes can affect the network performance of hosts connected in a wireless media of a home environment. In this context, it is vital for a user to have a tool that will assist her in inspecting the network health of a running host as per process network utilization is concerned. Indeed, this need is profound; lately there have been proposed standards [4] for per application network statistics to be included in modern operating systems. This effort will further assist the development of applications that collect and present per process network accounting information.

Network Monitoring Systems

Over the past few years, there has been an increasing interest in passive network monitoring systems and tools. Indeed, several infrastructures are available for deployment by system and network administrators in order to monitor the traffic usage of their networks by parsing either raw network packets, SNMP data, or network flow data.

For example, MRTG [41] is a popular tool for monitoring SNMP network devices and visualizing network usage. FlowScan [43] and commercial systems including Cisco Network Analysis Module Software [12] and IBM Aurora [25] present traffic usage patterns using Netflow exported data, and classify traffic using mainly the IANA port number list.[2] AutoFocus [20] aims at identifying important network consumption by clustering flows of similar or same interest, i.e., a large number of small network flows by a single web server that will not be positioned at the top flows if considered one by one.

Although these systems are widely deployed, they aim to provide a ''bird's eye view'' of the network traffic, informing system administrators of the overall status of their network and alerting them of major traffic events. On the contrary, *Topnet* focuses on the traffic usage not of entire networks, but of *individual applications* running on specified end host computers, and thus present a functionality complementary to these systems.

We consider, though, atop [1] to be the tool most like *Topnet* in terms of operation. atop provides a complete set of process resource statistics (RAM, CPU, storage, network, etc.). However, *Topnet* differentiates from atop in two basic ways. First, atop needs kernel modifications and thus has deployment complications, whereas *Topnet* uses widely used libraries for packet capturing in user-space, and second, *Topnet* is built over a well known and trusted framework that is well established in the community: the classic UNIX top tool.

Network Packet Capturing Systems

To provide a better understanding of network traffic, some systems enable administrators to capture (and subsequently process) all traffic which passes through a router (or computer). The popular libpcap [37] packet monitoring library provides a portable Application Programming Interface (API) for user-level packet capture. The libpcap interface supports a filtering mechanism based on the BSD Packet Filter [36], which allows for selective packet capture based on packet header fields. The Linux Socket Filter [27] offers similar functionality with BPF, while xPF [28] and FFPF [10] provide a richer programming environment for network monitoring at the packet filter level. Iftop [47] is a libpcap-based application that reports the bandwidth usage of a network interface by displaying a breakdown of the network traffic according to the active network flows.

To improve the functionality of monitoring sensors, beyond the naive capturing of network packets, FLAME [7] allows users to directly install custom modules on the monitoring system, similarly in principle to Management-by-Delegation models [23]. Windmill [35] is an extensible network probe environment which allows loading of ''experiments'' on the probe for analyzing protocol performance.

To provide intelligent packet capturing and sophisticated packet processing in a single system, in our earlier work, we designed and implemented MAPI [44, 46] an Application Programming Interface for network monitoring. MAPI captures packets passing through a monitoring link and provides mechanisms to build custom applications for network monitoring. Hughes in [24] presented qcap: a library for capturing and decoding live traffic streams in network level. The

[2] http://www.iana.org/assignments/port-numbers

library provides the functionality for developing network monitoring applications able to decompose the flows up to the layer 7 protocol.

Although packet-capturing systems may provide a global view of network traffic and of the IP addresses which generate/consume the observed traffic, they are usually fined-tuned for deployment at network nodes and not at end hosts. Thus, these systems provide little, if any, information on the end user *applications* which generate the traffic in question. In contrast, our approach, *Topnet*, targeted for deployment at end hosts, aims to help end users as well as system administrators understand the individual applications which are responsible for the network traffic generated from (or destined to) a particular computer.

Traffic Classification Tools

To improve the understanding of the types of applications which generate a particular amount of traffic in the network, several packet- or flow-capturing systems are equipped with traffic-classification tools. These tools are able to attribute network traffic to classes of applications that generated this traffic, such as peer-to-peer systems, web browsing, and IP telephony.

The first generation of traffic classification tools attributed traffic to applications based on the IANA port number list, associating, for example, all traffic destined to (or originated from) port 80 with web browsing, all traffic destined to port 25 with email, and so on [43]. However, in the wake of elusive peer-to-peer applications that use dynamic ports, the accuracy of this port-based approach was quickly shown to be inaccurate [39, 30]. As a result, recent approaches use deep packet inspection and application signatures for attributing traffic flows to the corresponding applications [29, 45].

NetADHICT [26] provides a hierarchical decomposition of traffic, based on similar patterns both in header and payload level, yet the labeling of the nodes derived from the application is left for the administrator. Another application that can classify network traffic by packet inspection is ntop [17, 18]: an extended network monitoring tool for displaying the top users of a network. It uses a plugin architecture for deploying decoders that will assign the network flows to the applications that generated them. Finally, Appmon [8] is an open-source passive network monitoring application that applies deep packet inspection and searches the network packets for specific application signatures.

Although highly accurate, methods based on deep packet inspection suffer from two major drawbacks: (i) they have high computational costs, and (ii) they are ineffective in the presence of encryption. To overcome these deficiencies, recent approaches try to identify the applications that generate the traffic by not looking at the packet payload, but only at the transport layer [31, 13] or at the statistical characteristics of the network flow, like packet sizes and round-trip times [9, 50, 15, 14]. A different approach presented by Karagianis, et al. tries to classify traffic by characterizing the behavior of the host generating this traffic [32].

Although the previously described traffic-classification tools have been widely used, they have been developed for deployment at network nodes and may not appropriate for host-level use. Although the motivated user could run such a tool, e.g., Appmon [8], for monitoring a single host, it will still suffer from several drawbacks:

- Appmon will not be able to categorize encrypted traffic.
- Appmon has no notion of user-level applications. That is, Appmon reports how much ftp traffic is generated by an IP address, but does not know which process generated the traffic.
- Appmon can not distinguish between similar applications. For example, if the ftp traffic is generated by two or more ftp clients, Appmon will not be able to report the ftp traffic generated by each one of the clients.

On the contrary, *Topnet* provides all the above functionality with the same ‘‘look and feel’’ of the familiar top UNIX application.

Visualization

Work has also been done in visualization of network data with the goal to provide the administrator with an easily understandable picture of the network. VisFlowConnect [49], NVisionIP [33] and PortVis [38] are all tools providing connection level visualization of the network hosts in order to identify suspicious network usage though they lack host level information that would be able to identify the exact process responsible for the suspicious traffic. *Topnet* can tie up loose ends by providing the user of these tools with the real source of the traffic and enable her to decide for the proper administration actions to be taken.

Personal firewall tools like ZoneAlarm [11] and Objective Development’s Little Snitch [40] provide the user with sufficient information about the network usage experienced by running applications. But these tools are limited to a single operating system, and explicitly target the user of the machine and are of little to zero use for the administrator.

Fink, et al. implemented HoNe, a host-network visualization tool for packet-process correlation [22, 21]. Their tool is based on a loadable kernel module and the netfilter packet filtering framework [5]. HoNe focuses on visualizing the number of network connections a machine has and the corresponding applications for that connections. It mainly targets security analysis, through visualizing the process of an intrusion attempt, malware download and operation, in a connection level. Although HoNe shares some aspects with *Topnet*, our approach has several advantages including:

- HoNe needs kernel modifications while *Topnet* is completely implemented in user level.
- *Topnet* provides the simple and familiar ''look and feel'' of the *top* UNIX utility.
- Although HoNe does not provide any performance analysis, based on its description we believe that *Topnet* is faster and scales to larger network speeds more easily.

To summarize, we believe that *Topnet* fills a clearly identified gap in host-level network traffic monitoring. *Topnet* capitalizes on popular network level traffic monitoring (libpcap) and a proven resource monitoring system (*top*). *Topnet* provides an easy-to-use tool which helps users and administrators understand the traffic generated by the applications running on their computers.

Application Description

Implementation

Our implementation goal was to provide a tool that will be as efficient as possible, provide the maximum functionality and would be easily installed on any system. Thus we decided to build our tool based on existing, widely used system tools. We used the top UNIX utility as an interface, since it is widely used for system monitoring and easily understandable. More precisely, we used the Linux version of top. The choice of Linux was made for convenience, since we had easy access to a Linux based testbed. As we point out in the Future Work Section, we plan to port *Topnet* to various flavors of UNIX.

Although top provides comprehensive monitoring of the computing and memory resources consumed by the running processes, it lacks information about the network usage of the processes. To provide this functionality, we needed (i) a way to read the incoming and outgoing packets and (ii) a way to correlate the network traffic flows with the process responsible for them.

To read the packets from the network and correlate them with the process responsible for sending/receiving them, we used the widely available libpcap and netstat tools. Both tools are available for both UNIX-like and Windows systems and are widely deployed and used. Using libpcap, we implemented a simple sniffer that reads packets from the network interface. The sniffer is implemented as part of top's source code. At initialization, it tries to find the default network interface of the machine and next starts reading all the packets coming to and going from that interface. For each network flow, our application keeps an entry in a hash table where the correlation and statistics take place.

When a new packet is received by libpcap, *Topnet* first looks if the flow it belongs to has already been attributed to a process. If so, the statistics of the flow are updated and the application moves on to the next packet. If the flow has not been assigned to a process yet, we make a call to the netstat function to correlate the newly observed flow with the corresponding process. netstat reads various network-related structures and outputs results about active sockets, network protocol traffic statistics, remote endpoints and routing information. We use part of the netstat functionality to retrieve the list of active network sockets on the system. Having that information, we can assign the newly observed network flow to the process responsible for it.

Fink, et al. [21] argue that netstat would not be a complete solution since it will fail to track short-lived flows due to the fact that it polls the kernel with a user specified rate. Due to polling, a short-lived flow may not be captured by netstat in case it is created and destroyed between two consecutive polling periods. In contrast to netstat, we address with this limitation by reading the relevant data from the kernel whenever *Topnet* encounters a new packet belonging to a non-correlated flow. This allows us to track the new flow as soon as its first packet arrives, before the flow is terminated, even if the flow is short-lived.

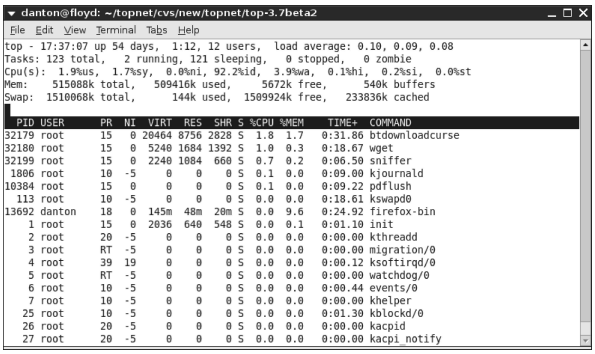
Periodically, every five minutes, the application performs a full pass of the hash table in order to clean invalid entries. We consider a flow entry to be invalid if it neither received nor transmitted any data for a period of 60 seconds. This way, the hash table contains only the active flows of the machine and minimizes the memory overhead of the application.

Features

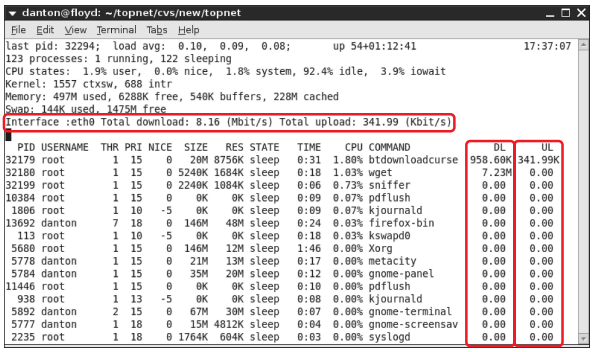
The *Topnet* default window (see Figure 1b) is almost identical to the window of the top utility (see Figure 1a), with the addition of two more columns next to the existing columns. The first column shows the incoming traffic rate for each process, while the second shows the outgoing traffic rate. The default values are in Kbps per second. We should note here, that *Topnet* reports traffic from a process perspective. That is, it counts only the payload bytes (layer 4) of each packets. Though, since it does not reassemble flows, bytes belonging to retransmissions are also going to be reported.

Also in the general statistics header, the upper lines in the *Topnet* interface, we added a line that reports the traffic speed of the default interface of the monitored machine. Figure 1b shows the default interface of *Topnet*. As we can see *Topnet* informs its user for the rate of the network consuming process. In this figure we have a wget process downloading a file, and a BitTorrent client with an active BitTorrent download.

Figure 2 presents the *Topnet* window in more detail. Here we show only the processes that use the network. In the top two rows of the per-process utilization part we see the three applications that make use of the network during the time the screenshot was taken. These three processes are operating three file transfers, an HTTP download using wget, an active



(a) top



(b) topnet

Figure 1: Screenshots of (a) the Linux top utility and (b) the *Topnet* tool.

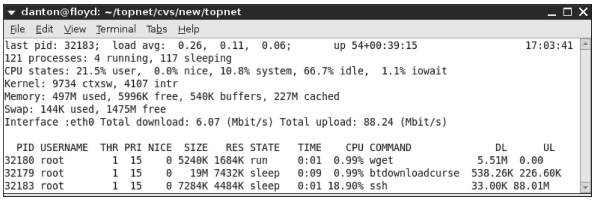


Figure 2: The *Topnet* window showing the network usage of three applications.

BitTorrent download, and an ssh file copy from another machine.

To ease the use of *Topnet* and to make as much information as possible available instantly, we added some hot keys to *Topnet*'s user interface. Table 1 presents the keys we added and the effect they have on the data presentation.

Hot keys can be used to highlight the applications with the most network activity. Pressing *r*, *t* and *R* sort processes according to the inbound, outbound or total traffic respectively. Using *a*, the user is able to select information about just the processes which are responsible for the network load of the machine.

Pressing *F* toggles the presentation of active network flows per process. The displayed information includes a list with the process' active connections and the network utilization for each connection. As an example, Figure 3 presents the full information available by *Topnet*'s interface. For each process *Topnet* displays the incoming, outgoing and total traffic rate per update interval in the process line, as in the default case of running *Topnet*.

For each process with active network connections, *Topnet* displays these connections as well as the traffic rate of each connection. In Figure 3 we see the analysis for three network consuming process running in a machine. Two of them (wget and ssh) have one flow each, while the third one, a BitTorrent process, has three active flows sharing its bandwidth.

Key	Functionality
r	Sort by download rate (receiving)
t	Sort by upload rate (transmitting)
R	Sort by total Rate
F	Toggle the display of network Flows per process
a	Show only processes with network load (active)
D	Dump processes traffic to tcpdump file format
y	Toggle the display of unknown network flows
$	Monitor a different interface

Table 1: Function keys and their functionality.

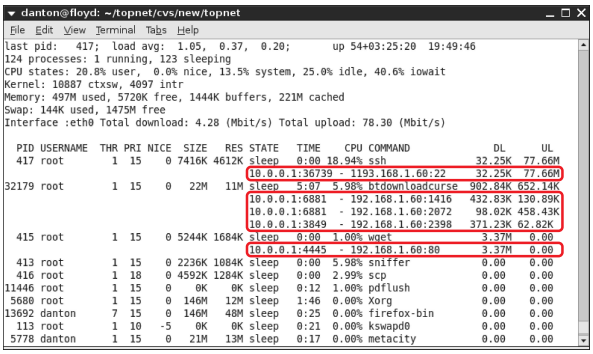


Figure 3: *Topnet* showing the list of active connections for three processes with network activity.

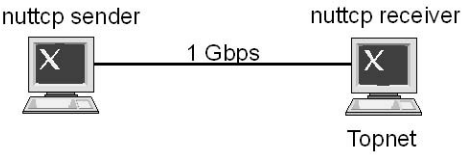


Figure 4: The testbed used for the evaluation of *Topnet*.

One of the most useful functions of *Topnet* is the ability to log the traffic of a specific application to a dump file for later use and analysis. This is done with the *D* key followed by the specific process id of interest. The raw packets are dumped in a file in the directory from which top was started with the PID and the process name in the filename.

Finally, using *y* one can have a view of the network flows not known to belong to a specific process. This case may be very useful when a user hides from *Topnet*, for instance, by using an application that uses raw sockets for communication. The *Topnet* utility, since it is based in libpcap, can capture packets sent using raw sockets. However, it is hard to map this traffic to a specific process. In order to cope with this issue, we group all *orphan traffic* – all traffic that we can not assign to a process – in a special *Topnet* window.

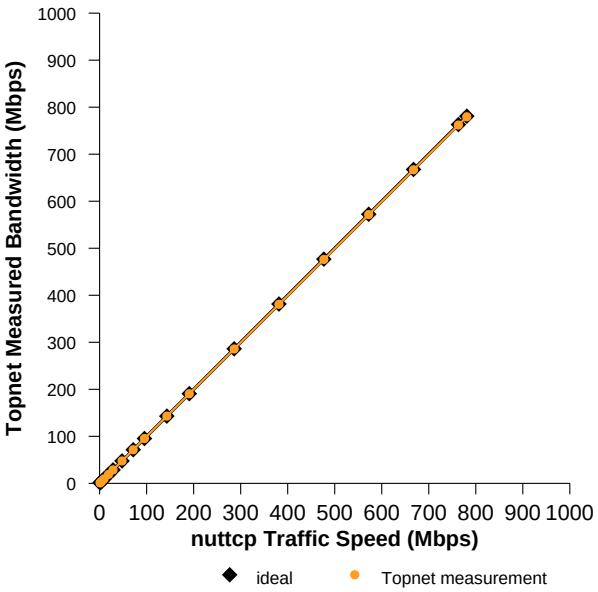


Figure 5: *Topnet* reported network consumption for the artificial traffic created with nuttcp.

Evaluation

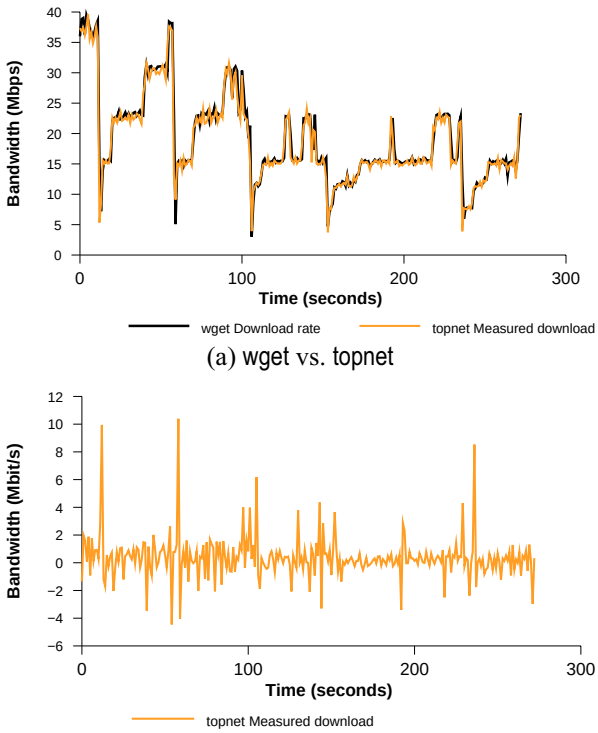
In this section we provide a twofold evaluation of our application. First we show the accuracy of the application using (i) artificial traffic in a controlled testbed environment, and (ii) real world traffic from two popular application protocols, HTTP and BitTorrent. Next, we present the resource overhead consumed by *Topnet*.

Experimental Environment

For all the experiments presented in this section we used a Dell 1420 computer equipped with a Dual Xeon 2.8 MHz CPU and 512 Mbyte of memory. The machine was connected to the network using a commodity Gigabit network interface.

Accuracy

The first and most important test for the application was to evaluate its accuracy in measuring the network load of a specific process. We first used artificial traffic in a testbed environment and as a next step we evaluated the accuracy using real world HTTP and BitTorrent traffic. In all our experiments we experienced zero packet loss with tcpdump, therefore those measurements represent ground truth for comparison.



(a) wget vs. topnet

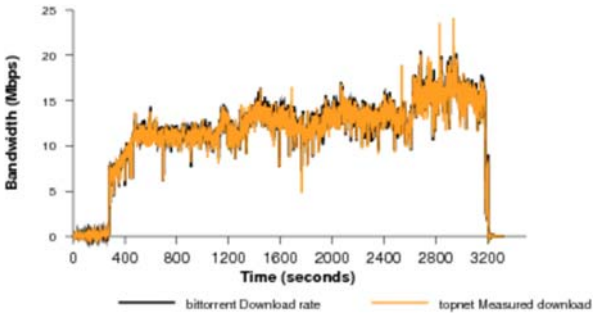
(b) Inaccuracy

Figure 6: *Topnet* results while monitoring a web download compared with the traffic reported by a tcpdump running in parallel. Figure (a) shows the actual traffic seen by both tcpdump and *Topnet* during the lifetime of the experiment. Figure (b) shows the difference between the two measurements. With an average accuracy of 98.1%, in most cases *Topnet* provides accurate results with the exception of some short reporting periods.

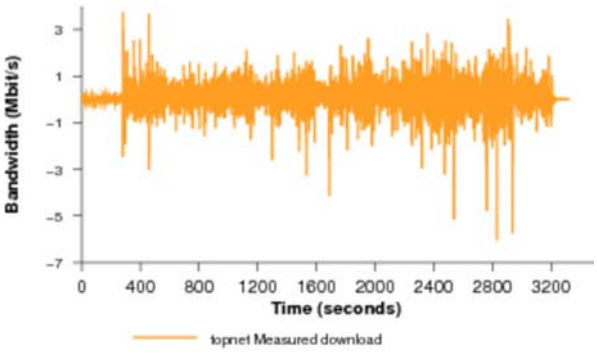
Measuring Synthetically-Generated Traffic

For the testbed experiments we used two machines on a Gigabit network, exchanging traffic with each other using nuttcp [3]. One machine was used as the sender and the other as receiver. We ran *Topnet* on the receiver machine and logged the measurement results per second for the nuttcp receiver process. We instrumented the nuttcp sender to send traffic to the receiver in different rates, ranging from 1 Mbps to about 870 Mbps. The complete testbed is depicted in Figure 4.

Figure 5 presents the measured traffic load reported by *Topnet* during the experiment. The *x* axis presents the traffic send through nuttcp in Mbps, while the *y* axis shows the traffic measured by *Topnet*. The results are averages for the total of 10 runs for each traffic speed and present the average traffic sent and measured for the duration of each run. Figure 5 also plots the $y = x$ *ideal* curve. We see that *Topnet* has an almost exact match with the ideal in all ranges of the experiments. Our data suggest that there is less than 0.1% difference between the two curves.



(a) torrent vs. *Topnet*



(b) Inaccuracy

Figure 7: *Topnet* results while monitoring a BitTorrent client compared with the traffic logged by tcpdump. Figure (a) shows the actual inbound traffic measured by the two tool, *Topnet* manages to follow the exact traffic rates of the BitTorrent download. Figure (b) shows the differences from the two measurements in more detail. Again, *Topnet* seems to show only slight differences from the tcpdump traffic, with the overall accuracy to be 98.8%.

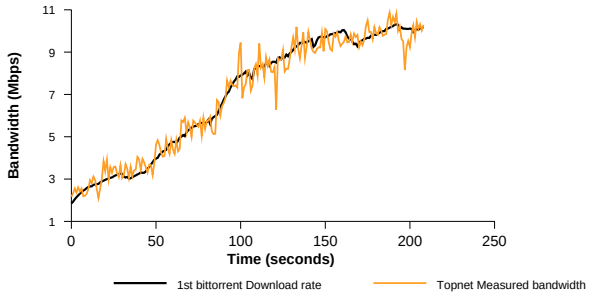
Measuring HTTP and BitTorrent Clients

Since *Topnet* shows high accuracy on a controlled environment with artificial traffic, the next step was to evaluate the measurement accuracy using real world traffic. We used two applications utilizing two popular protocols: HTTP and BitTorrent.

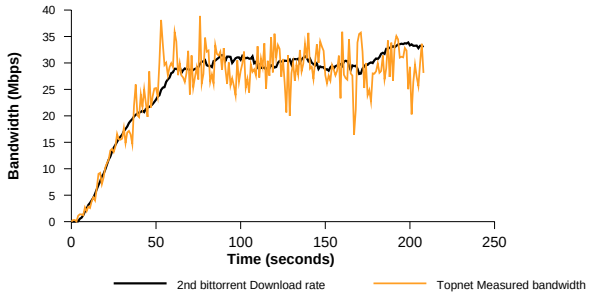
For the evaluation with HTTP traffic we used the well known wget application, a command-line web HTTP GET application. We used tcpdump to log the HTTP traffic exchanged by the machine during the time of the experiment and compared it with the traffic reported by *Topnet*. *Topnet* was instrumented to report the per-application traffic every second.

We used wget to download a Linux distribution ISO file.[3] Figure 6a presents the incoming traffic reported by *Topnet* during the HTTP download in comparison with the traffic reported by tcpdump. The relative time for the whole duration of the download is plotted on *x* axis, while the measured one-way traffic (inbound) in Mbps is plotted on the *y* axis. Each point

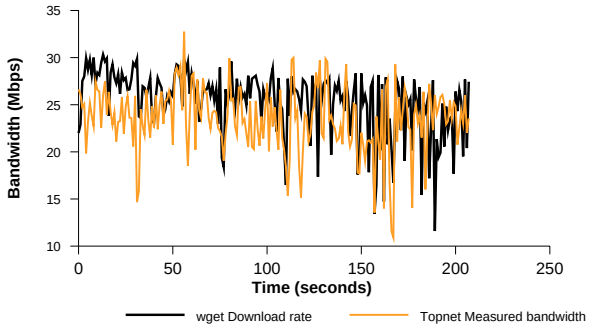
[3]http://ftp.belnet.be/mirror/ubuntu.com/releases/hardy/ubuntu-8.04.1-desktop-i386.iso .



(a) BitTorrent 1



(b) Bittorrent 2



(c) Wget

Figure 8: *Topnet* reported traffic rates when monitoring one wget and two BitTorrent downloads in parallel. Both measurements for BitTorrent show an average difference around 1% (1.05% for client 1 and 0.43% for client 2). The wget measurement shows a larger difference around 10%.

gives the traffic for one second. The figure shows that *Topnet* managed to accurately measure the traffic of the download during its whole duration and also follows the changes experienced during the download. Figure 6b plots the difference in the measurement reported by each tool, as the inaccuracy of our tool. As we see, *Topnet* experiences limited differences from the tcpdump logged traffic. This differences can be explained from slightly different times for reporting the traffic.

For the next experiment we used the BitTorrent client to download a Linux distribution through the BitTorrent protocol.[4] We once again used tcpdump to

[4]http://cdimage.debian.org/debian-cd/4.0_r3/i386/bt-dvd/debian-40r3-i386-DVD-1.iso.torrent

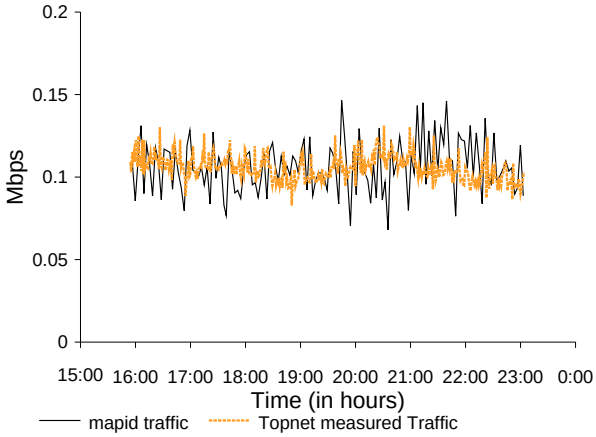


Figure 9: *Topnet* accuracy while monitoring a server process for several hours. The overall difference between *Topnet* and tcpdump is 1.73%.

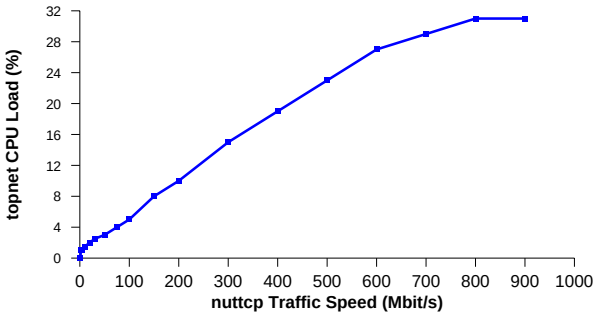


Figure 10: *Topnet* CPU usage using artificial traffic created with nuttcp. *Topnet* uses at most 31% of the CPU while experiencing a full network loaded machine with 900 Mbps of traffic.

report the downloaded traffic per second for the whole duration of the data transfer.[5] Figure 7a reports the true traffic exchanged by a BitTorrent client while downloading the file, as incoming traffic in bits per second, in comparison with the traffic reported by *Topnet* for the downloading process. We see that the results reported by *Topnet* and the BitTorrent client are very close. To better show the accuracy of *Topnet* we plot the difference between the two applications for each measurement second in Figure 7b. *Topnet* measurements exhibit little differences from tcpdump-based measurements, with the exception of a small number of outliers. The differences can be explained with the different timing that the two applications have, due to manual start-up.

For the next experiment, we tried to simulate the everyday network usage of a normal user. For instance, the user might browse someweb pages while there may be some active BitTorrent downloads in the background. To simulate this case we concurrently employed an HTTP download and two BitTorrent downloads for different files. The comparison is pictured in Figure 8.

[5]During this experiment, access was limited only to the BitTorrent client so as to avoid unexpected non-BitTorrent flows.

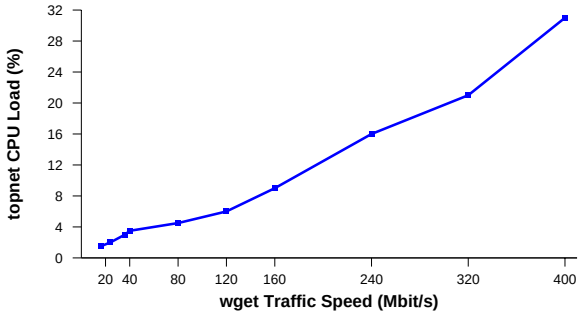


Figure 11: *Topnet* CPU load for several HTTP downloads, each at a different network speed.

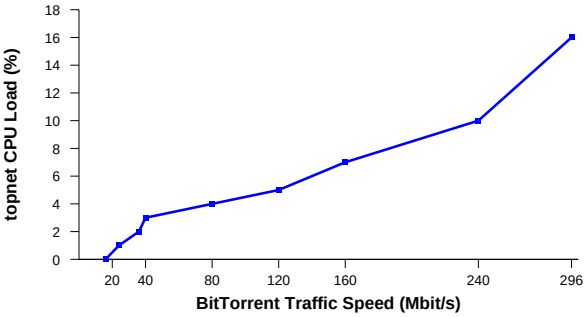


Figure 12: *Topnet* CPU load for several BitTorrent downloads, each at a different network speed. *Topnet* uses less than 10% of the CPU while experiencing download rates at 240 Mbps.

Topnet manages to report the traffic with high accuracy (differences are around 1%). Also in the case of the wget download, where the traffic rate experiences large changes between the measurement periods, *Topnet* follows these changes.

Comparison to Other Traffic Measurement Tools

To further evaluate the measurement accuracy of *Topnet* we installed *Topnet* on a server and monitored outgoing traffic for several hours. The server machine was running mapid, a distributed monitoring daemon, which allows users to run remote monitoring applications. That is, it provides the monitoring capabilities and allows the user to get the information in which she is interested. As an example, the user may ask from the monitoring daemon to count all packets coming in and out from a specific subnet, and periodically will ask for the results. We used tcpdump to log all the traffic coming in and out of the machine using mapid's port. At the same time we were running *Topnet* on the machine measuring the traffic for the specific process number. The comparison of the two traffic measurements is shown in Figure 9. *Topnet*, manages to follow the traffic consumption experienced by the server process. The figure shows one minute averages of the traffic during a period of seven hours.

Performance Measurements

The second part of our evaluation identifies and quantifies the performance overhead that *Topnet* induces during the monitoring process.

The first performance measurement was done in the testbed during the nuttcp experiment presented in the previous subsection. We used the systems' top utility to measure the CPU overhead induced by *Topnet* during the different nuttcp traffic exchanges. The results are presented in Figure 10. The *x* axis plots the exchanged traffic rate in Mbps and the *y* axis shows the average CPU percentage used by *Topnet* during each measurement as a function of the traffic measured. We see that the CPU overhead of *Topnet* is almost proportional to the traffic measured. This should be expected as the main overhead of *Topnet* is attributed to the libpcap library, which is being used to read the packets from the network card. We observe, however, that for low network traffic, i.e., under 200 Mbps, the CPU usage is below 10%.

To evaluate the performance of *Topnet* using real world traffic, we once again used HTTP and BitTorrent traffic. As in earlier experiment, we used wget to download the Linux ISO image mentioned earlier. To understand the impact of network traffic on the performance overhead of *Topnet*, we rate-limited wget to download speeds ranging from 20 Mbps to 400 Mbps. Figure 11 shows the CPU load used by *Topnet* for the different download rates used by wget. As seen previously, the performance of *Topnet* is proportional to the network traffic monitored, staying below 10% as long as the traffic received is less than 200 Mbps.

We repeated the previous experiment using the BitTorrent client, instead of wget. The results, shown in Figure 12, follow the same pattern: i.e., the performance overhead of *Topnet* is proportional to network traffic observed.

Our experiments with a commodity PC suggest that as long as the network traffic observed is relatively low, i.e., up to a few tens Mbps, the performance overhead of *Topnet* is usually below 5%, which we consider acceptable for occasional use, given the information it provides the administrator. We understand though that under very heavy traffic, i.e., several hundreds of Mbps, the performance penalty of *Topnet* may become significant and the tool might experience packet losses. Current trends, however, suggest that this performance overhead, which is completely attributed to the overhead of the packet capture library, tends to decrease with time, as more optimized versions of libpcap are being implemented and deployed (PFring [16], pcap-mmap [48]). Using the aforementioned versions of libpcap, the overhead from copying packets from system to user level decreases, and any packet loss disappears. In our current case, since no packet loss was experienced, running *Topnet* compiled with pcap-mmap gave the same performance measurements.

Memory Usage

As mentioned earlier, *Topnet* uses a hash table to hold all active flows on the machine. During our experiments, where the number of active flows was small, *Topnet* experienced memory overhead below 1%, in all cases.

Reporting Period

In all of our experiments we used 1 second as a reporting period from *Topnet*. Changing the reporting period to a larger number of seconds does not affect the performance overhead of the application, since the CPU utilization is due to libpcap, for reading packets from the network and transferring them to the application. Though, increasing the reporting period may smooth the reported network measurement and reduce any discrepancies observed.

Use Cases

In this section we discuss various real world scenarios in which *Topnet*'s usage could be beneficial. This discussion unveils the strength of *Topnet* in every day situations and incidents. We explore the use of *Topnet* by three target groups: administrators, researchers and end users. In every use case, we focus on the convenience provided to the end user by *Topnet*.

Administrators

Observing system performance is vital to the system administrator. The classic UNIX top utility is designed towards that direction, giving summaries for CPU and memory consumption per process. Our extended top, *Topnet*, broadens this model by adding inspection for network resources consumption per process. Understanding the network usage of an individual process has become increasingly important, if not more important than, as understanding CPU and memory resource consumption.

A summary of network consumption per host, something that most operating systems provide by default, is not sufficient. There is a need for greater *per-process* granularity. This is especially important for tracking security threats like rootkits, trojan horses, and other malware, which consume network resources in a stealthy way, without altering the overall condition of the host as far as the network consumption is concerned. More precisely, a malicious program that performs IP/port scanning or communicates with other members of a Botnet, may never be spotted by just observing the total outbound traffic of a host, since the traffic exported by the malicious program has a low volume. However, this malicious program will have active flow records in the *Topnet* utility and thus it will be easily spotted by an administrator.

Researchers

System research, very often, involves the inspection and measurement of network capabilities of an application. Sometimes, the researcher has to correlate consumption in all available resources, namely CPU, memory and network. The most well known tactic is

to use multiple tools for the inspection of different kind of resources. For example, the top UNIX utility may be used for the inspection of CPU/memory resources and an application trace collected with tcpdump, or another packet capturing tool, can be used to inspect the network utilization experienced by the application.

Topnet combines all above tasks to one program able to monitor in real-time an application's utilization for all kind of resources. Thus, *Topnet* offers a convenient way for the researcher to inspect in real-time the behavior of an application and gives immediately an indication of its resource consumption.

End Users

End users' systems have drastically evolved and become more sophisticated as far as the amount of running applications and their complexity is concerned. In addition, end users are frequently mobile, utilizing wireless access media. As an example, excessive bandwidth consumption might significantly reduce the lifetime of the battery of a user's laptop. A tool such as *Topnet* that can inform the user of such events is considerably valuable.

We believe that the increased complexity of a modern home machine (a desktop PC, or a laptop) has driven the home user to cope with tasks that are usually performed by an administrator. The user, in a sense, frequently becomes *an administrator of her own system*. Thus, we propose *Topnet* as a convenient tool for the quick inspection of the resource utilization of a user's machine. With increasing popularity of networking applications (Web surfing, IM, file-sharing, etc.), inspecting the network activity per process significantly changes the picture the user has about the actual condition of her machine.

Communication Patterns

The cases above depict various classes of users that receive benefit and convenience from the *Topnet* tool. In contrast with traditional programs that experience network utilization upon user activity, modern applications remain active even when the user is not explicitly using them. This *background behavior* causes significant network activity, without easily being spotted by the user.

In Table 2 we list a few representative applications and their background behaviors, i.e., network activity not explicitly initiated by the user. In all these application profiles, *Topnet* can easily expose such background network activity.

Future Work

Topnet fills a substational gap to current system administration by monitoring network demands at the process level. However, we believe that *Topnet* can be further enhanced. In this Section we list some features we plan to implement in following versions of the tool.

Security Enhancements

As we have pointed out in Use Cases Section, *Topnet* can be used for tracking security threats, like rootkits, trojan horses and other malware. In such a case, the administrator can inspect network consumption, related to malicious activity rather than a legitimate user application. We, consider this feature important to expose security threats. Thus, we plan to implement an accounting feature in *Topnet* in order to give system administrators the opportunity to get a better global view of the monitored network.

We plan to have a *Topnet* mode that logs all network activity per process to a file. Then, it would be easy to create a script that collects all log files from *Topnet*-enabled hosts running in the network and combine the information to one file. This file can then be processed for malicious activity. For example, someone can search, for the same process running in a series of hosts, consuming constantly network resources. If this process is not known, this observation might indicate that the machines running the specific process are suspicious for participating to a BotNet.

Application	Background Behavior
BitTorrent	A host becomes a seeder
File Sharing	Routing/Downloads/ Uploads are served
Peer-to-Peer	A host becomes a relay
Web 2.0	Polling in periodic intervals for Server data
Social Utilities	Polling in periodic intervals for Server data

Table 2: Application network profiles, when they are not explicitly used.

In general, logging process network activity to a file, and aggregating the information to a central place, may substantially assist in spotting security issues. Analyzing the collected information might not be trivial, but we believe that there are some practical heuristics that may be applied and reveal anomalous host behavior.

Portability

Our implementation of *Topnet* is currently tested on a number of Linux distributions. Though, in the near future, we plan to port *Topnet* to various different UNIX flavors, as well as Windows. *Topnet* is based on two widely used tools. libpcap is available and supported on all UNIX flavors and also on Windows systems. The source code of netstat is available for Linux, and the same information can be retrieved in Windows using the GetTcpTable and GetUdpTable MSDN functions.[6] We are currently exploring ways to achieve the functionality given by netstat on other UNIX flavors,

[6] http://msdn.microsoft.com/en-us/library/aa366071(VS.85).aspx

using tools like lsof [6] and libproc [2], in order to make *Topnet* more portable.

Kernel Space. The network has become a perpetually used resource of every system. Our implementation aim was to achieve per-process network monitoring on a host machine without changing the OS kernel. However, we believe that this data should be provided by the OS kernel, so that it would be completely accurate and more efficient and would also allow for a new generation of monitoring tools with the network as an inherently monitored resource.

Deployment. Finally, one of our major concerns is the deployment of the tool in large scale networks, that have different purposes. We plan to install the tool in enterprise, research and academic networks. Each different network class has specific properties. Deploying *Topnet* in such environments will give us a better understanding of the tool's potential usage. It will also stress the utility in real environments that exhibit realistic system and network load.

Conclusion

In this paper, we presented a process-centric approach to network resource attribution for UNIX computers. We described the design and implementation of *Topnet*, an extension of the familiar top utility. While top focuses on process attributes like CPU and memory usage, *Topnet* provides information about network usage as well.

In contrast to prior approaches, we implemented *Topnet* without needing any kernel modifications, using only user-level functionality based on the libpcap library. Our evaluation suggests that the performance overhead incurred by *Topnet* is reasonable in most situations.

Overall, *Topnet* provides a familiar and accurate way to attribute network traffic to individual processes, enabling users to gain a process-centric view of the network traffic in their systems.

Acknowledgments

This work was supported in part by the project CyberScope, funded by the Greek Secretariat for Research and Technology under contract number PENED 03ED440. Elias Athanasopoulos is also funded by the Ph.D. Scholarship Program of Microsoft Research Cambridge. We are grateful to our anonymous reviewers for their thoughtful and valuable feedback. We also want to thank our shepherd, Dave Plonka, for his collaboration in improving the paper.

Author Biographies

Antonis Theocharides received his Bachelor in Computer Science from the University of Crete in 2008. He is continuing his studies at the University College of London. He can be reached at atheohar@gmail.com .

Demetres Antoniades received his M.Sc. and B.Sc. degrees in Computer Science from the University of Crete, in 2005 and 2007 respectively. He is currently a Ph.D. candidate in Computer Science in the same university. Since 2004, he is also with the Institute of Computer Science, FORTH, Crete, where he currently works in the Distributed Computing Systems Lab. His main research interests include network monitoring and traffic classification.

Michalis Polychronakis is a Ph.D. candidate in the Computer Science department at the University of Crete, Greece. He received the M.Sc. and B.Sc. degrees in Computer Science from the same university in 2003 and 2005, respectively. Since 2002, he is also with the Institute of Computer Science, FORTH, Crete, where he currently works in the Distributed Computing Systems Lab. His main research interests include systems and network security, intrusion detection, and network monitoring.

Elias Athanasopoulos received a B.Sc. in Physics from the University of Athens and an M.Sc. in Computer Science from the University of Crete, in 2004 and 2006 respectively. He is currently a Ph.D. candidate in Computer Science with the University of Crete. He has published in ACNS, CMS, EC2ND, IWSEC and ISC. He is a Research Assistant with FORTH and he has received a Ph.D. scholarship from Microsoft Research (Cambridge) for the period 2008-2011. He interned in Microsoft Research during the summer of 2007.

Prof. Evangelos Markatos (markatos@ics.forth.gr) received his diploma in Computer Engineering from the University of Patras in 1988, and the M.S. and Ph.D. degrees in Computer Science from the University of Rochester, NY in 1990 and 1993 respectively. Since 1992, he is an associated Researcher at the Institute of Computer Science of the Foundation for Research and Technology – Hellas (ICS-FORTH) where he is currently the founder and head of the Distributed Computing Systems Laboratory. He conducts research in several areas including distributed and parallel systems, the World-Wide Web, Internet Systems and Technologies, as well as Computer and Communication Systems Security. He is currently the project manager of the LOBSTER and NoAH projects, both funded in part by the European Union and focusing on developing novel approaches to network monitoring and network security. Since 1992, he has also been affiliated with the Computer Science Department of the University of Crete, where he is currently a full Professor.

Since 2001 Professor Markatos has been the head of the W3C (World Wide Web Consortium) Office in Greece, one of only 17 such offices around the world. Since 2005, he serves as a member of the Permanent Stakeholders Group of ENISA, the European Network and Information Security Agency.

Bibliography

[1] *ATOP – System & Process Monitor*, http://www.atcomputing.nl/Tools/atop/home.html .

[2] *libproc*, http://opensolaris.org/os/community/observability/process/libproc/ .

[3] *nuttcp Official site*, http://www.lcp.nrl.navy.mil/nuttcp/ .

[4] *RFC 4898 – TCP Extended Statistics MIB*, http://www.ietf.org/rfc/rfc4898.txt .

[5] *The netfilter.org Project*, http://www.netfilter.org .

[6] Abell, V., *lsof – LiSt Open Files*.

[7] Anagnostakis, K. G., S. Ioannidis, S. Miltchev, J. Ioannidis, M. B. Greenwald, and J. M. Smith, "Efficient Packet Monitoring for Network Management," *Proceedings of the Eighth IFIP/IEEE Network Operations and Management Symposium (NOMS)*, pp 423-436, April, 2002.

[8] Antoniades, D., M. Polychronakis, S. Antonatos, E. P. Markatos, S. Ubik, and A. Øslebø, "Appmon: An Application for Accurate per Application Traffic Characterization," *Proceedings of IST Broadband Europe 2006 Conference*, December, 2006.

[9] Bernaille, L., I. Akodkenou, A. Soule, and K. Salamatian, "Traffic Classification on the Fly," *ACM SIGCOMM Computer Communication Review*, Vol. 36, Num. 2, pp. 23-26, 2006.

[10] Bos, H., W. de Bruijn, M. Cristea, T. Nguyen, and G. Portokalidis, "FFPF: Fairly Fast Packet Filters," *Proceedings of OSDI'04*, 2004.

[11] Check Point Software Technologies Ltd., *ZoneAlarm*, http://www.zonealarm.com/store/content/home.jsp .

[12] Cisco Systems, *Cisco Network Analysis Module Software*, http://www.cisco.com/en/US/products/sw/cscowork/ps5401/index.html .

[13] Constantinou, F. and P. Mavrommatis, "Identifying Known and Unknown Peer-to-Peer Traffic," *Proc. of Fifth IEEE International Symposium on Network Computing and Applications*, pp. 93-102, 2006.

[14] Crotti, M. and F. Gringoli, "Traffic Classification Through Simple Statistical Fingerprinting," *ACM SIGCOMM Computer Communication Review*, Vol. 37, Num. 1, pp. 5-16, 2007.

[15] Crotti, M., F. Gringoli, P. Pelosato, and L. Salgarelli, "A Statistical Approach to IP-level Classification of Network Traffic," *ICC'06, IEEE International Conference on Communications*, 2006.

[16] Deri, L., *PF_RING* http://www.ntop.org/PF_RING.html .

[17] Deri, L. and S. Suin, "Ntop: Beyond Ping and Traceroute," *Proceedings Tenth IFIP/IEEE Workshop on Distributed Systems: Operations and Management*, pp. 271-283.

[18] Deri, L. and S. Suin, "Effective Traffic Measurement Using ntop," *Communications Magazine, IEEE*, Vol. 38, Num. 5, pp. 138-143, 2000.

[19] Dischinger, M., A. Haeberlen, K. P. Gummadi, and S. Saroiu, "Characterizing Residential Broadband Networks," *IMC '07: Proceedings of the Seventh ACM SIGCOMM Conference on Internet Measurement*, pp. 43-56, 2007.

[20] Estan, C., S. Savage, and G. Varghese, "Automatically Inferring Patterns of Resource Consumption in Network Traffic," *Proceedings of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, pp. 137-148, 2003.

[21] Fink, G., V. Duggirala, R. Correa, and C. North, "Bridging the Host-Network Divide: Survey, Taxonomy, and Solution," *Proceedings of the 20th Conference on Large Installation System Administration Conference*, pp. 247-262, 2006.

[22] Fink, G., P. Muessig, and C. North, "Visual Correlation of Host Processes and Network Traffic," *Visualization for Computer Security (VizSec)*, 2005.

[23] Goldszmidt, G. and Y. Yemini, "Distributed Management by Delegation," *Proceedings of the 15th International Conference on Distributed Computing Systems (ICDCS)*, pp. 333-340, 1995.

[24] Hughes, E. and A. Somayaji, "Towards Network Awareness," *Proceedings of the 19th Large Installation System Administration Conference (LISA '05)*, pp. 113-124.

[25] IBM, *Aurora – Network Traffic Analysis and Visualization*, http://www.zurich.ibm.com/aurora .

[26] Inoue, H., D. Jansens, A. Hijazi, and A. Somayaji, "NetADHICT: A Tool for Understanding Network Traffic," *Proceedings of the 21st Conference on Large Installation System Administration*, 2007.

[27] Insolvibile, G., "Kernel Korner: The Linux Socket Filter: Sniffing Bytes Over the Network," *The Linux Journal*, p. 86, June, 2001.

[28] Ioannidis, S., K. G. Anagnostakis, J. Ioannidis, and A. D. Keromytis, "xPF: Packet Filtering for Low-Cost Network Monitoring," *Proceedings of the IEEE Workshop on High-Performance Switching and Routing (HPSR)*, pp. 121-126, May, 2002.

[29] Karagiannis, T., A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos, "File-sharing in the Internet: A Characterization of P2P Traffic in the Backbone," *University of California, Riverside, USA, Tech. Rep*, 2003.

[30] Karagiannis, T., A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos, "Is P2P Dying or Just Hiding," *IEEE Globecom*, 2004.

[31] Karagiannis, T., A. Broido, and M. Faloutsos, "Transport Layer Identification of P2P Traffic," *Proceedings of the Fourth ACM SIGCOMM Conference on Internet Measurement*, pp. 121-134, 2004.

[32] Karagiannis, T., K. Papagiannaki, and M. Faloutsos, "BLINC: Multilevel Traffic Classification in

the Dark,'' *ACM SIGCOMM Computer Communication Review*, Vol. 35, Num. 4, pp. 229-240, 2005.

[33] Lakkaraju, K., W. Yurcik, and A. Lee, ''NVisionIP: Netflow Visualizations of System State for Security Situational Awareness,'' *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, pp. 65-72, 2004.

[34] LeFebvre, W., ''Kernel Mucking in top,'' *LISA '94: Proceedings of the Eighth USENIX Conference on System Administration*, pp. 47-56, 1993.

[35] Malan, G. R. and F. Jahanian, ''An Extensible Probe Architecture for Network Protocol Performance Measurement,'' *Proceedings of the ACM SIGCOMM '98 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communication*, pp. 215-227, 1998.

[36] McCanne, S. and V. Jacobson, ''The BSD Packet Filter: A New Architecture for User-level Packet Capture,'' *Proceedings of the Winter 1993 USENIX Conference*, pp. 259-270, January, 1993.

[37] McCanne, S., C. Leres, and V. Jacobson, *libpcap*, Lawrence Berkeley Laboratory, Berkeley, CA, http://www.tcpdump.org/ .

[38] McPherson, J., K. Ma, P. Krystosk, T. Bartoletti, and M. Christensen, ''PortVis: A Tool for Port-Based Detection of Security Events,'' *Proceedings of the 2004 ACM Workshop on Visualization and Data Mining for Computer Security*, pp. 73-81, 2004.

[39] Moore, A. and K. Papagiannaki, ''Toward the Accurate Identification of Network Applications,'' *Proceedings: Passive And Active Network Measurement: Sixth International Workshop, PAM 2005*, March 31-April 1, 2005.

[40] Objective Development Software GmbH, *Little Snitch 2*, http://www.obdev.at/products/littlesnitch/index.html .

[41] Oetiker, T., ''MRTG – The Multi Router Traffic Grapher,'' *Proceedings of the 12th USENIX Conference on System Administration*, pp. 141-148, 1998.

[42] Papagiannaki, K., M. D. Yarvis, and W. S. Conner, ''Experimental Characterization of Home Wireless Networks and Design Implications,'' *INFOCOM*, 2006.

[43] Plonka, D., ''Flowscan: A Network Traffic Flow Reporting and Visualization Tool,'' *Proceedings of the USENIX Fourteenth System Administration Conference LISA XIV*, 2000.

[44] Polychronakis, M., E. Markatos, K. Anagnostakis, and A. Øslebø, ''Design of an Application Programming Interface for IP Network Monitoring,'' *Network Operations and Management Symposium (NOMS 2004, IEEE/IFIP)*, pp. 483-496 2004.

[45] Sen, S., O. Spatscheck, and D. Wang, ''Accurate, Scalable In-Network Identification of P2P Traffic Using Application Signatures,'' *Proceedings of the 13th International Conference on World Wide Web*, pp. 512-521, 2004.

[46] Trimintzios, P., M. Polychronakis, A. Papadogiannakis, M. Foukarakis, E. Markatos, and A. Øslebø, ''DiMAPI: An Application Programming Interface for Distributed Network Monitoring,'' *Proceedings of the Tenth IEEE/IFIP Network Operations and Management Symposium (NOMS)*, 2006.

[47] Warren, P. and C. Lightfoot, *iftop*, http://www.ex-parrot.com/pdw/iftop/ .

[48] Wood, P., *MMAP libpcap*, http://public.lanl.gov/cpw/ .

[49] Yin, X., W. Yurcik, M. Treaster, Y. Li, and K. Lakkaraju, ''VisFlowConnect: Netflow Visualizations of Link Relationships for Security Situational Awareness,'' *Proceedings of the 2004 ACM workshop on Visualization and Data Mining for Computer Security*, pp. 26-34, 2004.

[50] Zuev, D. and A. Moore, ''Traffic Classification Using a Statistical Approach,'' *Proccedings: Passive And Active Network Measurement: Sixth International Workshop, PAM 2005*, March 31-April 1, 2005, 2005.