Computer Science Department
University of Crete

Appmon: An Application for Accurate Per-Application Network Traffic Characterization

Master's Thesis

Demetres Antoniades

October 2007
Heraklion, Greece

University of Crete
Computer Science Department

Appmon: An Application for Accurate Per-Application Network Traffic Characterization

Thesis submitted by
Demetris Antoniades
in partial fulfillment of the requirements for the
Master of Science degree in Computer Science

THESIS APPROVAL

Author: ______________________________

Demetris Antoniades

Committee approvals: ______________________________

Evangelos P. Markatos
Professor, Thesis Supervisor

____

Mema Roussopoulos
Assistant Professor

____

Maria Papadopouli
Assistant Professor

Departmental approval: ______________________________

Panos Trahanias
Professor, Chairman of Graduate Studies

Heraklion, October 2007

Abstract

Accurate per-application network traffic characterization is becoming increasingly difficult in the face of emerging applications that use dynamically negotiated port numbers. At the same time, information about the contribution of different network applications and services to the traffic mix is highly demanded by network administrators for facilitating effective network management and traffic engineering.

In this thesis we present *appmon*, a passive monitoring application for per-application network traffic classification. *Appmon* uses deep packet inspection to accurately attribute traffic flows to the applications that generate them, and reports in real time the network traffic breakdown through a Web-based GUI. *Appmon* manages to classify traffic up to Gigabit speeds and shows a steady performance when monitoring a real network environment. *Appmon* is easy to configure and deploy, and is publicly available as an open source application.

Using *appmon* deployed sensors we were able to collect a large amount of traffic data. We present the results extracted from the analysis of data collected from the academic networks of three different countries and try to understand the trends in real world networks.

Supervisor: Professor Evangelos Markatos

GR

Περίληψη

Ο ακριβής χαρακτηρισμός της κυκλοφορίας του δικτύου ανά την εφαρμογή που τη δημιούργησε, γίνεται όλο και πιο δύσκολος με την εμφάνιση όλο και περισσότερων εφαρμογών που χρησιμοποιούν δυναμικές θύρες. Ταυτοχρόνως, πληροφορίες για τη συμβολή των διαφορετικών εφαρμογών δικτύου στο μίγμα κυκλοφορίας απαιτούνται από τους διαχειριστές δικτύων για τη διευκόλυνση και πιο αποτελεσματική διαχείριση τόσο του δικτύου όσο και της κυκλοφορίας του.

Σε αυτήν την εργασία παρουσιάζουμε το *appmon*, μια εφαρμογή παθητικής επίβλεψης για την ταξινόμηση της κυκλοφορίας του δικτύου ανάεφαρμογή. Το *Appmon* κάνει βαθιά εξέταση των πακέτων για να αποδώσει ακριβώς τις ροές κυκλόφοριας στις εφαρμογές που τις παράγουν, και εκθέτει σε πραγματικό χρόνο την ανάλυση της κυκλοφορίας του δικτύου μέσω ενός WEB GUI. Το *Appmon* κατορθώνει να ταξινομήσει την κυκλοφορία ακόμη και σε Gigabit ταχύτητες και παρουσιάζει σταθερότητα κατά της επίβλεψη ενός πραγματικού δικτύου. Το *Appmon* είναι εύκολο να εγκατασταθει και να επεκταθεί, και είναι διαθέσιμο ως εφαρμογή ανοικτού κώδικα.

Εγκαθιστώντας το *appmon* σε αρκετά συστήματα εποπτείας είμαστε σε θέση να μαζέψουμε ένα μεγάλο αριθμό πληροφοριών για τη κυκλοφορίας του δικτύου. Παρουσιάζουμε αποτελέσματα από την ανάλυση πληροφοριών που συλλέκτικαν από τα ακαδημαϊκά δίκτυα τριών διαφορετικών χωρών και προσπαθούμε να καταλάβουμε τις τάσεις στα πραγματικά δίκτυα.

Επόπτης: Καθηγητής Ευάγγελος Μαρκάτος

Acknowledgments

I am deeply grateful to my supervisor, Professor Evangelos Markatos, for his invaluable advice and assistance. He has been extremely patient in explaining the concepts and ideas in the simplest way, and his encouraging words and positive attitude always gave me the strength to go on.

A big thanks to Dr. Kostas G. Anagnostakis (I2R/Singapore), for his invaluable help and cooperation.

My special thanks and best regards to my colleagues in the Distributed Computing Systems Laboratory (ICS/FORTH). Working with them was enjoying and always interesting. *"Players win games, teams win championships."* When the team is full with friendship, things get even better: Elias Athanasopoulos, Spiros Antonatos, Michalis Polychronakis, Manos Athanatos, Antonis Papadogiannakis, Dimitris Koukis, Christos Papachristos, Nikos Nikiforakis, Alexantros Kapravelos and Jason Polakis.

I would like to thank my family and friends for the support and patience they showed all these years.

To my parents

This thesis is based on the paper: **Appmon: An Application for Accurate per Application Network Traffic Characterization**, authored by Demetres Antoniades, Michalis Polychronakis, Spiros Antonatos, Evangelos P. Markatos, Sven Ubik and Arne Øslebø. The paper was accepted for the proceedings of the IST Broadband Europe 2006 Conference in December 2006 (Geneva).

Work done in the thesis is also published at the paper: **ABW - Short-timescale passive bandwidth monitoring**, authored by Sven Ubik, Demetres Antoniades, and Arne Øslebø and accepted for the proceedings of the ICN 2007, April 2007, Martinique, France.

Contents

1 Introduction **1**

1.1 The Emerging Need for Network Traffic Characterization . . . 2

1.2 Contributions 5

1.3 Thesis Outline 5

2 Application Design **7**

2.1 Classification Algorithm 7

2.2 Long-term Statistics 11

2.3 Graphical User Interface 12

2.3.1 Web Interface 12

2.3.2 Batch text mode interface 15

2.4 Implementation 16

2.4.1 Implementation within the Monitoring API 17

3 Experimental Evaluation **19**

3.1 Performance 19

3.2 Real World Experiments 21

3.2.1 Comparison with port classification 22

3.3 False Positives 23

3.4 Evaluation Remarks 24

4 Real World Observations **27**

4.1 Data Description 27

4.2 Traffic Distribution 28

4.3 Active IP Addresses 29

4.4 “Elephants” in Traffic Contribution 32

4.4.1 Incoming Versus Outgoing Traffic 35
4.5 Summary 38

5 Related Work 39

6 Conclusions and Future Work 41

Appendices 43

A *Appmon* Installation and Configuration Instructions 45
A.1 Database Installation and Configuration 46

B *Appmon* Manual Page 49

C Monitoring API Tracker Library 51

List of Figures

1.1 Traffic distribution of the University of Wisconsin-Madison for the last five years. The figure shows that while port-based traffic classification was effective back in 2002, nowadays it limits classification only in HTTP traffic and leaves a large amount of unaccounted-for traffic, about 50%. 3

2.1 *Appmon* Application Architecture Overview. After capturing and decoding, packets traverse the classification structures (trackers) which "colorize" the packets according to the application that generated them. Database and statistics are then updated accordingly. 8

2.2 *Appmons'* Web Interface. The central frame presents the incoming and outgoing per-Application traffic distribution graph, while the right frame shows the TOP 10 bandwidth consuming IP addresses. The left frame provides a menu for specific protocol view. 13

2.3 Per-Application Bandwidth Usage for the last hour as presented by *appmon*. Each categorized application is shown with a different color. The legend also gives the incoming and outgoing traffic rate of the last measurement interval. 14

2.4 Top 10 incoming traffic IP addresses as presented by *appmon*. For each IP the traffic belonging to a specific application is aggregated. 15

2.5 *Appmon* ncurses console mode interface. The left frame shows the traffic rate for each application during the last measurement interval. The right frame shows the list of TOP 10 incoming and outgoing IP addresses. 16

3.1 The testbed environment used for the performance measurements. *Appmon* runs on a separate machine and traffic created by two different machines is mirrored to the network device monitored by the application. 20

3.2 CPU usage of *appmon* when tested with various traffic rates . 20

3.3 *Appmon* CPU Load Vs. Traffic Load while running on a live monitoring sensor at University of Crete for a period of four days. Each point corresponds to a five minute interval. . . . 21

3.4 *Appmon* Classification Vs. Port-based Classification. Port-based classification is not enough nowadays. 23

4.1 Per Application Incoming Traffic distribution for each of the three organizations during the measurement period 29

4.2 Per Application Outgoing Traffic distribution for each of the three organizations during the measurement period 30

4.3 Active IP Addresses for each of the three networks. The red line shows Incoming Active IP addresses and red line give Outgoing Active IPs. The large difference in the Incoming versus Outgoing Active IP addresses is believed to be due to scanning activities or backscatter traffic 31

4.4 Per-Application Percentage of Active IP Addresses. Client-Server protocols are used by the majority of IP Addresses while Peer-to-Peer protocols appear to be used by 20-40% of the IP Addresses 31

4.5 Percentage of traffic received by each unique IP address (red bars). The (green) line gives the cumulative percentage of traffic while the number of IP addresses increases. 32

4.6 Percentage of traffic transfered by each unique IP address (red bars). The (green) line gives the cumulative percentage of traffic while the number of IP addresses increases. 33
4.7 Percentage of IP addresses that contribute the corresponding percentage of incoming (4.7(a)) and outgoing (4.7(b)) traffic . . 34
4.8 Per-Application Traffic Vs. Active IP addresses. Figure shows the percentage of IP addresses that contributes to the respective portion of traffic. The left group of bars shows results for Incoming traffic and the right one those from Outgoing traffic 36
4.9 Incoming and Outgoing traffic percentage for the top 50 Incoming traffic IP addresses. Symbols are cumulative (actual traffic of an IP is the difference of the corresponding symbol from the preceding one). Figures shows that only a few IP addresses have both large incoming and outgoing traffic percentage, while most upload a dis-analogous number of bytes as it compares with the bytes they download. 37

List of Tables

2.1 Application-Level Protocols classified by *appmon*. These protocols include the most known client-server protocols and also widely used P2P protocols, specialized for file sharing and live streaming. 11

4.1 Number of Incoming and Outgoing traffic observed during the measurement period and number of monitored and active IP addresses 28

Chapter 1

Introduction

Over the last years network traffic usage increases dramatically. Latest studies show that network traffic grows with rates of about 50% every year [1]. Beside the growth on traffic rates, the number of users, hosts, domains and enterprise networks connected to the Internet has been also growing explosively. Along with these continusly increasing numbers of the Internet traffic and usage, comes the deployment of new and massively used applications. The emergence of the Internet came with the deployment of the World Wide Web; since then numerous applications made their appearance and are massively used by Internet users. Applications for distributed file sharing (Peer-to-Peer systems) appeared at the beginning of the 21th century, and also applications for chatting combined with Voice over IP and live streaming of real video, in the last two years.

With these traffic and population increases, network administrators came upon of great difficulties in their everyday task of monitoring their networks. At the early days things where a lot easier, since each application used a predefined port number that gave the way of classifying, rate limiting and/or dropping. But as the users where not satisfied by these administration policies, they came up with new protocols that use random ports in order to communicate and that leaves administrators without the knowledge of what is running in their network.

In this thesis, we present an application for accurate per-application

[1] MINTS:www.dtc.umn.edu/mints/home.html

network traffic classification. Our application manages to distinguish network traffic by the application that generates it and presents the results in an easy to use and manage web interface. Our application, called *appmon*, manages to classify traffic up to Gigabit speeds and shows a steady performance when monitoring a real network environment. *Appmon* is easy to configure and deploy, and is publicly available as an open source application.

Appmon has been deployed in about 15 sensors worldwide, where it is up and running for a sufficient period of several months. Using *appmon* deployed sensors we were able to collect a large amount of traffic data. We present the results extracted from the analysis of data collected from the academic networks of three different countries and try to understand the trends in real-world networks.

1.1 The Emerging Need for Network Traffic Characterization

Both the research community and network administrators lack of publicly available tools able to distinguish network traffic by the application that generates it. Researchers in the traffic classification area need a reference application in order to evaluate new classification approaches. Most researches tend to build customized tools in order to evaluate their approaches, which may result into inconsistent results among different classification methods. On the other hand, one of the most frequent requests of network administrators is to identify the applications and hosts that generate the largest amount of network traffic.

The emergence of peer-to-peer file sharing, multimedia streaming, and conferencing applications has resulted to a substantial increase in the traffic volume, since they transfer a large amount of data. However, monitoring the traffic generated from such applications is becoming increasingly difficult.

Traditionally, traffic attribution to the corresponding applications is performed using the statically assigned port numbers. Widely used network

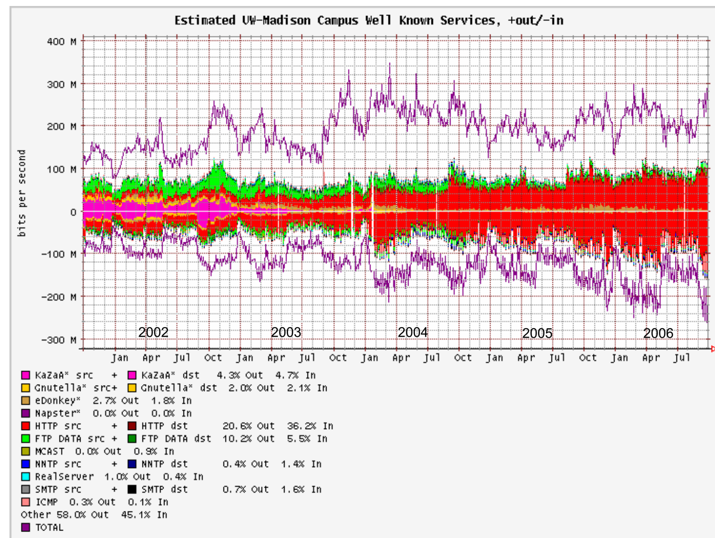


FIGURE 1.1: Traffic distribution of the University of Wisconsin-Madison for the last five years. The figure shows that while port-based traffic classification was effective back in 2002, nowadays it limits classification only in HTTP traffic and leaves a large amount of unaccounted-for traffic, about 50%.

services, like the Web, Telnet, SSH, and many others, are associated with well-known port numbers which can be used for identifying the traffic related with each application. However, many major new applications, including popular, bandwidth-hungry file sharing applications and widely used video and voice conferencing applications, do not use well-known port numbers. Instead, they allocate and use dynamically negotiated ports. Furthermore, some applications masquerade their traffic using pervasive, firewall-friendly protocols, like HTTP, in order to bypass firewall restrictions and make the identification of their traffic harder. Indeed, several widely used applications like BitTorrent [16] and Skype [36] can be configured to operate through port 80, which is usually left open even in environments with strict firewall configurations. Nowadays, the assumption that port 80 traffic is solely HTTP Web traffic is hardly true.

As an example take a look at Figure 1.1. The Figure shows traffic dis-

tribution as seen by the University of Wisconsin-Madison [2], based on information gathered from sampling flow-level summaries over a period of five years. The classification shown on the figure is performed using the statically assigned port numbers. Each application is assumed to operate in one or more well-known ports; and thus all the traffic destined to, or originating from, those ports is assumed to be associated with this application. It is interesting to see that while back in 2002 port based traffic classification managed to separate traffic into several protocols, nowadays the vast majority of traffic is declared as unaccounted-for. Even worse, the classified traffic is limited only to few protocols, that are mostly HTTP and FTP traffic.

It is clear from the above that traditional network monitoring methods for determining per-application network usage are not effective anymore for accurate traffic categorization [29]. Having identified this issue, several researchers have conducted significant work towards alternative ways for network traffic classification. Due to the popularity and high bandwidth demands of peer-to-peer file sharing applications, a significant body of work has focused on the identification and categorization of peer-to-peer application traffic. Initial approaches used deep packet inspection and application signatures for attributing traffic flows to the corresponding applications [24, 34]. Recent approaches identify the applications that generate the traffic either by deriving statistical models for certain protocols [13] or by characterizing the behavior of the host generating this traffic [26].

Motivated by the significance of traffic categorization for effective network management and traffic engineering and aiming at gaining a better understanding of Internet traffic, we have developed *appmon*, a passive network monitoring application for accurate per-application traffic identification and categorization. *Appmon* uses three different approaches for attributing flows to the applications that generate them. First, it searches inside application messages for characteristic application protocol patterns. For certain applications that dynamically negotiate the ports that are going to be used, *appmon* fully decodes the applications protocol to identify the new, dynamically generated port number and then tracks further traffic

[2] http://wwwstats.net.wisc.edu/

flows through these ports. Finally, legacy applications that do not match above filters are categorized based on well-known port numbers and protocols using BPF filters.

1.2 Contributions

This thesis presents an application for accurate network traffic classification. Our application manages to distinguish network traffic by the application that generates it and presents the results in an easy to use and manage web interface. The application is tested both in a testbed and real network environments, where it appears able to classify traffic in high speeds, even at 1 Gb/s with specialized hardware. The application has been installed to more than 15 sensors worldwide and shows stability in its performance and accuracy in the classification done.

Furthermore, in this thesis, we present real world statistics of network traffic usage, collected from three academic networks located at three different countries. We present the most common applications, classified by the number of active IP addresses, and the applications that contribute the majority of the traffic exchanged through the network. Furthermore we explore the existence of “elephant” IP addresses, that is addresses that exchange a vast percentage of the total traffic.

1.3 Thesis Outline

The rest of this thesis is organized as follows. Chapter 2 presents the application we have developed for accurate per-Application Network Traffic Classification. It gives the details for of the classification algorithm and the format we used for presenting the results. Chapter 3 outlines our experimental evaluation both in a controlled environment and in a live sensor where the application is deployed. Chapter 4 presents statistics from real network environments, collected with *appmon* for a period of a few months from some of our sensors. Chapter 5 presents related work on traffic classification. Finally Chapter 6 concludes the thesis.

Chapter 2

Application Design

Appmon passively monitors traffic passing trough a monitored link and categorizes active network flows (identified by the 5-tuple) according to the application that generated them. A network flow is defined as a set of IP packets with the same transport layer protocol, source and destination IP address, and source and destination port (also known as a 5-tuple). Traffic categorization is performed using information from both the packet header and payload.

In this chapter, we present the design of our traffic classification application. First we describe the classification algorithm used by *appmon*. The algorithm is based on the concept of network flows which tries to categorize to the application that generated them. Following we describe the structures we used for storing and displaying traffic statistics

2.1 Classification Algorithm

The classification algorithm operates as follows: *appmon* processes each captured network packet sequentially. For each captured packet, it first checks if the packet belongs to an already categorized network flow. Information about the network flows seen so far is stored into a hash table, along with information about the matching application. *Appmon* keeps the minimal state required in order to reduce the packet processing time. This allows for a “fast path” processing of subsequent packets of an already

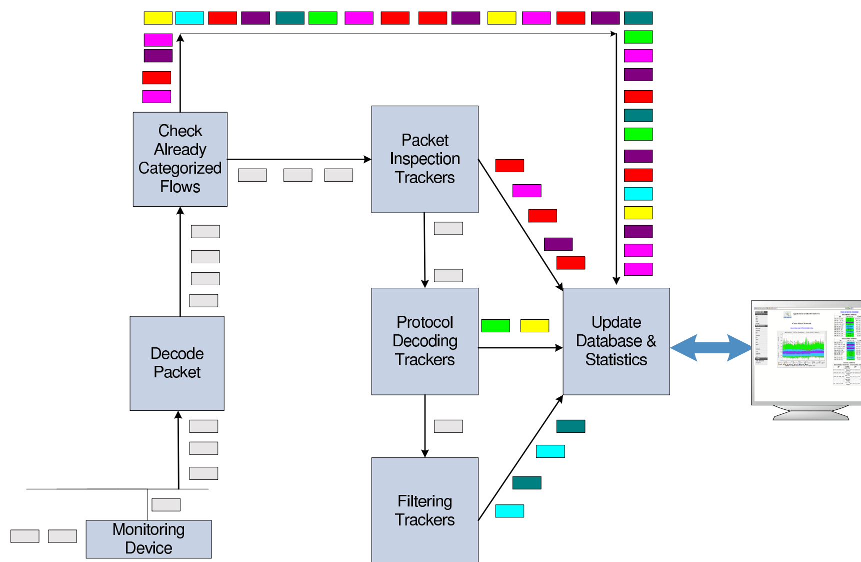


FIGURE 2.1: *Appmon* Application Architecture Overview. After capturing and decoding, packets traverse the classification structures (trackers) which "colorize" the packets according to the application that generated them. Database and statistics are then updated accordingly.

categorized flow, since they will only result to a look up in the hash table for finding the record of the network flow in which they belong, and, consequently, the matching application, without the need for any further processing.

Packets that do not have a matching entry in the hash table are passed down to the next processing level, where each packet is sequentially processed by a set of modules called *application trackers*. Each tracker is responsible for identifying the traffic of a particular application or protocol. There are three different types of application trackers, depending on the traffic classification method: *packet inspection* trackers, *protocol decoding* trackers, and *header filtering* trackers. Figure 2.1 visualizes the packet flow through the application structure from the moment it is captured from the network device.

Packet inspection trackers: Packet inspection trackers are used for tracking application-level protocols, mainly used in peer-to-peer file shar-

ing applications such as Gnutella [22] and BitTorrent. Each packet inspection tracker searches inside packet payloads for characteristic application messages or binary byte sequences that are used by application protocols. These application messages where selected by extensively reverse-engineering the network traffic of popular file sharing applications, as well as by studying the related work on signature-based traffic classification [7, 24, 34]. Although pattern matching inside packet payloads is a quite CPU intensive operation, in most cases the characteristic application patterns, usually protocol control messages, are present in the first 100 bytes of the packet payload, and thus the pattern matching is performed only to this portion of the payload, reducing significantly the processing overhead.

As an example consider the BitTorrent application. For two BitTorrent peers to be able to exchange data, they first must perform the protocol handshake. The handshake starts with number nineteen (decimal) followed by the string 'BitTorrent protocol'. In our inspection tracker for classifying BitTorrent traffic, we search the beginning of each packet for number nineteen followed by the 'BitTorrent protocol' string. This is one of the payload signatures used for BitTorrent classification.

Protocol decoding trackers: Protocol decoding trackers are used for publicly documented application level protocols that operate through well known control ports, but use a dynamically assigned ports for data exchange. For example, in passive FTP, control messages are exchanged through port 21, but actual data transfers are made through a dynamically negotiated port. Protocol decoding trackers operate by fully decoding the application-level messages exchanged through the well-known control port, trying to identify the messages related with the negotiation of port numbers that will be used for future data transfers. When such a message is identified, the number of the dynamic port is extracted and then the tracker will correctly classify the new network flow that is going to be used for the data transfer, since the flow will use this dynamically negotiated port.

Header filtering trackers: If none of the above groups of trackers succeeds in identifying a given packet, then the packet is passed to the header

filtering trackers. Filtering trackers classify traffic based on packet header information such as identifying predefined registered ports [4] and other protocol information. Filtering trackers are implemented using BPF filters [27]. As an example the BPF filter “tcp and (port 80 or port 443)” will classify web traffic, since this are the common ports used by HTTP and HTTPS respectively.

As we have already discussed, several applications masquerade their traffic using widely used, firewall-friendly protocols, like HTTP, in order to bypass firewall restrictions and make identification of their traffic harder. To avoid potential traffic misclassification due to such tricks, trackers are prioritized, with packet inspection trackers applied first, then the protocol decoding trackers, and finally header filtering trackers. When a packet is matched by a tracker, then it is not processed further by subsequent trackers. For example, the BitTorrent tracker has higher priority than the HTTP Web tracker. Thus, the flow of a BitTorrent packet through port 80 will be correctly attributed to the BitTorrent protocol, and not to Web traffic.

If none of the above methods manages to classify the flow in which the packet belongs, then the packet is temporarily considered as unknown, and the application waits for more packets of the same flow in order to classify it.

It is worth mentioning that since most of the application specific patterns are located at the beginning of a flow, the vast majority of the monitored packets will belong to an already active – and thus categorized – network flow. As a result, expensive deep packet inspection operations are performed only to a small subset of the traffic, and *appmon* manages to process traffic loads of several hundred Mbit/s.

We have implemented tracker functions for the majority of application protocols we observe in our monitoring sensors. These tracker functions include widely used Peer-to-Peer protocols, such as BitTorrent [10], Gnutella [3], DirectConnect [1], eDonkey [2], and also the more common Internet applications like HTTP, FTP SSH and others. Latest additions of tracker functions include two streaming P2P application used mostly in China. These are BlueSky and PPStream [6]. Table 2.1 presents the currently implemented protocol trackers in *appmon*. The table contains

Application Protocols		
BitTorrent	eDonkey	Gnutella
Direct Connect	FTP	HTTP
Bluesky	PPStream	SSH
SMTP	DNS	NetBIOS
RTSP	OpenVPN	GRID FTP
BDII	GRIS	

TABLE 2.1: Application-Level Protocols classified by *appmon*. These protocols include the most known client-server protocols and also widely used P2P protocols, specialized for file sharing and live streaming.

only the application-level protocols, including those used by several popular traffic-dominating peer-to-peer applications.

2.2 Long-term Statistics

Appmon outputs results for the traffic generated from each protocol every 10 seconds. At every 10 seconds *appmon* reports the incoming and outgoing traffic rates observed for each protocol during the last measurement interval. It also reports the 10 most bandwidth consuming IP addresses. Although this reporting period is configurable through the *appmon* command line arguments, there is also the need for long-term statistics of the traffic distribution among applications. To fulfill this need, we decided to store *appmon* results in a database.

Our first approach was to use the high-performance data logging and graphing system, rrdtool [8]. Rrdtool is used for storing time series data and is fully appropriate for the data we want to store. Using rrdtool we manage to store data for the traffic distribution, up to the period of the last year using day averages.

During the deployment phase of *appmon* and after discussing with several network administrators came the need for more specific data statistics. We needed a way to answer to more specific questions, like the distribution of a specific IP/subnet during a certain measurement period, that could be

a specific day/week/month or even hours of a specific day.

In order to be able to answer these kind of questions we decided to add extra logging capability to our application. Every one minute we store the traffic distribution for each distinct active IP address in a separate database. In this way using the aggregation and retrieval mechanisms offered by the database we are able to answer to a lot more of questions about the traffic of a specific network, subnet or IP.

2.3 Graphical User Interface

Appmon reports the classification results through two different user interfaces, depending on the requirements of the user. For quick and easy network monitoring, there is a console-mode version which can report the results either through a batch text mode printout, or a more user-friendly ncurses [12] version. For long-term usage, *appmon* provides a powerful GUI accessible using any web browser. The following subsections describe the two different user interfaces.

2.3.1 Web Interface

Appmon reports the per-application traffic distribution through the web interface presented in Figure 2.2. The main page is split into three frames. The central frame presents a graph of the incoming and outgoing traffic portion of each categorized application with a different color, while any remaining non-categorized traffic is shown in light gray. The topmost/bottommost line represents the total observed traffic load.

The information of this frame is better viewed in Figure 2.3 which presents the per-application distribution of the incoming and outgoing traffic at the University of Crete in Greece. The values are expressed in Mb/s, and the graph is updated every 10 seconds. A detailed per-application breakdown of the traffic load, for the last updated time-period, is presented underneath the graph.

The application offers five different time period views of the traffic distribution. The main view presents the per-application traffic distribution of

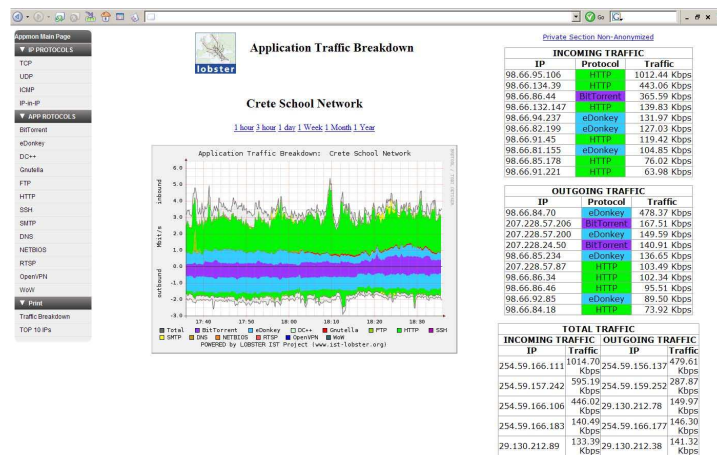


FIGURE 2.2: *Appmons'* Web Interface. The central frame presents the incoming and outgoing per-Application traffic distribution graph, while the right frame shows the TOP 10 bandwidth consuming IP addresses. The left frame provides a menu for specific protocol view.

the last hour. Links also exist for the time period of the last three hours, last day, last week, last month and last year. In this way, one can instantly observe the time-period of interest and make diurnal observations about the per-application traffic behavior along several time continuum.

Besides traffic classification, *appmon* also reports the K top bandwidth consuming IP addresses. This is done by accumulating the traffic of each IP address after every packet is categorized at a specific application. In order to achieve this some extra state is needed. For every protocol we keep a hash table with all the IP addresses that belong to flows marked as belonging to this protocols. For every IP address we keep the number of bytes it transmitted, and the addresses are sorted in descending order according to the amount of traffic seen so far.

The top bandwidth consuming IP addresses are shown in three tables in the right frame of the Web interface. The first two tables contain the IP addresses of the K (10 by default) flows that consumed the largest portion of bandwidth during the last measurement period. Each record contains

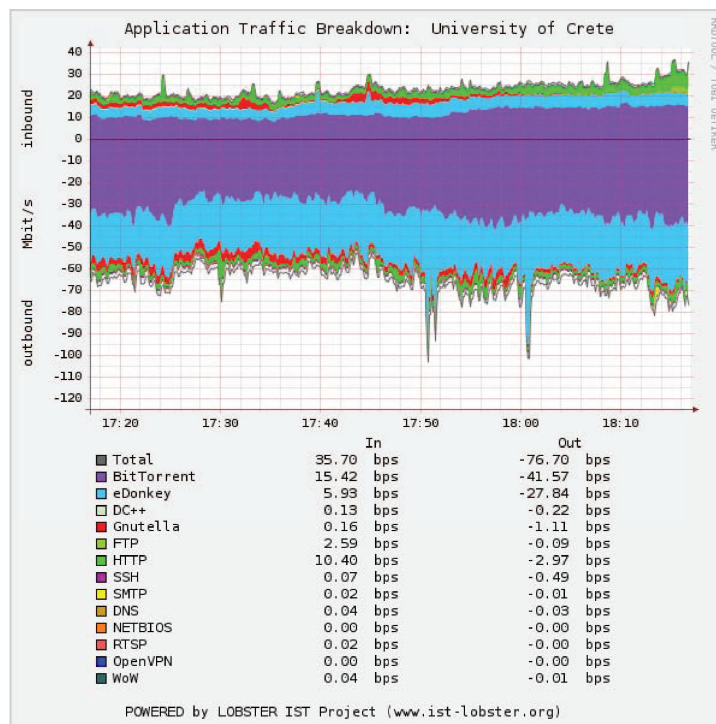


FIGURE 2.3: Per-Application Bandwidth Usage for the last hour as presented by *appmon*. Each categorized application is shown with a different color. The legend also gives the incoming and outgoing traffic rate of the last measurement interval.

INCOMING TRAFFIC		
IP	Protocol	Traffic
98.66.95.106	HTTP	1012.44 Kbps
98.66.134.39	HTTP	443.06 Kbps
98.66.86.44	BitTorrent	365.59 Kbps
98.66.132.147	HTTP	139.83 Kbps
98.66.94.237	eDonkey	131.97 Kbps
98.66.82.199	eDonkey	127.03 Kbps
98.66.91.45	HTTP	119.42 Kbps
98.66.81.155	eDonkey	104.85 Kbps
98.66.85.178	HTTP	76.02 Kbps
98.66.91.221	HTTP	63.98 Kbps

FIGURE 2.4: Top 10 incoming traffic IP addresses as presented by *appmon*. For each IP the traffic belonging to a specific application is aggregated.

information about the application in which the flow belongs to and the exact amount of bandwidth that it consumed. The third table presents the same information at the IP level, which corresponds to the top K IP addresses that consumed the largest portion of bandwidth irrespective of application. Figure 2.4 shows an example of how the top 10 IP addresses are presented through the web interface.

Since information about IP addresses is sensitive and in some cases it may not be desirable to be exposed, *appmon* can anonymize all the IP addresses presented by the Web interface. Address anonymization is performed using prefix-preserving anonymization [40], which preserves subnet information. A non-anonymized version of the TOP IP address is also available for view only by authorized personnel using a login procedure.

Finally, the left frame of the Web interface gives the user the ability to view the traffic of only a selected protocols of interest through a menu with all available protocols.

2.3.2 Batch text mode interface

Since *appmon* is a (network) monitoring application one might want to use it to have instant results in order to solve problems appearing in the network at the given time, and not be interested in keeping long-term statistics. In order to fulfill this requirement we provide a second batch mode interface for *appmon* results. This console-mode version reports the results either

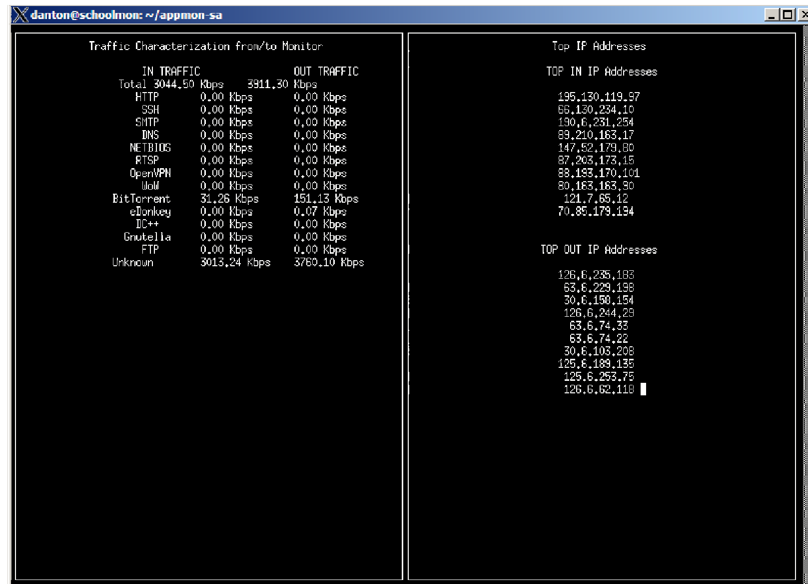


FIGURE 2.5: *Appmon* ncurses console mode interface. The left frame shows the traffic rate for each application during the last measurement interval. The right frame shows the list of TOP 10 incoming and outgoing IP addresses.

through a batch mode printout, or an ncurses version.

Figure 2.5 shows the ncurses console-mode interface. The left half of the screen shows the current (updated every 10 seconds) per-application traffic distribution while the right half shows the TOP K IP addresses.

2.4 Implementation

To be freely available and easy installable, we have implemented *appmon* using only a few external libraries. *Appmon* is build in C language and uses the libpcap packet capturing library [28], which supports live traffic capture using standard Ethernet interfaces, as well as DAG cards.

The crucial pattern matching operation within the packet payload is performed using an implementation of the Boyer-Moore [15] string searching algorithm.

Appmon uses the RRDtool suite [8] for storing measurement data and graphing the traffic distribution. The Round Robin Database provided by

RRDtool efficiently stores time-series data for very long periods in very little space using data aggregation. The database used by *appmon* has a size of few megabytes and can store measurements for a period as long as one year.

The installation of the Web interface requires a web server like Apache with no extra packages. The results are rendered using simple CGI scripts and plain html code.

2.4.1 Implementation within the Monitoring API

Appmon can also operate on top of the Monitoring Application Programming Interface (MAPI) [37]. MAPI is an expressive programming interface for network monitoring that has been developed in the context of the SCAMPI [33] and LOBSTER [11] Projects. MAPI gives the ability for both local, remote and distributed monitoring [39] without the need of user access to the remote monitoring sensors.

MAPI is based on the abstraction of a network flow, which gives the ability to the user to capture all the traffic from a monitoring interface. Then using the functionality provided by MAPI it can filter out the traffic of its' interest. MAPI provides a variety of functions; from simple BPF filters and string searching to regular expression matching and packet anonymization.

The core functionality of *Appmon* has been implemented as a new MAPI function library. The Tracker MAPI function library (`trackflib`) provides implementation of the *packet inspection* and *decoding* tracker functions described in Section 2.1 inside the Monitoring API. A description of all available tracker functions implemented in `trackflib` is presented in Appendix C.

For each application protocol, *appmon* creates a new network flow and applies the appropriate function from the `trackflib` library. It then applies a `BYTE_COUNTER` function and retrieves the results needed for visualization to take place. *Appmon* also makes use of the `TOP` function of MAPI to find the top bandwidth-consuming IP addresses for each protocol. The TOP function is located in `extraflib` library.

The following pseudo-code shows how a typical MAPI flow created by

appmon looks like.

```
1  /* **********************************************
2   * This sample MAPI program monitors a network
3   * interface for Gnutella traffic, measures the
4   * transfered bytes and sorts IP addresses by
5   * the amount of Gnutella traffic they exchanged.
6   ***********************************************/
7
8  /* Create the monitoring flow. This flow captures all the
9   * traffic from the regular nic interface ``eth1''.
10  * A flow consists of a change of functions where each
11  * packet is passed to the next applied function if it
12  * satisfies the condition of its' previous function.
13  */
14 fd = mapi_create_flow("remote.sensor:eth1");
15
16 /* Apply the tracker function for gnutella.
17  * If a packet is classified to be originated by
18  * a Gnutella application is then passed to the
19  * next function; else it is discarded.
20  */
21 mapi_apply_function(fd, "TRACK_GNUTELLA");
22
23 /* Apply byte counter function. This function
24  * aggregates the number of bytes for the packets
25  * provided to it. Since only Gnutella packets are
26  * passed to this function (from the previous one)
27  * it aggregates all Gnutella traffic seen in ``eth1''
28  */
29 fid = mapi_apply_function(fd, "BYTE_COUNTER");
30
31 /* Apply the TOP function. For each unique source IP address
32  * it counts the bytes it transfered into the network. Again
33  * it receives only Gnutella packets.
34  */
35 top_fid  = mapi_apply_function(fd, "TOP",
36    10, TOPX_IP, TOPX_IP_SRC_IP, SORT_BY_BYTES, 0);
37
38 while(1) {
39                 /* read results every 1 second */
40     sleep(1);
41
42    /* Read the current value of the BYTE_COUNTER function */
43     res = mapi_read_results(fd, fid);
44
45     /* Read the current TOP list of IP addresses */
46     res = mapi_read_results(fd, top_fid);
47
48     /* report results */
49 }
```

Chapter 3

Experimental Evaluation

In these chapter we present an experimental evaluation of *appmon* application. The evaluation is based on three directions. First we examine the performance of our application on a controlled environment (Section 3.1). Following we present performance results observed while the application was running on some of our sensors monitoring real live traffic (Section 3.2). Finally we present some tests done to assess the number of false positives reported by the application (Section 3.3).

3.1 Performance

Our first experiment aims at exploring the performance of our application. We used a local testbed consisting of three PCs connected to a gigabit switch, as shown in Figure 3.1. The "Sender" PC generates traffic destined to the "Receiver" PC using the nttcp [5] tool. The traffic from both hosts is mirrored to the third monitoring machine which is running *appmon*.

The configuration of the measurement machine is as follows. We used an Intel Xeon 2.4 MHz, with 512 KB cache and 512 MB memory. The Operating System was Debian Linux with 2.6.15 kernel version. Two kinds of network interfaces were used. A regular Gigabit Ethernet interface (NIC), and a specialized DAG 4.3GE packet capturing card [12].

It is important to mention that nttcp produces artificial traffic by filling the packet payload with random bytes. This is a worst-case traffic load

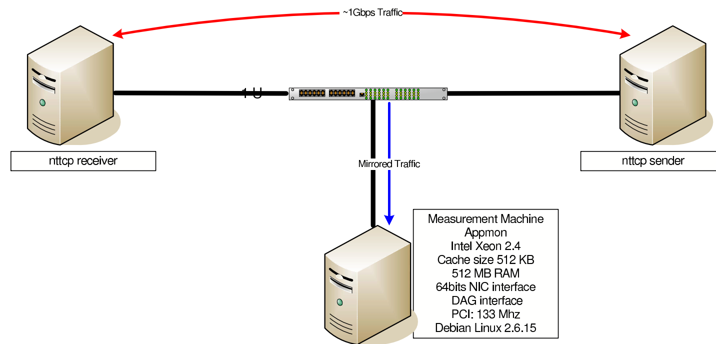


FIGURE 3.1: The testbed environment used for the performance measurements. *Appmon* runs on a separate machine and traffic created by two different machines is mirrored to the network device monitored by the application.

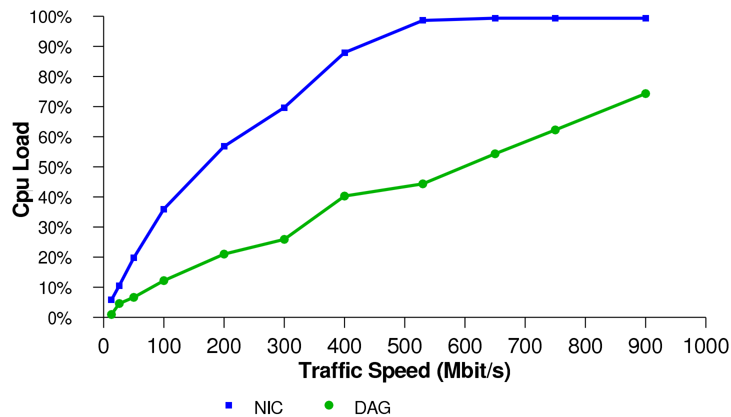


FIGURE 3.2: CPU usage of *appmon* when tested with various traffic rates

for *appmon* since none of the packets matches any of the monitored protocols. Thus, every packet passes through the "slow" processing path, going through all tracker functions, since none of the packets has a matching entry in the hash table, and none of the trackers is able to find a matching packet.

We stressed *appmon* by sending traffic in various speeds. Figure 3.2 shows the results for both NIC and DAG interfaces. As Figure 3.2 implies *appmon* can process up to 500 Mbit/s without any packet loss when running on a regular NIC interface, while it is able to process all 900 Mbit/s when running on top of the DAG card. The results imply that the application

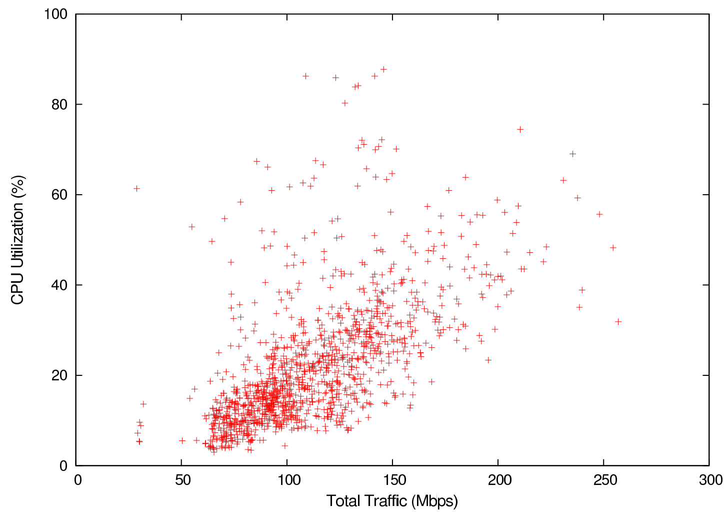


FIGURE 3.3: *Appmon* CPU Load Vs. Traffic Load while running on a live monitoring sensor at University of Crete for a period of four days. Each point corresponds to a five minute interval.

can fully monitor a Gigabit link using a DAG card.

3.2 Real World Experiments

Despite the encouraging results about the performance of *appmon* when running on a controlled testbed environment, we wanted to see how the application will behavior in a real network monitoring environment. Aiming at verifying the performance results of the previous section, we deployed *appmon* in several sensors monitoring various amounts of traffic and measured the performance of the application in terms of cpu usage.

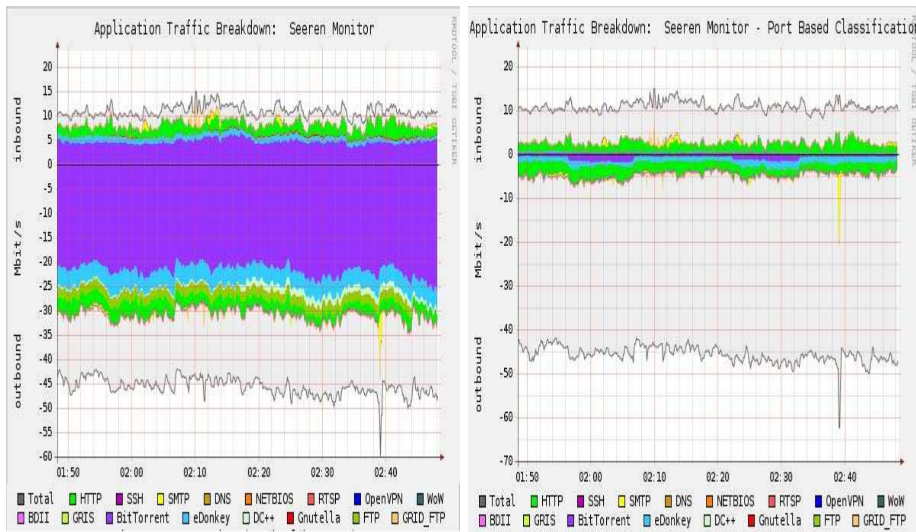
Appmon was running on a sensor at the University of Crete, monitoring the incoming and outgoing traffic from the campus to the Internet. The monitoring machine was an Intel Xeon 3.2GHz, with 2MB cache memory and 1GB main memory, running a Debian Linux, kernel version 2.6.15. The traffic is captured using a DAG 4.2GE passive monitoring card. Along with the traffic load reported by the application, we measured the CPU load of the machine. A new measurement result was produced every 5 minutes for a measurement period of four days.

Figure 3.3 presents the CPU load (y-axis) of the monitoring sensor as

a function of the monitored traffic load (x-axis). Each point corresponds to a five minute interval, computed as the average of the measurements performed every 10 seconds in that interval. The measurement period was four days. *Appmon* has a steady behavior, since the CPU load increases as the traffic load increases. Some corner cases in which the load is increased significantly while the traffic load is low are probably caused due to the almost simultaneous arrival of many new traffic flows that have not yet been categorized.

3.2.1 Comparison with port classification

One of the motivations for this work was that port-based traffic classification is not enough nowadays. It is not able to identify the application that originated the traffic, since most of the applications do not use a predefined standard port. Among *appmons'* goals was to overcome the limitations of port-based classification and manage to "colorize" the traffic distribution. Figure 3.4 shows the fulfillment of this goal. We deployed a version of modified version of *appmon* monitoring traffic from collected from SEEREN [9] beneficiary countries , running in parallel with the original *appmon* application. In this modified version of *appmon* we have replaced the Packet Inspection and Protocol Decoding tracker functions with simple BPF filters for the default port(s) of the corresponding applications, as these are reported from applications' documentation. Figure 3.4(b) shows the traffic classification we would have if only port classification was used. As we can see only a small portion of the traffic is classified while about 90% remains unclassified. On the other hand using *appmons'* functionality the unclassified traffic limits to 30%. Figure 3.4(a) shows the "colorized" traffic distribution that is produced by *appmons'* classification algorithm. As we can see, using *appmon* the classified traffic for BitTorrent and eDonkey has increased and traffic from other protocols, like Direct Connect and Gnutella, not classified by the port-based classification, has been identified.



(A) *Appmon* Classification (B) Port-based Classification

FIGURE 3.4: *Appmon* Classification Vs. Port-based Classification. Port-based classification is not enough nowadays.

3.3 False Positives

For the *appmon* application we define as false positives the flows that belong to a different protocol than the one categorized by *appmon*. We limit our evaluation regarding false positives only to the heavy tracker functions. Those are the trackers for following P2P protocols (Bittorrent, Gnutella, DirectConnect, eDonkey, Bluesky and PPStream) and for the FTP protocol. These tracker functions are the ones searching for specific protocol strings in the packet payload.

As a first step we used three HTTP traces. The traces were captured on a machine monitoring the traffic of the ICS-FORTH Web Server and contain only the traffic originating and designating to the web server. Since the traffic on the traces is explicitly HTTP we do not expect to have any matching entries for other protocols. We passed the traces through each tracker function separately and also through the application as it would be run in a monitoring environment, with all trackers enabled. As expected none of the trackers reported any categorized traffic for any of the traces.

As a second step we used some application specific traces collected by capturing traffic on a host running the application of interest. We used traces captured from BitTorrent, Gnutella, eDonkey, DirectConnect and FTP applications. Before we proceed we have to make clear that these traces were not used during the signature creating procedure and were not used for calibrating *appmon* classification accuracy. This process was done before with different traces and these traces are used only for exposing false positives.

We passed these traces through the same *appmon* tracker functions mention in the previous paragraphs and look for traffic misclassification. The only tracker that reported classified traffic was the tracker for the corresponding application the traces was created from. None of the other trackers reported any traffic.

In both cases, using web traces and application specific traces, *appmon* did not exposed any false positive. Although we expect to have a small percentage of false positives when the application is deployed in real networks. Though, we did our best during the development period to assure that the signatures used where only present in traffic of their corresponding protocol and eliminated signatures that where prone in creating false positives. Nevertheless this result assures us that our application will accurate categorize traffic and would not export any misleading classification results.

3.4 Evaluation Remarks

In this chapter we evaluated our application in terms of performance and classification accuracy. From our controlled performance measurements we can see that *appmon* can classify traffic without any problems up to 500 Mb/s using a regular NIC, while it reaches Gigabit speeds when using a specialized DAG interface.

When used on a real network, *appmon* did not encounter any problems, and was able to to classify the monitored traffic without stressing the monitoring machine.

Finally the accuracy of *appmon* is explored by using some HTTP and application specific traces. In all cases *appmon* categorized traffic to the

corresponding application without any miss-classifications.

Chapter 4

Real World Observations

Among the world-wide *appmon* sensors, and through the cooperation of two projects, Lobster [11] and Seeren [9], we have established a sensor monitoring the network of three institutions; The *Academic & Research Network of Serbia and Montenegro* (AMRES), the *Macedonian Academic & Research Network* (MARNET) and the *Montenegro Research & Education Network* (MREN). The sensor is monitoring a total of 152320 IP addresses, using *appmon*. The database feature of our application was used. Every minute we log into the database the traffic volume created by each IP address. In order to reduce the space requirements we keep records only for the active IP addresses in each measurement time interval.

In the analysis of this Chapter, we illustrate some observations extracted from the data during a measurement period of two months. Apart from the traffic distribution we present data for the number of IP addresses participating in an "application network". We also try to quantify whether the traffic volumes observed are produced equally by all the IP addresses and if the concept of "mice and elephants" is present in the different protocols.

4.1 Data Description

The data presented in this document were selected for a period of two months, from July 06/2007 15:00 to September 07/2007 20:10. During this period *appmon* processed about 63 Terra-bytes of data, originated

	TBytes Received	TBytes Transfered	Total IP Range	Incoming Active IPs	Outgoing Active IPs
Total	17.2	46.2	152320	150957	13933
AMRES	10.5	32.9	139776	138865	10506
MARNET	5.1	12.0	8448	7996	1873
MREN	1.6	1.2	4096	4096	1554

TABLE 4.1: Number of Incoming and Outgoing traffic observed during the measurement period and number of monitored and active IP addresses

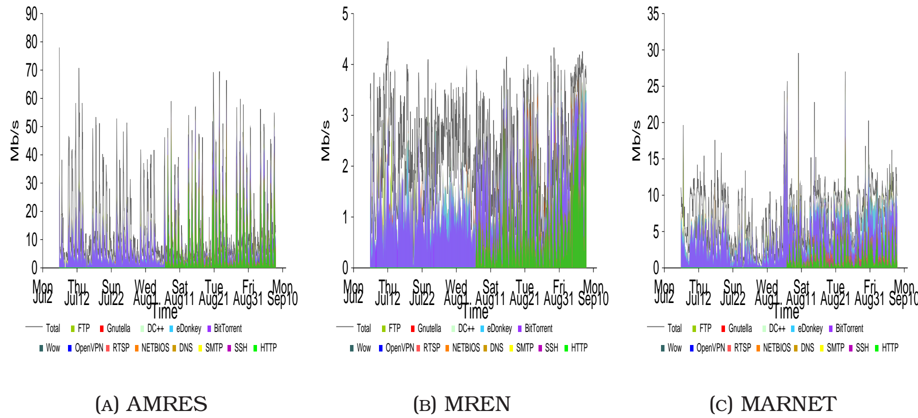
from about 150000 IP addresses. Table 4.1 summarizes the data received and transfered from each network during the measurement period and also gives the number of IP addresses monitored and the number of active IP addresses.

Appmon does not keep any information about the packets exchanged. For each IP address it stores into the database the incoming and outgoing traffic rate experienced in the last measurement interval (1 minute in this case). For each of these IP addresses an analysis of which application protocol originated the traffic is also kept in the database. Extra information about the network flows, like the duration and participating host is not kept.

4.2 Traffic Distribution

Figures 4.1 and 4.2 show the per application distribution of incoming and outgoing traffic, respectively, for each of the three organizations separately. For the first month results are only present for the P2P protocols due to a collection problem we had and overcame afterward. We plot the different protocols using stacked areas, that is the contribution of each protocol is given by getting the difference with the previous protocol. The black line shows the total IP traffic rate. The order that the protocols are presented can be found by following the legend.

As we see BitTorrent and HTTP are the most traffic consuming protocols followed by eDonkey. Other protocols contribute only a small percentage



(A) AMRES (B) MREN (C) MARNET

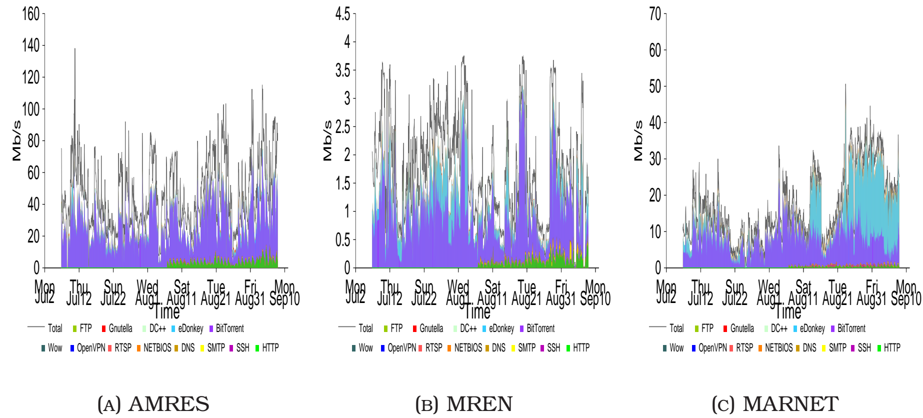
FIGURE 4.1: Per Application Incoming Traffic distribution for each of the three organizations during the measurement period

in the traffic distribution. In both AMRES and MARNET networks outgoing traffic is observed in larger rates (some times even doubled) than incoming traffic. Also, the two main traffic contributing P2P protocols (Bittorent and eDonkey) are responsible for a greater portion of the outgoing traffic than for the incoming traffic. This can be explained by the fact that a user of the application contributes to the protocol for the whole time it is connected to the application network, by seeding other users, even if she does not receives any data at the time.

4.3 Active IP Addresses

In this section we study the percentage of active IP addresses for both incoming and outgoing traffic on the three monitored networks. An IP address is characterized as active if it received or transmitted traffic for at least one time interval during the whole measurement period.

Figure 4.3 shows the number of active IP addresses for each organization separately. The red line shows IP addresses that had incoming traffic, while the green line presents addresses that produced outgoing traffic. As we can see the number of active incoming IP addresses is, at least, an order of magnitude larger than the number of active outgoing IP addresses. We

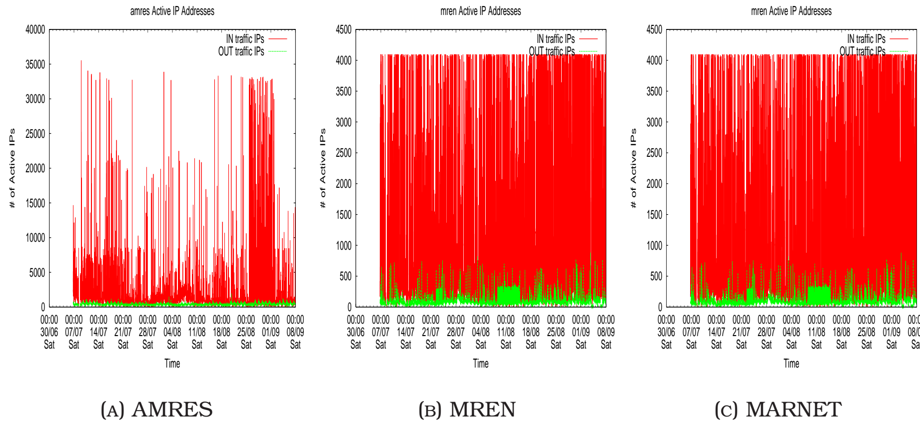


(A) AMRES (B) MREN (C) MARNET

FIGURE 4.2: Per Application Outgoing Traffic distribution for each of the three organizations during the measurement period

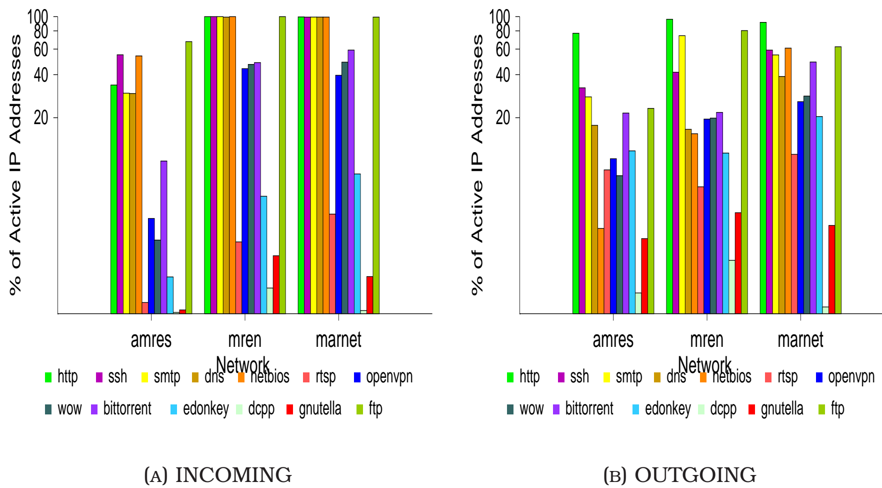
believe the large difference in the number of active IP addresses concerning incoming and outgoing traffic to be due to a large portion of scans and/or backscatter traffic.

Figure 4.4 shows the percentage of IP addresses that where active for each separate application during the measurement period. As we can see from the plots, common protocols (HTTP, MAIL, FTP, NetBIOS, SSH) have a high percentage of incoming active IP addresses (Fig. 4.4(a)). Since the percentages are not the analogous in the case of outgoing active IP addresses, we suspect that these high percentages are due to scanning probes or backscatter traffic. From both figures we can see that WEB, MAIL and FTP are the most common protocols regarding usage from distinct IP address, while among the Peer-to-Peer Protocols BitTorrent seems to be the most used protocol, followed by eDonkey. On the other hand, by looking at Figures 4.1 and 4.2, we see that the P2P applications contribute a vast amount of the traffic generated and received compared to the most popular protocols. This result comes comes to fulfill our forthcoming analysis that a small percentage of IP addresses is responsible for the vast majority of traffic.



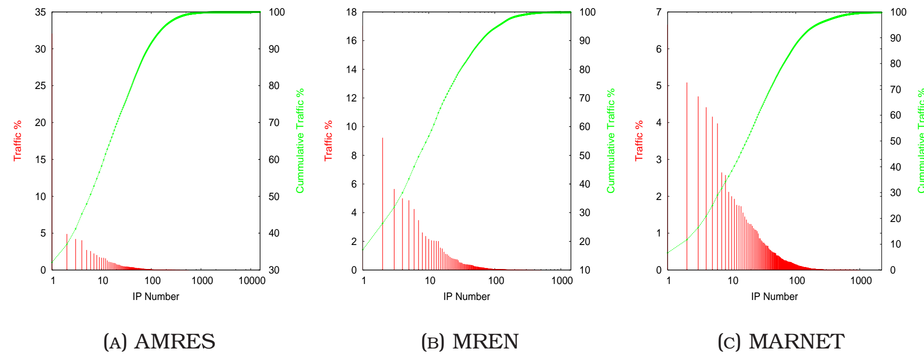
(A) AMRES (B) MREN (C) MARNET

FIGURE 4.3: Active IP Addresses for each of the three networks. The red line shows Incoming Active IP addresses and red line give Outgoing Active IPs. The large difference in the Incoming versus Outgoing Active IP addresses is believed to be due to scanning activities or backscatter traffic



(A) INCOMING (B) OUTGOING

FIGURE 4.4: Per-Application Percentage of Active IP Addresses. Client-Server protocols are used by the majority of IP Addresses while Peer-to-Peer protocols appear to be used by 20-40% of the IP Addresses



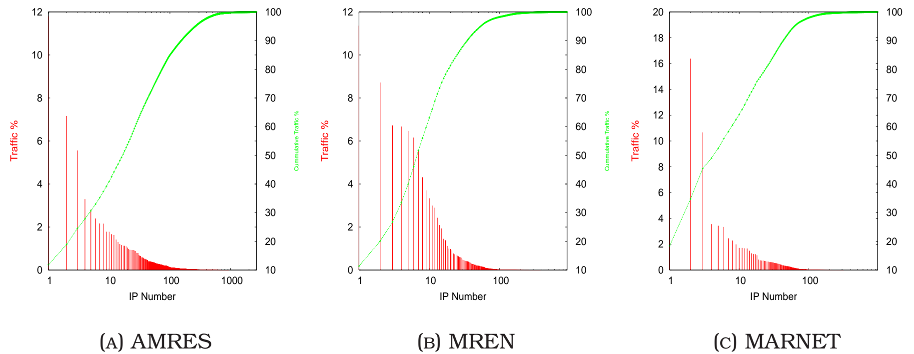
(A) AMRES (B) MREN (C) MARNET

FIGURE 4.5: Percentage of traffic received by each unique IP address (red bars). The (green) line gives the cumulative percentage of traffic while the number of IP addresses increases.

4.4 "Elephants" in Traffic Contribution

In this section we look at the traffic from the granularity of an IP address. We explore whether the contribution of each IP address, as a function of the total traffic distribution, is similar to the other IP addresses. The results we get is that for most of the protocols a very small portion of the IP Addresses contributes the majority of the traffic while the rest generate only a small percentage.

Figure 4.5 shows the number of bytes received from each IP address as a percentage of the total incoming traffic transfered during our measurement period. The X axis presents the number of distinct IP address in descending order, with the IP that transfered the larger amount of bytes as number 1. The (red) bars present the percentage of traffic received by each unique IP addresses, while the (green) line gives the total percentage of traffic received as the number of IP addresses increases. We noticed incoming traffic presence for 150957 distinct IP addresses for the whole network. About 91.99% (138865) of these IP addresses belong to the AMRES Network, 2.71% (4096) belong to MREN Network and 5.31% (7996) to the MARNET Network. The first IP, which is the IP address that transfered the largest amount of traffic, accounts for about 19.55% of the whole number of bytes transfered, while the first 50 IP addresses (that is 0.033% of the total number of distinct IPs)



(A) AMRES (B) MREN (C) MARNET

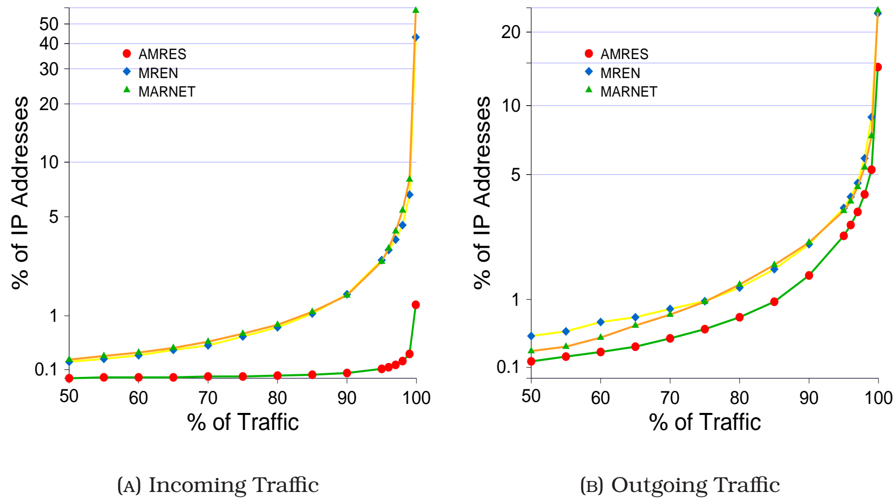
FIGURE 4.6: Percentage of traffic transfered by each unique IP address (red bars). The (green) line gives the cumulative percentage of traffic while the number of IP addresses increases.

cumulatively account for nearly the 65% of the total number of bytes.

For the AMRES network the TOP IP addresses contributes the 32% of the traffic while the first 50 (0.036%) IP addresses accept the 83.23% of the incoming bytes. In the case of the MREN network “elephants” contribute a large portion of the total incoming traffic since 87.3% is received by 50 (1.2%) IP adresses; while the top IP address receives 17.04%. Finally, in the case of MARNET Network the first IP address receives about 6.65%, while the first 50 (0.63%) IP addresses account for the 74.87% of the incoming traffic.

Figure 4.6 shows the outgoing traffic per IP for the three organizations separately. Outgoing traffic was produced from 13933 distinct IP addresses. 75.4% (10506) of these IP addresses belongs to the AMRES Network, 11.15% (1554) to the MREN Network and 12.44% (1873) to the MARNET Network. The top IP address outputs about 8.41% of the total traffic, while the top 50 IP addresses (0.36%) account for 65.77% of the total traffic.

Looking at each organization separately we see that for the AMRES Network the top IP address transfers 11.8% of the networks’ outgoing traffic, while the first 50 (0.48%) IP addresses output 72.7% of the traffic. For the MREN Network 94.3% of the traffic is originated from 50 (3.22%) IP addresses and the top ranked IP address outputs 11.36%. Finally, in the



(A) Incoming Traffic

(B) Outgoing Traffic

FIGURE 4.7: Percentage of IP addresses that contribute the corresponding percentage of incoming (4.7(a)) and outgoing (4.7(b)) traffic

case of the MARNET organization number 1 IP address contributes the 18.42% of the traffic while the 50 (2.67%) first IPs accounts for the 92.3% of the traffic.

To have a better view of the observation done in the previous paragraphs, we decided to examine what percentage of the IP addresses is responsible for what percentage of the traffic. In Figure 4.7 we plot the number of IP addresses (as a percentage of the total active IP address) that contributed the 50% - 99.9% percent of the traffic bytes received (Fig. 4.7(a)) or transmitted (Fig. 4.7(b)). The results extracted from the figures are impressive. For both MREN and MARNET networks the 80% of Incoming and Outgoing traffic is received (transfered) from less than 1.0% of the IP addresses. In the case of the AMRES network this percentage is even smaller, especially for the incoming traffic where 99% of the traffic is received by 0.3% of the IP addresses. A large increase in the IP percentage is observed when moving from 99.0% of the traffic to 99.9%; where the contributing IP addresses are up to 8 times more. This indicates that the majority of the IPs transfer a very small number of bytes in compare with the top IP addresses.

In the same concept, Figure 4.8 analyzes the relationship between IP address percentage and traffic percentage in a per-Application basis. We

use the sum from the three networks for the traffic portion given in these figures. The left bar group in each sub-figure shows IP percentage for incoming traffic and the right one for outgoing traffic. We choose to show the relationship for only three traffic percentage; the two edge cases of our analysis, 50% and 99.9%, and the active IP addresses for 90% of traffic. The last value is chosen since we observe the largest change in the percentage of active IP address. From 50 to 90 percent the number of IPs contributing to the traffic experiences only small increases.

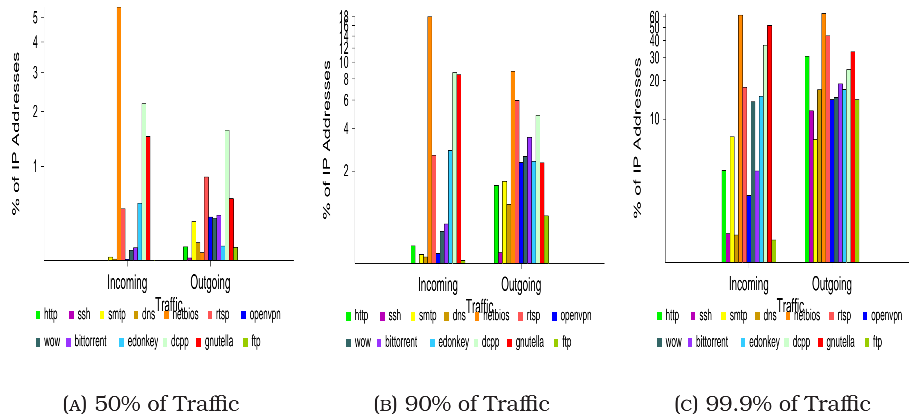
With an exception from the Netbios Incoming traffic; where about 5% of the IPs are responsible for the 50% of the traffic; for all other protocols we have portions smaller than 2% of the IP addresses (Fig. 4.8(a)). For the most used protocols (HTTP, FTP, BitTorrent and eDonkey) the 50% of the traffic is produced by about 0.1% of the IP address, with an exception for BitTorrent where 0.3% of the IPs are responsible for this amount of traffic.

For all the protocols the 99.9% of the traffic is always produced by less than 60% of the IP addresses (Fig. 4.8(c)). Particularly, for BitTorrent, eDonkey and FTP this amount of outgoing traffic is produces by about 14 – 18%, while the corresponding percentage for HTTP traffic is 30%. On the other hand incoming traffic is produced by about 3.5% of the IP addresses for HTTP and BitTorrent, while for FTP just the 0.45% is responsible this traffic.

The results from this section show the existence of “elephant” IP addresses corresponding to network traffic usage in all the protocols we are able to classify with *appmon*. The vast majority of the IP addresses has a very small participation in the applications’ network traffic. This might be due to limited usage of the application, inconsistent application caches (that is a host that previously participated in the application network has now a different IP address and other host prompt for it), or attack traffic.

4.4.1 Incoming Versus Outgoing Traffic

An interesting question is whether the IP addresses that contribute the most traffic are symmetric. That is, if they have the same contribution both for incoming and outgoing traffic. Figure 4.9(a) plots the relationship

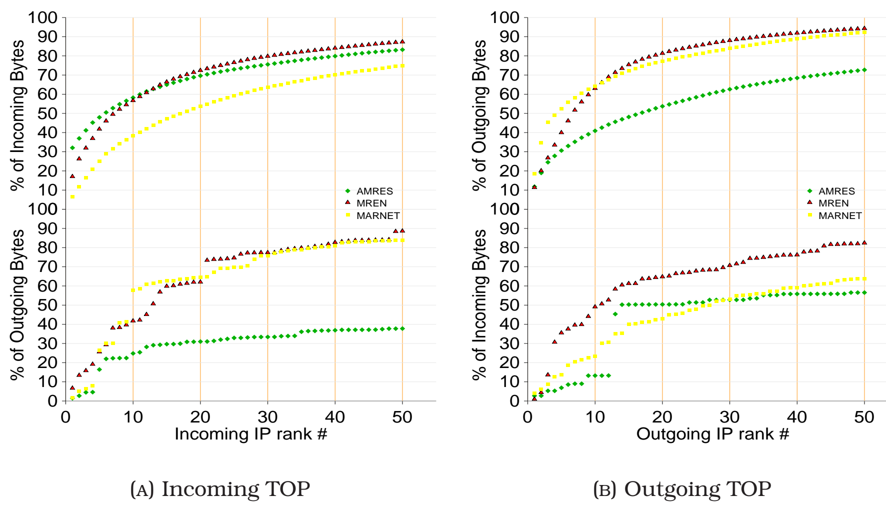


(A) 50% of Traffic (B) 90% of Traffic (C) 99.9% of Traffic

FIGURE 4.8: Per-Application Traffic Vs. Active IP addresses. Figure shows the percentage of IP addresses that contributes to the respective portion of traffic. The left group of bars shows results for Incoming traffic and the right one those from Outgoing traffic

between incoming and outgoing percentage for the top 50 incoming traffic IP addresses, for each of the monitored networks. Note that the plot is cumulative, that is the percentage transferred from each IP address is the value shown by the corresponding symbol minus the value shown by the preceding symbol. As we can see the portion of traffic transfered by each IP address is not analogous to the traffic received by the same IP address. If we look at the rank that each of the incoming traffic top IPs has in the outgoing traffic distribution we see that only 14 out of 50 IP address are in the TOP 50 of the outgoing traffic rank in the case of the AMRES network (and only 6 in the top 10). In the case of MREN and MARNET the results are different since 38 out of 50 IP addresses are in both incoming and outgoing high ranks, in the case of MREN and 32 out of 50 for MARNET network.

Figure 4.9(b) plot the relationship between outgoing and incoming traffic percentage for the top 50 outgoing traffic IP addresses. The results are similar to the case of the incoming traffic top 50 in-out relationship.



(A) Incoming TOP

(B) Outgoing TOP

FIGURE 4.9: Incoming and Outgoing traffic percentage for the top 50 Incoming traffic IP addresses. Symbols are cumulative (actual traffic of an IP is the difference of the corresponding symbol from the preceding one). Figures shows that only a few IP addresses have both large incoming and outgoing traffic percentage, while most upload a dis-analogous number of bytes as it compares with the bytes they download.

4.5 Summary

In this chapter we presented a large scale analysis of network traffic data for a period of two months from three academic institutions located in three different countries. The observations from this analysis show HTTP, BitTorrent and eDonkey to be the three protocols contributing to the most to the traffic rates observed. HTTP is still the most used protocol since it has the larger number of active IP address. On the other hand, although not so popular (in terms of active IP addresses) BitTorrent and eDonkey produce a larger portion of traffic.

We identified the existence of “elephant” IP addresses in all the protocols. A small portion of IP addresses is responsible for most of the traffic while the vast majority of IPs is limited to only a small portion of the total traffic exchanged.

Chapter 5

Related Work

Although the research community has given great interest in traffic classification the late years, there does not exist a free application that accurately classifies network traffic. Most of the network administrators use the MRTG [31] application that monitors SNMP network devices and shows how much traffic has passed through each interface. MRTG is a useful tool for capturing traffic outbreaks but does not give any information about the application that generated the traffic. FlowScan [32] analyzes and reports on Internet Protocol flow data exported by routers. Flowscan provides classification only for well-known services by using port-based classification and such cannot accurately categorize traffic belonging to protocols that use dynamic port numbers.

Several researchers have conducted significant work towards alternative ways for network traffic classification. Due to the popularity and high bandwidth demands of peer-to-peer file sharing applications, a significant body of work has focused on the identification and categorization of peer-to-peer application traffic. Authors in [29] and [24] identified the inaccuracy of port-based classification, by comparing it with a payload based classifiers. Both papers use packet payload signatures for classifying traffic. Approaches presented in [23, 34] use deep packet inspection and application signatures for attributing traffic flows to the corresponding applications. Some of the signatures presented in these papers where also used by our tool, others where refined and new ones where created.

Recent approaches try to identify the applications that generate the traffic by not looking at the packet payload, but only at the transport layer [17,25] or the statistical characteristics of the network flow, like packet sizes and Round Trip Times [13, 18, 19, 41]. A different approach presented by Karagianis et. al. at [26] tries to classify traffic by characterizing the behavior of the host generating this traffic. Though promising, work has still to be done for these approaches to be able to classify and present the traffic distribution in real time.

Traffic analysis has always been of interest for the networking research community. In the analysis provided by [20], the authors also find HTTP to be the dominant client-server application. Sen in [35] and Gerber et. al. in [21] also show that the majority of traffic is generated by small percentages of the total IP addresses. The most popular P2P protocol vary according to the region and time of the study.

Chapter 6

Conclusions and Future Work

This thesis presents *appmon*, an application for real time per-application network traffic categorization. The main goal of the application is to visualize the network traffic usage in order to help in effectively monitoring the network traffic usage. *Appmon* uses a large set of protocol trackers for the classification of traffic from many emerging applications, while its module design allows for the easy addition of more protocol trackers in the future.

Through our experimental evaluation we show that *appmon* is able to categorize traffic in speeds that reach 1 Gbit/s, using specialized network hardware and up to 500 Mb/s using commodity network interfaces. Also when deployed in a real network monitoring environment, *appmon* behaves steadily and presents no problems even in high traffic speeds. Though we expect to have a very small percentage of false positives on real network monitoring circumstances, we show that using application specific traces no false positives wheres reported for our classification structures.

Appmon overcomes port-based traffic classification methods, and through its structure is able to identify applications that masquerade their traffic through well-known port numbers; i.e. BitTorrent traffic over HTTPs' port 80.

Furthermore we present a large scale analysis of network traffic data for a period of two months from three academic institutions located in three different countries. The observations from this analysis show HTTP, BitTorrent and eDonkey to be the three protocols contributing to the most to

the traffic rates observed. HTTP is still the most used protocol since it has the larger number of active IP address. On the other hand, although not so popular (in terms of active IP addresses) BitTorrent and eDonkey produce a larger portion of traffic.

We also identify the existence of “elephant” IP addresses in all the protocols. A small portion of IP addresses is responsible for most of the traffic while the vast majority of IPs is limited to only a small portion of the total traffic exchanged.

An initial goal of our study was to classify as much traffic as possible and build classification functions for a large number of application protocols. This proved to be a difficult task, since in real world exist applications we where not aware off. An example of these applications were Bluesky and PPSstream, two P2P streaming applications used mainly in China. We became aware of these applications only when we installed a monitoring sensor in Singapore. The process, followed by *appmon*, for creating an application tracker function is manual. Since we identify unknown traffic in one of our sensors we capture the traffic and try to isolate the flows responsible for this traffic; by excluding all known traffic. Then we look at the packets of each of those flows, and try to extract a payload signature that will identify the traffic. If succeeded, the signature will be tested for false positives using known protocol traces and eventually added to *appmon*. As described the process of extracting new signatures is 100% manual and time consuming. To minimize the time needed for this process we plan to look at automated or semi-automated signature generation algorithms that would inform the sensors administrator (and if the organizations policy accepts it, us), for the existence of a new application protocol that needs its' attention in order to be added to the classified protocols.

One of the limitations of our application is the inability to classify encrypted traffic. With several “bandwidth-hungry” applications increasingly trying to make their traffic difficult to detect, the use of encrypted traffic is expected to overcome plain text traffic. In order to address this problem, we plan to explore whether non payload traffic classification methods can be used to identify and classify network traffic. Since the payload is encrypted, signature-based classification methods can not be used. Having

this in mind we plan to explore flow inspection methods for classifying the traffic. By flow inspection we mean methods that try to extract common properties among flows of the same application protocol. This properties may vary from packets sizes, round trip times (RTT), to traffic rates experienced by the flows. Work in this field, not limited in encrypted traffic classification, has be done by [14, 30], but none of these approaches is able to classify and report traffic in real time; functionality that would enable network administrators to take action by prioritizing the flows.

Another critical issue, that needs to be addresses in the future, is the use of tunneling. That is multiplexing several applications over a single NetFlow. For example a user, in order to traverse a traffic shaping policy, may encapsulate traffic shaped protocols, such as BitTorrent and eDonkey, inside a commonly allowed protocol such as SSH. In this case to classify traffic correctly, we need to distinguish the different protocol encapsulated in the SSH traffic. This is a very difficult task but worth been explored.

Appendix A

Appmon Installation and Configuration Instructions

Appmon is a tool build over the Packet Capture Library (libpcap). It monitors the traffic of a network and tries to identify application specific patterns inside network packets. In this way it manages to categorize traffic into the application it belongs to.

This is a preliminary version of appmon so it does not come with any configure scripts.

In order to compile you first need to install:

libpcap0.8-dev

librrd2-dev

libncurses5-dev

Appmon can be run in three different modes:

1 With simple text output

```
./appmon -w -d eth0 net
```

2 With ncurses output

```
./appmon -d eth0 net
```

3 Exporting a web interface

```
./appmon -c -d eth0 net
```

In order to configure appmon with a web interface you need to install apache and rrdtool.

To setup appmon files simply run the provided script:

```
./setupdirs <path_to_public\_dir>
eg:
./setupdirs /home/me/public\_html
```

The script will create all the necessary directories and links.

You also need to configure the apache web server to execute the cgi files in *appmons'* specified directory. To do this add the following lines to your http.conf file

```
<Directory /home/*/public\_html/appmon/cgi-bin>
    Options +ExecCGI
    SetHandler cgi-script
</Directory>
```

and uncomment the following line in order to activate the cgi handler

```
AddHandler cgi-script .cgi .sh .pl
```

A.1 Database Installation and Configuration

To use the database logging functionality offered by *appmon* you first need to install *postgres* SQL database.

postgresql-8.1

The communication with the database is done using some simple *perl* scripts. For them to function properly the following packages are needed:

libpg-perl

postgresql-plperl-8.1

install libdbd-pg-perl

The first step, after installing the needed packages, is to create the appmon database user, who will add and retrieve data from the database.

Create the database user:

```
sudo su postgres -c 'createuser appmon'
```

You will be prompt with the next question:

```
Shall the new role be a superuser? (y/n)
```

Answer YES(y) since the appmon db-user must be a superuser in order to copy files into the database.

Add some configuration for the user, so he can connect to the database

```
edit /etc/postgresql/8.1/main/pg_hba.conf
```

and add the following line

```
local all appmon md5
```

This will allow the db-user appmon to access the database locally.

Then we need to create the database for appmon and create the needed tables to it. The tables are created using the script *create_appmondb.sql* provided with *appmon*. You need to run the following.

```
createdb -O appmon -U appmon appmon
psql -U appmon -f create_appmondb.sql
```

Now that the database and the user for appmon are created the final step is to set a cron job that will collect the data exported by *appmon* into the database. Add the following line to your cron configuration file (*/etc/crontab*):

```
*/5 * * * * root  /path/to/appmon/appmon-sa/php/collect_data.pl
```

The script will be run every 5 minutes.

Now you are ready to run *appmon* with the *-b* and *-c* command line options in order to start exporting data for the database (*-b*) and create the necessary web pages for viewing the results (*-c*).

Appendix B

Appmon Manual Page

NAME

appmon - Per-Application Network Traffic Classification Tool

SYNOPSIS

appmon [-hvVwcstb] [-d device] [-r file] [-i num] [-l file] [-n string] [-u num] [expression]

DESCRIPTION

Appmon passively monitors traffic passing trough a monitored link and categorizes active network flows (identified by the 5-tuple) according to the application that generated them. A network flow is defined as a set of IP packets with the same protocol, source and destination IP address, and source and destination port (also known as a 5-tuple). Traffic categorization is performed using information from both the packet header and payload.

OPTIONS

-h display a short help text.

-d use the given network device instead of eth0.

-r file

read from tcpdump formated file instead of live traffic.

-i num

time in seconds for exporting the results (default: 10 sec).

-v verbose output.

-n string

Give the monitor name to be displayed in the web-page.

-w Use simple batch format instead of ncurses.

-u num

time in seconds to reset the top bandwidth consuming IPs structs (default: 100 sec). This is done to avoid classifying as TOP an IP address that transfered a large amount of bytes in the past but is not that active in the present.

-c Create and update web-pages instead of using ncurses output.

-t Do not collect data for the TOP IP addresses.

-V Verbose output for the RRD Traffic distribution graph.

-s Print total traffic statistics when program ends.

-b Output files for logging data in a database.

EXAMPLES

Run with simple text output separating incoming/outgoing traffic to/from 192.0.0.1

```
./appmon -w -d eth0 192.0.0.1
```

Run with ncurses output, reading from tcpdump trace tracefile.pcap separating incoming/outgoing traffic to/from 10.0/16 subnet

```
./appmon -r tracefile.pcap 10.0/16
```

Run with web interface for network device /dev/dag0

```
./appmon -c -d /dev/dag0
```

AUTHOR

Demetres Antoniades (danton@ics.forth.gr)

SEE ALSO

pcap(3) bpf(4) tcpdump(8)

Appendix C

Monitoring API Tracker Library

The Tracker MAPI function library (`trackflib`) provides a set of predefined functions for per-application traffic monitoring, even for hard-to-track applications that use dynamically negotiated ports or use existing protocols and port numbers (e.g., non-web traffic using HTTP through port 80) to masquerade their traffic.

Each tracker function scans the network traffic for packets that belong to some specific application-level protocols. If such a packet is found, then the traffic of the flow in which the packet belongs to is attributed to the identified application. All tracker functions operate as network filters, for example in a similar way to the `BPF_FILTER` function. Thus, applying a tracker function to a network flow will result to capturing only the traffic of the particular application. For example, a network flow on which the `TRACK_GNUTELLA` function has been applied, will receive only traffic that belongs to Gnutella P2P file sharing applications.

A detailed description and evaluation of the Tracker MAPI function library (`trackflib`) is presented in [38].

TRACK_FTP The `TRACK_FTP` function, when applied to a network flow, filters the network traffic by keeping only packets that belong to the FTP protocol. The function supports both active and passive FTP modes.

TRACK_DC The `TRACK_DC` function, when applied to a network flow, filters the network traffic by keeping only packets that belong to the DC++ file sharing application/protocol.

TRACK_GNUTELLA The `TRACK_GNUTELLA` function, when applied to a network flow, filters the network traffic by keeping only packets that belong to the Gnutella file sharing application/protocol.

TRACK_EDONKEY The `TRACK_EDONKEY` function, when applied to a network flow, filters the network traffic by keeping only packets that belong to the eDonkey file sharing application/protocol.

TRACK_TORRENT The `TRACK_TORRENT` function, when applied to a network flow, filters the network traffic by keeping only packets that belong to the BitTorrent file sharing protocol.

TRACK_GRID_FTP The `TRACK_GRID_FTP` function, when applied to a network flow, filters the network traffic by keeping only packets that belong to the File Transfer Protocol used by GRID systems.

Bibliography

[1] Direct Connect. http://dcplusplus.sourceforge.net/.

[2] eMule. http:emule-project.net.

[3] Gnutella. http://www.gnutella.com.

[4] Internet Assigned Numbers Authority. http://iana.org.

[5] nttcp. http://sd.wareonearth.com/~phil/net/ttcp/.

[6] PPStream. http://en.wikipedia.org/wiki/PPStream.

[7] Protocol Information Wiki. http://protocolinfo.org/.

[8] rrdtool. http://oss.oetiker.ch/rrdtool/.

[9] South - Eastern European Reasearch and Education Network. http://www.seeren.org.

[10] The BitTorrent Protocol. http://www.bittorent.org.

[11] The LOBSTER IST project. http://www.ist-lobster.org.

[12] The Ncurses Library. http://www.gnu.org/software/ncurses/ncurses.html.

[13] L. Bernaille, I. Akodkenou, A. Soule, and K. Salamatian. Traffic Classification on the Fly. *ACM SIGCOMM Computer Communication Review*, 36(2):23–26, 2006.

[14] L. Bernaille, R. Teixeira, and K. Salamatian. Early Application Identification. *The 2nd ADETTI/ISCTE CoNEXT Conference, Lisboa, Portugal, December*, 2006.

[15] R. S. Boyer and J. S. Moore. A fast string searching algorithm. *Commun. ACM*, 20(10):762–772, 1977.

[16] B. Cohen. BitTorrent protocol specification. *First Workshop on Economics of Peer-to-Peer Systems (P2P"03)*.

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

[18] M. Crotti and F. Gringoli. Traffic classification through simple statistical fingerprinting. *ACM SIGCOMM Computer Communication Review*, 37(1):5–16, 2007.

[19] M. Crotti, F. Gringoli, P. Pelosato, and L. Salgarelli. A statistical approach to IP-level classification of network traffic. *Communications, 2006. ICC'06. IEEE International Conference on*, 1, 2006.

[20] T. Dang, M. Perenyi, A. Gefferth, and S. Molnar. On the Identification and Analysis of P2P Traffic Aggregation? *LECTURE NOTES IN COMPUTER SCIENCE*, 3976:606, 2006.

[21] A. Gerber, J. Houle, H. Nguyen, M. Roughan, and S. Sen. P2P The Gorilla in the Cable. *National Cable & Telecommunications Association (NCTA) 2003 National Show*, 2003.

[22] G. Kan. Gnutella. *Peer-to-Peer: Harnessing the Benefits of a Disruptive Technology*, 2001.

[23] T. Karagiannis, 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.

[24] T. Karagiannis, A. Broido, N. Brownlee, K. Claffy, and M. Faloutsos. Is P2P dying or just hiding. *Proceedings of IEEE Globecom*, 2004.

[25] T. Karagiannis, A. Broido, M. Faloutsos, and K. claffy. Transport layer identification of p2p traffic. In *IMC '04: Proceedings of the 4th ACM SIGCOMM conference on Internet measurement*, pages 121–134, New York, NY, USA, 2004. ACM Press.

[26] T. Karagiannis, K. Papagiannaki, and M. Faloutsos. BLINC: multilevel traffic classification in the dark. *ACM SIGCOMM Computer Communication Review*, 35(4):229–240, 2005.

[27] S. McCanne and V. Jacobson. The BSD packet filter: A new architecture for user-level packet capture. *Proc. Winter''93 USENIX Conference*, 1993.

[28] S. McCanne, C. Leres, and V. Jacobson. libpcap. Lawrence Berkeley Laboratory, Berkeley, CA. (software available from http://www.tcpdump.org/).

[29] A. Moore and K. Papagiannaki. Toward the Accurate Identification of Network Applications. *Passive And Active Network Measurement: 6th International Workshop, PAM 2005, Boston, MA, USA, March 31-April 1, 2005: Proceedings*, 2005.

[30] A. Moore and D. Zuev. Internet traffic classification using bayesian analysis techniques. *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, pages 50–60, 2005.

[31] T. Oetiker. Mrtg: The multi router traffic grapher. In *LISA '98: Proceedings of the 12th Conference on Systems Administration*, pages 141–148, Berkeley, CA, USA, 1998. USENIX Association.

[32] D. Plonka. Flowscan: A network traffic flow reporting and visualization tool. In *LISA '00: Proceedings of the 14th USENIX conference on System administration*, pages 305–318, Berkeley, CA, USA, 2000. USENIX Association.

[33] A. Scampi. Scaleable Monitoring Platform for the Internet. *URL:< http://www. ist-scampi. org*, 4.

[34] S. Sen, 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*, pages 512–521, 2004.

[35] S. Sen and J. Wang. Analyzing Peer-To-Peer Traffic Across Large Networks. *IEEE/ACM TRANSACTIONS ON NETWORKING*, 12(2):219, 2004.

[36] S. Skype Technologies. Skype, from http://www. skype. com, 2005.

[37] The LOBSTER Consortium. MAPI Public Release. `http://mapi.uninett.no/download/mapi-2.0-beta1.tar.gz`.

[38] The LOBSTER Consortium. Deliverable D3.2: Deployment - first phase, Nov. 2006. `http://www.ist-lobster.org/publication/deliverables/D3.3.pdf`.

[39] P. Trimintzios, M. Polychronakis, A. Papadogiannakis, M. Foukarakis, E. P. Markatos, and A. Øslebø. Dimapi: An application programing interface for distributed network monitoring. In *Proceedings of the 10th IEEE/IFIP Network Operations and Management Symposium (NOMS)*, Apr. 2006.

[40] J. Xu, J. Fan, M. Ammar, and S. Moon. Prefixpreserving ip address anonymization: Measurement-based security evaluation and a new cryptography-based scheme. *In Proceedings of the 10th IEEE International Conference on Network Protocols, November*, 2002.

[41] D. Zuev and A. Moore. Traffic Classification Using a Statistical Approach. *Passive And Active Network Measurement: 6th International Workshop, PAM 2005, Boston, MA, USA, March 31-April 1, 2005: Proceedings*, 2005.