UNIVERSITY OF OSLO
Department of Informatics

Traffic classification with passive measurement

Master thesis

Phong H. Pham

23th May 2005

Traffic classification with passive measurement

Phong H. Pham
Oslo University College
Uninett Research

Abstract

This is a master thesis from a collaboration between Oslo University College and Uninett Research. Uninett have a passive monitoring device on a 2.5 Gbps backbone link between Trondheim and Narvik. They uses measurement with optical splitters and specialized measuring interfaces to trace traffic with Gigabit speed. We would like to investigate the structure and patterns in these data. It is of special interest to classify the traffic belonging to different services and protocols.

Traffic classification enables a variety of other applications and topics, including Quality of Service, security, monitoring, and intrusion-detection that are of use to research, accountants, network operators and end users. The ability to accurately identify the network traffic associated with different applications is therefore important. However, traditional traffic to higher-level application classification techniques such as port-based is highly inaccurate for some applications.

In this thesis, we provide an efficient approach for identifying different applications through our classification methodology. Our results indicate that with our technique we achieves less than 6.5% unknown type in most cases compared to the port-based which is 46.6%.

The project is divided into three phases. First we will have a look at the problems dealing with collecting data traces in high speed network system. Second we will explore how we can identify and classify the data into different categories. Finally we will try to analyse our results offline.

Index terms – Passive network measurement, Cluster, Classification

Acknowledgments

First of all I want to express my gratitude to my internal supervisor Associate Professor Tore Jonassen, not only for introduction me to the area of cluster and how to use it to analyze network traffic, but also for all encouragement during this work. It has been highly educational working with you.

I also thank Arne Oslebo and Jon Kåre Hellan at Uninett Research for helping me with technical problems concerning with tracing large data at their backbone link and especially questions around the optical network interface card, the DAG card.

Furthermore, I thank Professor Mark Burgess for all Wednesday meetings that he has arranged for master students. It has been very inspiring to participate in these meetings.

In addition, I wish to express my appreciation for all the support and encouragement provided by my colleagues at the class of Network and System administration. I am also grateful to all my friends for making my life contain more than work.

Finally, very special thanks to my dear Hang, for all the support and patience during the work. Without you this thesis would never been finished.

Oslo, 20th May 2005

Phong H. Pham

Table of contents

1 Introduction

The focus of this thesis is to describe a model of classification on high-speed network which applies results from the statistical properties of plots to network monitoring systems.

Over the last few years, traffic on the Internet has increased tremendously, both in terms of amount of traffic, and in variety of applications. The introduction of voice, video and other real-time applications has changed the way the Internet is used. This has triggered the need for a change in traffic handling on the Internet. In particular, there is increasing demand for service differentiation. The Diffserv architecture [27] of the IETF is one such step towards fulfilling this demand. However, for any such service, the very basic problem one encounters is that of classification of services. Well-known port numbers can no longer be used to reliably identify network applications. There is a variety of new Internet applications that either do not use well-known port numbers or use other protocols, such as HTTP, as wrappers in order to go through firewalls without being blocked. In addition, emerging services avoid the use of well known ports altogether probably to avoid detection, e.g. some peer-to-peer, (P2P), applications. One consequence of this is that a simple inspection of the port numbers used by flows may lead to the inaccurate classification of network traffic.

In this project, we look at these inaccuracies in detail. Using a full payload packet trace collected from a Uninett Gigabit backbone link we attempt to identify the types of errors that may result from port-based classification and quantify them for the specific trace under study. To address this question we devise a classification methodology that relies on the full packet payload. We describe the building blocks of this methodology and elaborate on the complications that arise in that context.

We also show that our approach only requires the examination of the very first few packets to identify a P2P or passive FTP connections. Our technique can significantly improve the P2P and passive FTP traffics volume estimates over what pure network port based approaches provide.

These questions can be of interest both in the short term perspective and in a long term perspective. In the short term perspective this kind of information is used in traffic management and control, for example as user information systems. In long term perspective this information is the basis of traffic planning and design. It is essential to system design, capacity analysis as well as impact analysis.

In this thesis we will try two different approaches to the problem mentioned above. The first one is to try to classify the number of packets that flow between the backbone link for a large number of time intervals using port-based technique. While the second approach is to classify traffic using our content-based methodology.

There is a relative large body of research related to our problem. This research is reviewed in chapter 2. The following chapter, chapter 3, deals with the experimental

plans, how the infrastructure is setup and detailed descriptions of different equipments that were used. Chapter 4 describes the method to identify and classify the data into several categories against different criteria.
We also tried to analyse the result using both descriptive and inferential statistics. This is described in chapter 5, while the errors of the results are described in sub-chapter 5.3.4. The last chapter of the thesis summarized the conclusions drawn from the work.

2 Review of related research

2.1 Background on Uninett

The Uninett Group supplies advanced Internet to research and education and have responsibility for the development, operation, co-ordination and standardisation of network solutions. Uninett is owned by the Norwegian Ministry of Education and Research and consists of a parent company and three subsidiaries. The Group is located in Trondheim.

2.2 Measurement approaches

Most backbone Internet circuits currently operate at speeds ranging from 1 Gb/s to 10 Gb/s [21]. In order to verify operational, performance and security characteristics of the network and to enable problem resolution we need to measure a high-speed network monitoring system [19]. We need to measure elementary network performance characteristics, such as throughput, delay, packet loss rate and jitter. And we also need to search for traffic patterns indicating possible security problems, such as intrusion or denial of service attacks.

Measurement data can be collected in two principal ways:

- actively by injecting testing packets into the network and processes them as they are received in another part of the network [22] or
- passively by observing existing traffic in the network [23].

The monitoring can be performed by standalone units or be router-based.

Both types of monitoring have their advantages and difficulties. Active monitoring is the easiest way to measure one-way delay, but it is generally unsuitable for other network characteristics, as it measures characteristics experienced by testing packets, rather than by existing traffic. Therefore, passive monitoring, which does not influence existing traffic, has become a popular method of precise and reliable network monitoring.

However, passive network monitoring is becoming increasingly demanding on computing resources. The reason is that the physical network speed tends to increase faster than the computer processor speed. We already cannot monitor current high-speed network links just by tapping traffic with a regular network adaptor, catching all packets with tcpdump and processing them even on the most powerful PCs [20].

An approach to this problem is SCAMPI, a two-and-a-half years European project to develop a scalable monitoring platform for the Internet [21]. It has two main goals:

- To enable easy writing of portable monitoring applications
- To enable detailed monitoring of high-speed Internet

The first goal is realised by providing MAPI – Monitoring API, which enables application developers to start at higher level of abstraction of flows and monitoring functions.

The second goal is realised by performing certain time-critical functions needed for most monitoring tasks inside the SCAMPI adapter, a specialised programmable monitoring adapter. The data rate going further to the host computer is thus significantly reduced.

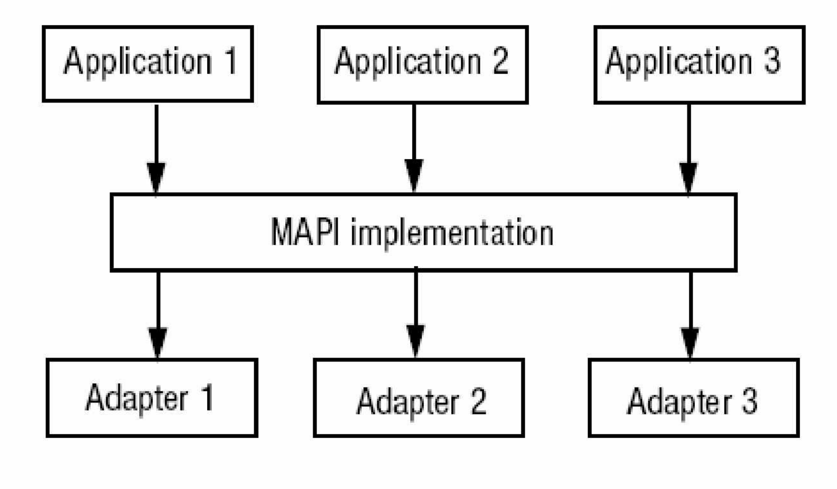


Figure 1: SCAMPI architecture

SCAMPI architecture is illustrated in Fig. 1. Several applications run concurrently on top of MAPI, which in turn runs on top of various network adapters. Currently, they support the SCAMPI adapter, DAG adapter and regular Ethernet NIC cards. Applications are portable between computers equipped with any of these adapters. When certain adapter provides some monitoring function in its hardware or firmware, MAPI will automatically use it. If it is not provided by the adapter, MAPI will use its own software implementation of the particular function.

Another monitoring adapter is the DAG card from Endace. When compared to DAG, the SCAMPI adapter will provide more functionality, it will be an open system allowing users to download their own firmware into the adapter and it is expected to be significantly less expensive.

2.3 IP traffic classification

One approach commonly used for identifying applications on an IP network is to associate the observed traffic (using flow level data, or a packet sniffer) with an application based on TCP or UDP port numbers.

The TCP/UDP port numbers are divided into three ranges: the well known ports (0-1023), the registered ports (1024 - 49151), and the dynamic and/or private ports (49152 - 65535). A typical TCP connection starts with a SYN/SYN–ACK/ACK handshake from a client to a server. The client addresses its initial SYN packet to the well known server port of a particular application. The source port number of the packet is typically chosen dynamically by the client. UDP uses ports similarly to TCP, though without connection semantics. All future packets in either a TCP or UDP session use the same pair of ports to identify the client and server side of the session. Therefore, in principle the TCP or UDP server port number can be used to identify the higher layer application, by simply identifying which port is the server port and mapping this port to an application using the IANA (Internet Assigned Numbers Authority) list of registered ports [4]. However, port-based application classification has limitations.

While networks changed significantly in terms of bandwidth available and type of traffic, network-monitoring applications basically remains the same. Besides large companies that can afford to buy expensive network traffic monitoring applications, most people still uses MRTG [24] polling traffic information out of network routers and switches interfaces via SNMP MIB-II variables [25].

Unfortunately today this way of monitoring networks is no longer effective because:

- The traffic has changed significantly from what it used to be a few years ago both in terms of protocols (HTTP is likely not to be the most used protocol anymore) being used and user (many end-user computers move more data than servers) behaviour.

- It is no longer possible to predict what is flowing across the network using aggregate information such as the one provided by the network interface counters.

- Security violation attempts are quite common and cannot be detect without using specialised tools.

- Well-known ports cannot be used anymore to identify a service (e.g. passive FTP and P2P use dynamic ports) making it difficult to calculate simple statistics such as how much FTP traffic is flowing across the local network.

- Many implementations of TCP use client ports in the registered port range. This might mistakenly classify the connection as belonging to the application associated with this port. Similarly, some applications (e.g. old `bind` versions), use port numbers from the well-known ports to identify the client site of a session.

- An application may use ports other than its well-known ports to circumvent operating system access control restrictions, e.g., non-privileged users often run WWW servers on ports other than port 80, which is restricted to privileged users on most operating systems.

- There are some ambiguities in the port registrations, e.g. port 888 is used for CDDBP (CD Database Protocol) and accessbuilder.

- The use of traffic control techniques like firewalls to block unauthorized, and/or unknown applications from using a network has spawned many work-around which make port based application authentication harder. For example port 80 is being used by a variety of non-web applications to circumvent firewalls, which do not filter port-80 traffic. In fact available implementations of IP over HTTP allow the tunneling of all applications through TCP port 80.

- Trojans and other security (e.g. DoS) attacks generate a large volume of bogus traffic which should not be associated with the applications of the port numbers those attacks use.

2.4 Related work

Due to its fundamental nature and its underpinning of many other techniques, the field of traffic classification has maintained continuous interest.
For example, still the most common technique for the identification of network applications through traffic monitoring relies on the use of well known ports: an analysis of the headers of packets is used to identify traffic associated with a particular port and thus of a particular application [28, 29]. It is well known that such a process is likely to lead to inaccurate estimates of the amount of traffic carried by different applications given specific protocols. Our work is presented in the light of these traditional classification techniques diminishing in effectiveness.

A recent work [30] uses application signatures to characterize the workload of P2P downloads. But they do not provide any evaluation of accuracy, scalability or robustness features of their signature.

Other authors that have noted the relationship between the class of traffic and its observed statistical properties include Paxson [32] who reports on the distribution of flow-bytes and flow-packets for a number of specific applications.

A previous related work has examined the variation of flow characteristics according to application. Claffy [5] investigated the joint distribution of flow duration and number of packets, and its variation with flow parameters such as inter-packet timeout. Differences were observed between the support of the distributions of some application protocols, although overlap was clearly present between some applications. Most notably, the support of the distribution of DNS transactions had almost no overlap with that of other applications considered. The use of such distributions as a discriminator between different application types was not considered.

There exists a wealth of other research on characterizing and modeling workloads for particular applications, e.g., [6, 7, 8, 9, 10, 11]. An early work in this space, [12], examines the distributions of flow bytes and packets for a number of different applications. Interflow and intraflow statistics are another possible dimension along which application types may be distinguished. [13] observed that simple (Poisson) models are unable to effectively capture some network characteristics. However, they did find a Poisson process could describe a number of events caused directly by the user; such as telnet packets within flows and connection arrivals for ftp-data.

All these studies assume that one can identify the application traffic unambiguously and then obtain statistics for that application. In contrast, we are considering the dual problem of inferring the application from the traffic statistics. This type of approach has been suggested in very limited contexts such as identifying chat traffic [14]. Analysis of Internet chat systems to make an effective use of the packet-size profile of particular applications. The authors note that packets relevant to their studies tend towards a specific size-profile, limiting themselves to this profile allowed for a more precise selection of traffic relevant to their study.

Signature-based detection techniques have also been explored in the context of network security, attack and anomaly detection, e.g. [15, 16, 17, 18] where one typically seeks to find a signature for an attack. However, we apply our classification techniques to identify everyday traffic. There is also a large body of literature on extracting information from packet traces (e.g. [31]) which provides and evaluates signatures at application layer.

3. Experimental plans and infrastructure setup

This chapter provides a detailed description of experimental plans, infrastructure setup and equipments that have been performed.

3.1 Passive monitoring process

Our traffic capture system is based on passive traffic measurement of Gigabit Ethernet link, using optical splitters. A whole copy of the traffic is collected by a PC equipped with a DAG 4.2GE card [2]. The passive monitoring process consists of three elements:

- A monitoring process which collects the packet traces.
- A data repository process that stores the traces once they have been collected.
- An analysis process which performs offline analysis.

Monitoring process
The monitoring process is responsible for collecting the packet traces. Each trace is a sequence of packets records at the link, together with timestamp indicating the time at which the packets were observed.

The monitoring process is handled by a PC, scampi1 at Uninett, and an optical network interface card, known as the DAG card. On scampi1 there is only one DAG card, at /dev/dag0, with two interfaces. The card is connected to the Trondheim-Narvik fiber and we can capture both directions at the same time. Existing DAG cards are capable of monitoring links ranging in speed from 155 Mbps to 10Gbps. The DAG card captures, timestamps, and transfers the IP packet to the main memory of the PC and then transfers the data to the disk space.

The optical splitter is installed on the monitored link, and one output of the splitter is connected to the DAG card in the PC. This is a receive-only connection, i.e. the DAG card does not have the capability of injecting data into the network. Since a receive-only passive optical splitter is used, failure or misbehavior of the monitoring entity or the DAG card cannot compromise network integrity. The amount of disk space bounds us to only capture some few hours of data trace at full link utilization. We can either schedule trace collection for a predefined interval or allow it to run until space on the hard disks is exhausted. The packet timestamps are generated by an embedded clock on the DAG card that is synchronized to an external GPS signal. GPS is a satellite-based system that provides global time information with an accuracy of 20 nanoseconds. Hardware errors as well as other system related issues bring the maximum error on timestamps to 5 µs.

Data Repository
The data repository is consisting of a disk storage space. It is located at the data center. For short traces, a dedicated optical link is available for transferring the data from the monitoring process back to the data center. For long traces consists of several TB, the

best method is by transferring to another PC, Cia also at Uninett, with larger storage space.
In our thesis we studies packet traces, and even an one hour trace on a 1Gb link we get over 35GB of data, while one day of data will be over 1.5TB. So it is not possible to store data for long periods of time.

Data Analysis Platform

Data analysis is performed offline on a local cluster located at Oslo University College. Two categories of analysis are performed on the platform:

- *Port-based analysis* classifies flows according to their port numbers. This analysis requires access only to the part in the packet header that contains the port numbers.
- *Content-based analysis* examine whether a flow carries a well-known signature or follows well-known protocol semantics.

More details on these analysis on chapter 4.

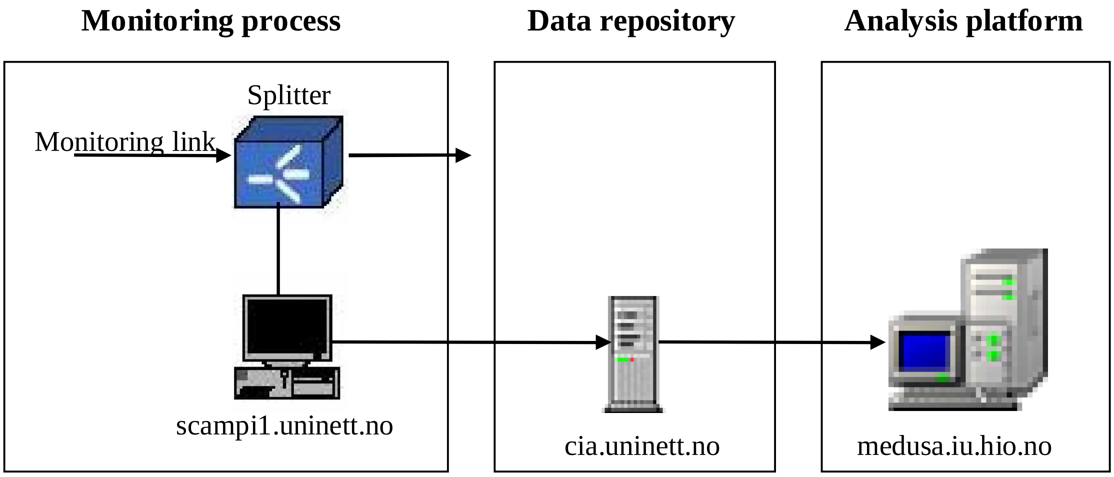


Figure 2: Infrastructure setup

System parameters:

scampi1.uninett.no

CPU: 2 x Intel(R) XEON(TM) CPU 2.20GHz
Card: Dag Device Driver version 2.4.14 Endace Measurement Systems Ltd,
Dag 0: Dag 4.22GE, Dual Gigabit Ethernet
Main memory: 2 GB
Storage space: 0.5 TB

cia.uninett.no
CPU: Intel(R) Xeon(TM) CPU 3.20GHz
Main memory: 1 GB
Storage space: 8.5 TB
Card: Gigabit Ethernet card

medusa.iu.hio.no
master: CPU: Intel(R) Pentium(R) 4 CPU 2.80GHz
Main memory: 2 GB
Storage space: 0.6 TB
Card: 2 x Gigabit Ethernet card

8 x nodes: CPU: Intel(R) Pentium(R) 4 CPU 2.80GHz
Main memory: 2 GB

Packet size
The packet size distribution depends on the number of requests and the file sizes requested. For instance, with web traffic the packet sizes usually vary from 40 bytes (connection setup packets) to 1500 bytes.
The path maximum transmission unit (MTU) is the minimum of the maximum transmission units on the path and Ethernet is usually on at least one segment of most paths. Hence most data packets tend to be at most 1500 bytes long.

3.2 DAG card

3.2.1 Introduction

The DAG cards are designed for network surveillance applications. Available with a wide range of LAN and WAN physical layers, DAG cards are optimized to enable header-only or full packet capture. Unlike commodity NICs (Network Interface Cards) that may drop packets under load, DAG cards are designed to operate smoothly on high speed links [1].

DAG cards are used to collect packet header and payload from ATM or Ethernet networks and are protocol independent. Full packet or cell capture at line rate allows recording of all header information and/or payload with a high precision timestamp. The packet header and payload information can be stored for later in-depth analysis, or used in real-time for a variety of network monitoring applications, such as billing and intrusion detection systems.

An important feature of DAG cards is the ability to move large quantities of data from network to the host computer with low CPU burden. Almost the entire resources of the host computer remain available to the analysis application. This makes applications run faster, enabling more processing per packet or increasing the rate at which packets are processed.

DAG cards accurately preserve timing information from network flows by generating precise timestamps in hardware for each packet as it arrives at the monitoring point. The hardware clock on the DAG may be further synchronized to other DAG cards or to an external time standard such as the Global Positioning System (GPS). This enables QoS applications such as one-way packet or cell delay, and delay variation measurements over WANs or the internet.

Uninett is currently using DAG 4.2GE dual interface 1000baseSX Gigabit Ethernet cards.

3.2.2 Architecture

The major components of the DAG 4.2GE are shown in Figure 3.

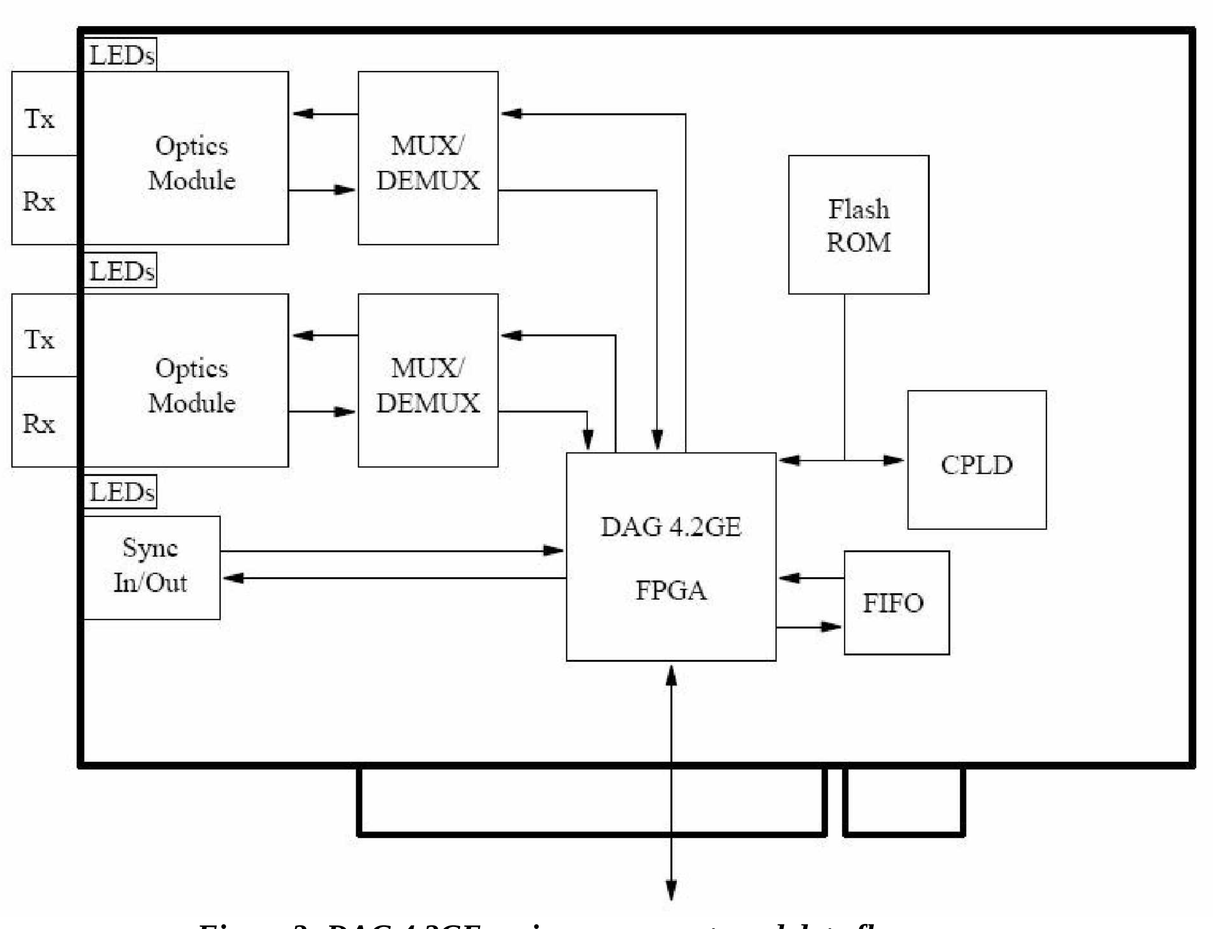


Figure 3: DAG 4.2GE major components and data flows

Serial Ethernet optical data is received by the two 1000baseSX optical interfaces, and passed through demultiplexors. The network data is then fed immediately into the FPGA.

This FPGA contains the timestamp engine, packet record processor, and PCI interface logic. The close association of these components means that packets or cells can be time-stamped very accurately. Time stamped packet records are then stored in an external FIFO before transmission to the host.

The functionality of the DAG 4.2GE can be extended in many ways. A physical transmit path is provided on the DAG 4.2GE so packet generation is possible, but this requires special FPGA images.

A DAG 4.2GE card can be installed in any free 3.3v signaling 64-bit Bus Mastering PCI slot. By default, the driver supports up to four DAG cards in one system, but it is not recommended to have more than 2 cards on a single PCI bus due to bandwidth limitations, as the cards make very heavy use of PCI bus data transfer resources. However, this is not usually a limitation as for most applications a maximum of two cards only can be used with reasonable application performance.

3.2.3 Software utilities

This sub-chapter provides a brief description of some of the important utilities provided in the tools directory of the DAG software package that we used.

dagfour - this program configures the network interface and capture parameters for the DAG 4.2 card, and displays network interface statistics.

dagsnap - this is a utility to capture network data from a DAG card and write it to a file or to stdout for piping into other programs.

dagconvert - a program that can convert DAG’s native ERF format trace files into libpcap format files. It is also capable of capturing from a DAG card directly and writing libpcap format to disk or piping to another program. It can optionally apply software BPF packet filters and can filter on input interface.

For further detailed explanation on the usage of different utilities look at the appendix.

3.3 Beowulf cluster

3.3.1 Introduction

Cluster is a widely-used term meaning independent computers combined into a unified system through software and networking. At the most fundamental level, when two or more computers are used together to solve a problem, it is considered a cluster. Clusters are typically used for *high availability* for greater reliability or *high performance computing*, to provide greater computational power than a single computer can provide.

The cluster at Oslo University College is a Beowulf cluster and consists of a master machine, a frontend, named medusa.iu.hio.no connected to the net, and eight nodes on a private network connected by a 10 Gigabit Ethernet switch.

Beowulf clusters are scalable performance clusters based on commodity hardware, on a private system network, with open source software (Linux) infrastructure. The designer can improve performance proportionally with added machines. The commodity hardware can be any of a number of mass-market, stand-alone compute nodes as simple as two networked computers each running Linux and sharing a file system or as complex as 1024 nodes with a high-speed, low-latency network.

Common uses are traditional technical applications such as simulations, biotechnology, and petro-clusters; financial market modeling, data mining and stream processing, and Internet servers for audio and games.

3.3.2 Pros and cons with Beowulf cluster

The pros are that a “standard” Beowulf setup - is very likely to result in a cluster that can accomplish certain kinds of work much faster than a single computer working alone. The entire network can be put to work in parallel on parts of the problem with tremendous increases in the amount of work accomplished per unit time. A cluster of the same size and computing power as a mainframe is many times cheaper than the mainframe and this is also a big reason why to use a cluster.

Another good thing about Beowulf is that it does not matter if we change the processor type and/or speed and network technology, the programming model is still the same making Beowulf cluster have good forward compatibility.

The cons are that the phrase “certain kinds of work” fails to encompass all sorts of common tasks. Only certain *kinds* of work can be run profitably (that is faster) on a parallel processing supercomputer.

Even worse, as a general rule a task that *can* be run profitably on a parallel supercomputer will generally *not* run any faster on one unless it is specially designed and written to take advantage of the parallel environment. Very little commercial software has yet been written that is designed *a priori* to run in a parallel environment and that which exists is intended for very narrow and specialized applications.

Beowulf-style cluster computing is not really just for computer scientists or physicists. It can provide real and immediate benefits to just about anyone with a need for *computation* (in the sense of lots of compute cycles doing real calculations) as opposed to an *interface*.

3.3.4 Speedup factor

In the following sub-chapter the number of processors will be identified as p. We will use the term “multiprocessors” to include all parallel computer systems that contain more than one processor.

Perhaps one of the most important points of interest when developing solutions on a multiprocessor is the question of how much faster the multiprocessor solves the problem under consideration.
In doing this comparison, one would use the best solution on the single processor, that is, the best sequential algorithm on the single-processor system to compare against the parallel algorithm under investigation on the multiprocessor. The speedup factor, S(p), is a measure of relative performance, which is defined as:

$$\text{S(p)} = \text{ Execution time using one processor / Execution time using a multiprocessor } = \frac{t_s}{t_p}$$

S(p) gives the increase in speed by using multiprocessor.

Several factors will appear as overhead in the parallel version and limit the speedup, notably:

1. Periods when not all the processors can be performing useful work and are simply idle.
2. Extra computations in the parallel version not appearing in the sequential version, as in our experiment, to recomputed constants locally.
3. Communication time between processes.

It is reasonable to expect that some part of a computation cannot be divided into concurrent processes and must be performed sequentially. We assume that during some period, perhaps an initialization period, only one processor is doing useful work, and for the rest of the computation additional processors are operating on processes.

Assuming there will be some part that are only executed on one processor, the ideal situation would be for all the available processors to operate simultaneously for the other times, if the fraction of the computation that cannot be divided into concurrent parts, the time to perform the computation with p processors is given $ft_s + (1-f)t_s / p$. Hence, the maximum speedup factor given by:

$$S(p) = \frac{t_s}{ft_s + (1 - f)t_s / p} = \frac{p}{1 + (p-1)f}$$

This equation is known as Amdahl’s law [26]. Figure 1.3 shows S(p) plotted against number of processors and against f. We see that indeed a speed improvement is indicated. However, the maximum speed up is limited to 1/f.

For example, with only 5% of the computation being serial, the maximum speedup is 20, irrespective of the number of processors.

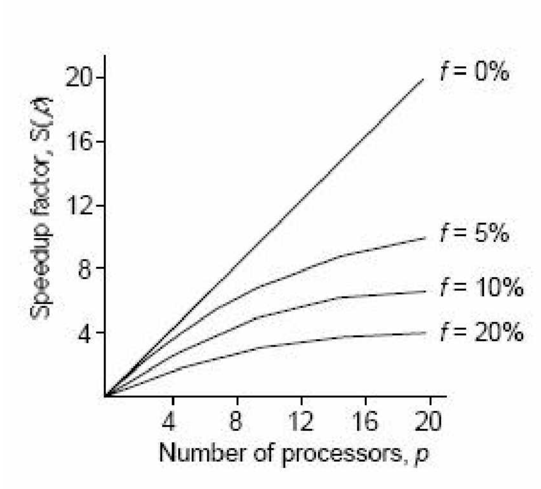
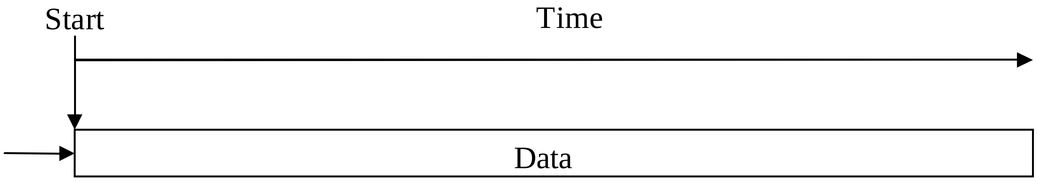


Figure 4: Speedup against number of processors, p

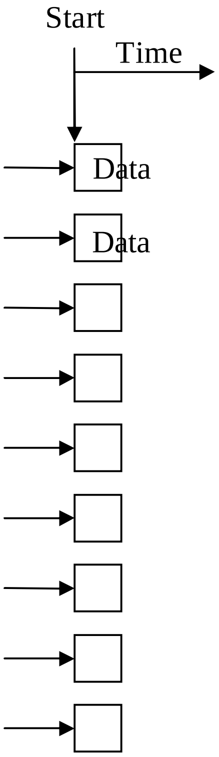
In our classifying problem, we divided a large trace among the processors for each one to perform an independent classifying process. By doing this we can decrease the classifying process time tremendously.

In a sequential implementation, the different traces are attacked one after other, while in parallel implementation, they can be done simultaneously.

a) Classifying trace sequentially



b) Classifying trace in parallel



4 Methods

4.1 Traffic categories

The fundamental classified process in our approach is a traffic-flow which is represented as a flow of one or more packets between a given pair of hosts. The flow is defined by a tuple consisting of the IP address of the pair of hosts, the protocol type (e.g., ICMP, TCP or UDP) and, in the case of UDP and TCP, the port numbers used by the two hosts. In the case of TCP, a flow has a finite duration defined by the semantics of the TCP protocol.

For our work we used TCPdump to classify the different protocols. TCPdump is a tool that allows us to sniff network packets and make some statistical analysis out of those dumps. One major drawback to TCPdump is the size of the flat file containing the text output. But TCPdump allows us to precisely see all the traffic and enables us to create statistical monitoring scripts.

TCPdump runs using BSD Packet Filter (BPF) which is the method of collecting data from the high speed network interface. BPF receives copies from the driver of sent packets and received packets. Before traveling through the kernel all the way up to the user process the user can set a filter so only interesting packets go through the kernel.

The TCPdump's outputs are organized like this:

Timestamp source -> destination: protocol

The timestamp is in format of hours, minutes, seconds and fractional parts of seconds. The source and destination fields are the source and destination host name or IP address. The protocol field for protocol TCP is unique. It contains a flag and a sequence number. When we see these distinguish characteristics, we know that the record is TCP.
Flags can be any of the list:

TCP Flag	Flag Representation	Flag Meaning
SYN	S	This is a session establishment request, which is the first part of any TCP connection
FIN	F	This flag indicates the sender's intention to gracefully terminate the sending host's connection to the receiving host.
RESET	R	This flag indicates the sender's intention to immediately abort the existing connection with the receiving host.
PUSH	P	This flag immediately "pushes" data from the sending host to the receiving host's application software. There is no waiting for the buffer to fill up. In this case, responsiveness, not bandwidth efficiency, is the focus.

We can also find ACK (Acknowledgement), URG (Urgent) and "." (Placeholder) flags following the ones above.

ACK	ack	This is used to generally to acknowledge the receipt of data from the sender.
URGENT	urg	This flag indicates that there is "urgent" data that should take precedence over other data.
Placeholder	.	If connection does not have a SYN, FIN, RESET or PUSH flag set, a placeholder (a period) will be found after the destination port.

A TCP outputs:

```
19:39:55.820857 hostA.55021 > hostB.20: . ack 54663 win 64240
```

UDP records are likely to have the word udp in the *protocol* field in output. Although true most of the time, TCPdump analyzes some UDP services, such as Domain Name Service (DNS) and Simple Network Management Protocol (SNMP), at the application level in addition to the protocol level as UDP. Like Ethereal, TCPdump is a protocol aware and can interpret normally coded payloads of certain protocols.

```
19:39:55.820925 hostA.27021 > hostB.25249:  udp 53 (DF)
```

Finally, *ICMP* is easily to identify because the word icmp appears, without exception, in the TCPdump output.

```
19:40:02.218130 hostA.136 > hostB.51: icmp: echo request
```

We monitored a trace for a full 24 hour, weekday period and for both link directions, and got these results:

	Total Packets	Total Percentage	Total MBytes
Total	2 277 548 800	100 %	1 066 115
	Packets	% Packets	MBytes
TCP	2 206 717 033	96.89 %	1 032 958
UDP	53 977 906	2.37 %	25 267
ICMP	15 031 822	0.66 %	7 037
OTHER	1 822 039	0,08 %	853

Table 1: Protocol summary of traffic analysed

Brief statistics on the traffic data collected are given in Table 1. Other protocols were observed in the trace, namely IGRMP, IPv6-crypt, PIM, ESP and private encryption, but the largest of them accounted for fewer than 1 million packets (less than 0.05 %) over the 24 hour period and the total of all OTHER protocols was fewer than 1.9 million packets.

All percentage values given henceforth are from the total of UDP, ICMP and TCP packets only.

Given the large number of identified applications, and for ease of presentation, we group applications into types according to their potential requirements from the network infrastructure. Table 2 indicates nine such classes of traffic.
Importantly, while each flow is mapped to only one category, the characteristics of the traffic within each category are not necessarily unique. For example, the BULK category which is made up of ftp traffic consists of both the ftp control channel which transfers data in both directions, and the ftp data channel which consists of a simplex flow of data for each object transferred. The grouping of applications into the categories we have given is largely a user-centric grouping.

Classification	Example Application
BULK	ftp
INTERAKTIVE	ssh, telnet
MAIL	smpt, pop2 and 3, imap
SERVICES	dns, ldap, ntp, auth
WWW	http, https, http_alternative
MULTIMEDIA	rtsp, Real_media
P2P	KaaZa, BitTorrent, GnuTella, eDonkey, Napster, DirectConnect
CHAT	Yahoo, AOL, MSN, IRC
GAMES	HalfLife, WarCraft

Table 2: Network traffic allocated to each category

Our content-based classification scheme can be viewed as an iterative procedure whose target is to gain sufficient confidence that a particular traffic stream is caused by a specific application. Grouping packets into flows allows for more-efficient processing of the collected information as well the acquisition of the necessary context for an appropriate identification of the network application responsible for a flow.
The first step we need to take is that of aggregating packets into flows according to their 5-tuple. In the case of TCP, additional semantics can also allow for the identification of the start and end time of the flow. The fact that we observe traffic in both directions allows classification of all nearly flows on the link. A traffic monitor on a unidirectional link can identify only those applications that use the monitored link for their datapath.

Type	INTERACTIVE	BULK	WWW
Session	A session starts when the TCP connection is opened and end when the connection is closed or aborted.	A session starts when the control connection is opened and ends when the control connection is closed.	A session starts with the first HTTP request issued by a user after a dormant period during which no HTTP interactions were seen. The session ends at the start of the next dormant period.

Table 3: Sessions overview

One outcome of this operation is the identification of unusual or peculiar flows specifically simplex flows. These flows consist of packets exchanged between a particular port/protocol combination in only one direction between two hosts. A common cause of a simplex flow is that packets have been sent to an invalid or non-responsive destination host. The data of the simplex flows were not discarded, they were classified commonly identified as carrying worm and virus attacks. The identification and removal of simplex flows (each flow consisting of between three and ten packets sent over a 24-hour period) allowed the number of unidentified flows that needed further processing to be significantly reduced.

The second step of our method iteratively tests flow characteristics against different criteria until sufficient certainty has been gained as to the identity of the application. Such a process consists of seven different identification sub-methods. We describe these mechanisms in the next section. Each identification sub-method is followed by the evaluation of the acquired certainty in the candidate application. Currently this is a manual process.

4.2 Identification methods

The seven distinct identification methods applied by our scheme are listed in Table 4. Alongside each method is an example application that we would identify using this method. Each one tests a particular property of the flow attempting to obtain evidence of the identity of the causal application.

	Identification method	Example
1	Port-based classification	
2	Packet header	Simplex flow
3	Single packet signature	Many worms/virus
4	Single packet protocol	IDENT
5	Signature on the first 1024 Byte	P2P
6	First 1024 Byte protocol	SMTP
7	Flow protocol	FTP

Table 4: Flow identification methods.

Method 1 classifies flows according to their port numbers. This method requires access only to the part in the packet header that contains the port numbers. Method 2 relies on access to the to the entire packet header for both traffic directions. It is this method that is able to identify simplex flows and significantly limit the numbers of flows that need to go through the remainder of the classification process. Methods 3 to 6 examine whether a flow carries a well-known signature or follows well-known protocol semantics. Such operations are accompanied by higher complexity and may require access to more that a single packet's payload. According to our experience, specific flows may be classified positively from their first packet alone. However, others flows may need to be examined in more detail and a positive identification may be feasible once up top 1024 Bytes of their data has been observed. Flows that have not been classified at this stage will require inspection of the entire flow payload. In method 7 we perform full-flow analysis for a subset of the flows that perform a control function. The control messages were parsed and further context was obtained that allowed us to classify more flows in the trace.

In our classification technique we will apply each identification method in turn and in such a way that the more-complex or more-data-demanding methods are used only if no previous signature or protocol method has generated a match. The outcome of this process may be:

1. We have positively identified a flow to belong to a specific application
2. A flow appears to agree with more than one application profile
3. No candidate application has been identified.

In our current methodology all three cases will trigger manual intervention in order to validate the accuracy of the classification, resolve cases where multiple criteria have generated a match or inspect flows that have not matched any identification criteria. We describe our validation approach in more detail in Section 4.4.

An illustration of the flow through the different identification sub-methods, as employed by our approach, is shown in Figure 5. In the first step we attempt to reduce the number of flows to be further processed by using context obtained through previous iterations. Specific flows in our data can be seen as "child" connections arising from "parent" connections that precede them. One such example is a web browser that initiates multiple connections in order to retrieve parts of a single web page. Having parsed the "parent" connection allows us to immediately identify the "child" connections and classify them to the causal web application.

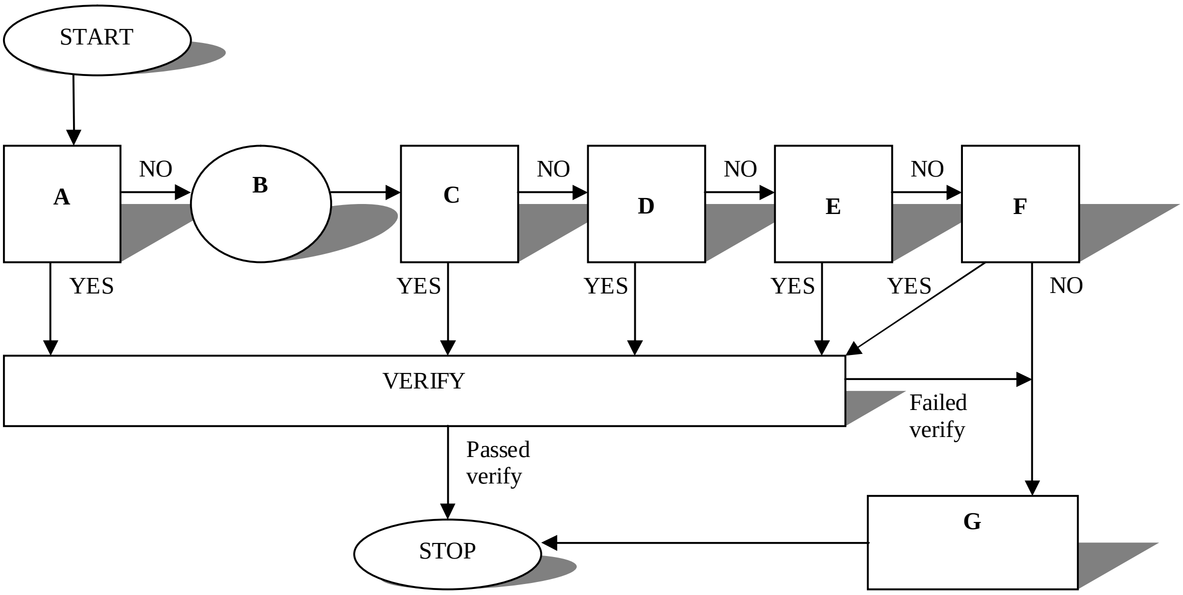


Figure 5: Flow diagram over classification approaches

A. Is flow result of another application?
B. Tag flows with known ports
C. First packet “well known” signature?
D. First 1024 Byte “well known” signature?
E. First 1024 Byte “well known” protocol?
F. Flow contains known protocol?
G. Manual intervention

Another example, is passive FTP. Parsing the ”parent” FTP session (Method 7) allows the identification of the subsequent ”child” connection that may be established toward a different host at a non-standard port. Testing whether a flow is the result of an already-classified flow at the beginning of the classification process allows for the fast characterization of a network flow without the need to go through the remainder of the process.

4.3 Classification process

If the flow is not positively identified in the first stage then it goes through several additional classification criteria. The first mechanism examines whether a flow uses a well-known port number. While port-based classification is prone to error, the port number is still a useful input into the classification process because it may deliver useful information about the identity of the flow.

Service	Port
ftp	20 = ftp-data and 21 = ftp
ssh, telnet	22, 23
smpt, pop2 and 3, imap	25,109 and 110, 143
dns, ldap, ntp, auth	53, 389, 123, 113
http, https, http_alternative	80, 443, 8080
rtsp, Real_media	554, 7070
Kaaza, BitTorrent, Gnutella,	1214, 6881-6889, 6346 and 6347
eDonkey, Napster, DirectConnect	4661-4672, 6699-6701, 412
Yahoo, AOL, MSN, IRC	5050, 5190, 1863, 2337
HalfLife, WarCraft	27005-27030, 6112-6119

Table 5: Port-number table

If no well-known port is used, the classification proceeds through the next stages.
In the next stage we test whether the flow contains a known signature in its first packet. At this point we will be able to identify flows that may be directed to well-known port numbers but carry non-legitimate traffic as in the case of virus or attack traffic. Signature-scanning is a process that sees common use within Intrusion Detection Systems such as Snort [6].

4.3.1 Snort

Snort is a signature-based Network Intrusion Detection System that uses a combination of rules and preprocessors to analyze traffic. The rules offer a simple and flexible means of creating signatures to examine a single packet. The preprocessor code allows more extensive examination and manipulation of data that cannot be done via rules alone. Preprocessors can perform a variety of tasks such as IP defragmentation, portscan detection, web traffic normalization and TCP stream reassembly.

Snort comes with a very large set of rules. These rules are updated continuously as new exploits are discovered. The latest rules can be downloaded from www.snort.org/rules/. Some of the rules needs to be tuned and adapted to the local site. In order to do this and understand the alerts given by Snort, it is necessary to know how the rules are constructed. A Snort rule is made of a rule header and rule options.

The header specifies what to do with packets of the given protocol, IP addresses and ports. It consists of at most seven fields:

Action Protocol IP port -> IP port
Action Protocol IP port <> IP port

where the arrows specifies one-directional or bi-directional flow.

A sample rule that was detected:

alert tcp any any -> any any (msg: "nmap TCP ping"; flag: A; ack: 0;)

The rule options follows the header and are contained within parentheses seperated by a semicolon which acts as a logical AND.

Understanding alert output:

```
[**] [1:469:3] nmap TCP PING [**]
05/19-21:05:35.078755 128.39.89.9 -> 128.39.89.2
TCP TTL:54 TOS:0x0 ID:37594
***A**** Seq: 0x1668004 Ack: 0x0 Win: 0xC00
```

The acknowledgement option examines the values of TCP acknowledgement number. The primary use of this currently is to detect nmap pings. When namp tries to access if a host is alive, it sends a unique signature. It sets ACK flag on, and it sets the acknowledgement values of 0. this would be a rare setting to find in a normal traffic because it would be indicative of an already established connecting acknowledging that the previous TCP sequence number received was 2^32 – 1, and now the acknowledgment number is wrapping back to 0.

4.3.2 IDENT protocol

If no known signature has been found in the first packet we check whether the first packet of the flow delivers semantics of a well-known protocol. An example to that is IDENT protocol [3] which is a single packet IP protocol.
The IDENT protocol is often used by TELNET, POP mail, FTP, and HTTP servers to identify incoming users.

This is how it works: A server listens for TCP connections on TCP port 113. Once a connection is established, the server reads a line of data which specifies the connection of interest. If it exists, the system dependent user identifier of the connection of interest is sent as the reply. The server may then either shut the connection down or it may continue to read/respond to multiple queries.

If this test fails we look for well-known signatures in the first 1024 Bytes of the flow, which may require assembly of multiple individual packets. At this stage we will be able to identify peer-to-peer traffic if it uses well known signatures.

4.3.3 P2P protocols and signatures

Historically in the client/server model content is stored on the server and all clients download content from the server. One drawback of this model is that if the server is overloaded, the server becomes the bottleneck. The P2P file sharing model addresses this problem by allowing peers to exchange content directly. To perform these file sharing tasks, all popular P2P protocols allow a random host to act as both a client and a server to its peers, even though some P2P protocols do not treat all hosts equally.

Typically the following two phases are involved if a requester desires to download content:

Signaling: During the signaling phase a client searches for the content and determines which peers are able and willing to provide the desired content. In many protocols this does not involve any direct communication with the peer which will eventually provide the content.

Download: In this phase the requester contacts one or multiple peers directly to download the desired content.

In addition to the two phases described above many P2P protocols also exchange keep-alive messages or synchronize the server lists between servers.

In the remainder of the thesis we focus on the download phase of the five most popular P2P protocols (Kazaa, Gnutella, eDonkey, DirectConnect, and BitTorrent). Unless otherwise specified, all the identified signatures are case insensitive.

Gnutella protocol

Gnutella is a completely distributed protocol. In a Gnutella network, every client is a server and vice versa. Therefore the client and server are implemented in a single system, called servent. A servent connects to the Gnutella network through establishing a TCP connection to another servent on the network. Once a servent has connected successfully to the network, it communicates with other servents using Gnutella protocol descriptors for searching the network - this is the signaling phase of the protocol. The actual file download is achieved using a HTTP-like protocol between the requesting servent and a servent possessing the requested file.

To develop the Gnutella signature we inspected multiple Gnutella connections and observed that the request message for Gnutella TCP connection creation assumes following format:

```
GNUTELLA CONNECT/<protocol version string>\n\n
```

And the response message for Gnutella TCP connection creation assumes:

```
GNUTELLA OK\n\n
```

We also observed that there is an initial request-response handshake within each content download. In the download request the servent uses the following HTTP request headers:

```
GET /get/<File Index>/<File Name>
/HTTP/1.0 \r \n
Connection: Keep-Alive\r\n
Range: byte=0-\r\n
User-Agent: <Name>\r\n
\r\n
```

The reply message contains the following HTTP response headers:

```
HTTP 200 OK\r\n
Server: <Name>\r\n
Content-type: \r\n
Content-length: \r\n
\r\n
```

Based on these observations and performance consideration, we have the following signatures for identifying Gnutella data downloads:

- The first string following the TCP/IP header is ‘GNUTELLA’, ‘GET’, or ‘HTTP’.
- If the first string is ‘GET’ or ‘HTTP’, there must be a field with one of following strings:

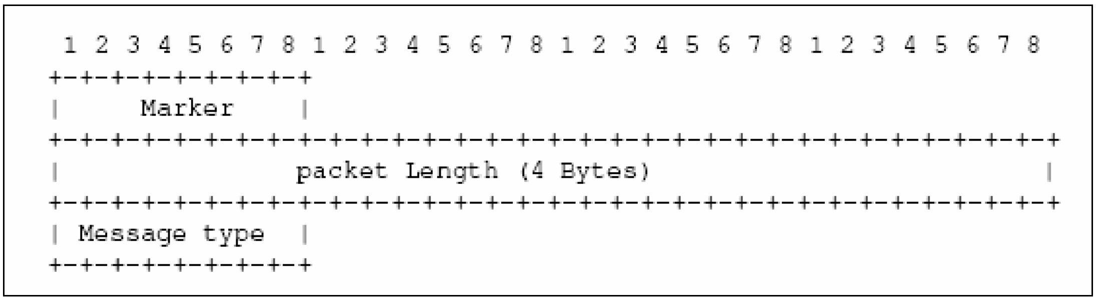
  ```
  User-Agent: <Name>
  UserAgent: <Name>
  Server: <Name>
  ```

  where <name> is one of the following: LimeWire, BearShare, Gnucleus, MorpheusOS, XoloX, MorpheusPE, gtkgnutella, Acquisition, Mutella-0.4.1, MyNapster, Mutella-0.4.1, MyNapster, Mutella-0.4, Qtella, AquaLime, NapShare, Comeback, Go, PHEX, SwapNut, Mutella-0.4.0, Shareaza, Mutella-0.3.9b, Morpheus, FreeWire, Openext, Mutella-0.3.3, Phex.

Generally it is much cheaper to match a string with a fixed offset than a string with varying locations. Hence we include ‘GET’ and ‘HTTP’ here to help early discard the packets, which do not start with ‘GNUTELLA’, and also are non-HTTP packets. For robustness, we included the signatures for the request and response header. This way, we can identify Gnutella traffic even if we only see one direction of the traffic.

eDonkey protocol

An eDonkey network consists of clients and servers. Each client is connected to one main server via TCP. During the signaling phase, it first sends the search request to its main server. (Optionally, the client can send the search request directly to other servers via UDP - this is referred to as extended search in eDonkey.) To download a file subsequently from other clients, the client establishes connections to the other clients directly via TCP, and then asks each client for different pieces of the file. After examining eDonkey packets, we discovered that both signaling and downloading TCP packets have the following common eDonkey header directly following the TCP header:



```
 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8 1 2 3 4 5 6 7 8
+-+-+-+-+-+-+-+-+
|     Marker    |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
|               packet Length (4 Bytes)                         |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| Message type  |
+-+-+-+-+-+-+-+-+
```

where the marker value is always 0xe3 in hex, the packet length is specified in network byte order and the value is the byte length of the content of the eDonkey message excluding the marker 1 byte and the length field 4 bytes.
Utilizing these discoveries, we have the following signatures for identifying eDonkey packets:

For TCP signaling or handshaking data packets, we use two steps to identify eDonkey packets.

- The first byte after the IP+TCP header is the eDonkey marker.
- The number given by the next 4 bytes is equal to the size of the entire packet after excluding both the IP+TCP header bytes and 5 extra bytes.

Since the accuracy for identifying the P2P connections is proportional to the length of the signatures, we tend to include as many fields as we can so long as they do not increase the computational complexity significantly. Here both marker and length fields have a fixed offset, therefore the computational complexity is the same for matching one of them or both, but the accuracy is improved by 2^32 times compared with matching the marker field alone.

We have also identified the signatures for UDP handshaking messages. However, since UDP is only used for extended searching, and is rare compared with TCP communications, we do not report it in this study.

DirectConnect protocol
The DirectConnect network is composed of hubs, clients, and a single superhub with multiple servers. All of them listen on TCP port 411 to connect and exchange commands such as search request. Clients (peers) store files and respond to search requests for those files. The single superhub acts as a name service for all the hubs. All hubs register with the superhub and clients discover hubs by asking the superhub. Each of the clients has a username (a.k.a. nick). Normally the clients listen at port 412 for client connections. If the port 412 is already in use, clients will use ports 413, 414 and so on. DirectConnect uses TCP for client to server and client to client communication, while UDP is used for communication between servers. The TCP/UDP data is a series of commands or a public chat message. In this study, we focus on the TCP commands. The TCP commands are identified with following form:

```
$command_type field1 field2 ...|
```

which starts with character '$', and ends with character '|'. The list of valid command types for TCP communications are: MyNick, Lock, Key, Direction, GetListLen, ListLen, MaxedOut, Error, Send, Get, FileLength, Canceled, HubName, ValidateNick, ValidateDenide, GetPass, Mypass, BadPass, Version, Hello, Logedin, MyINFO, GetINFO, GetNickList, NickList, OpList, To, ConnectToMe, MultiConnectToMe, RevConnectToMe, Search, MultiSearch, SR, Kick, OpForceMove, ForceMove, Quit.

To improve the evaluation performance we evaluate this signature in the following two steps:

1. The first byte after the IP+TCP header is '$', and the last byte of the packet is '|'
2. Following the '$', the string terminated by a space is one of the valid TCP commands listed above.

Although we are matching a list of strings which can be an expensive operation, we shall only perform the string match on packets which pass the first test.

BitTorrent protocol

The BitTorrent network consists of clients and a centralized server. Clients connect to each other directly to send and receive portions of a single file. The central server (called a tracker) only coordinates the action of the clients, and manages connections. Unlike the protocols discussed above, the BitTorrent server is not responsible for locating the searching files for the clients, instead the BitTorrent network client locates a torrent file through the Web, and initiates the downloading by clicking on the hyperlink. Hence there is no signaling communication for searching in the BitTorrent network.

To identify BitTorrent traffic, we focus on the downloading data packets between clients only since the communication between the client and server is negligible.

The communication between the clients starts with a handshake followed by a never-ending stream of length-prefixed messages. We discovered that the BitTorrent header of the handshake messages assumes following format:

```
<a character(1 byte)><a string(19 byte)>
```

The first byte is a fixed character with value '19', and the string value is 'BitTorrent protocol'. Based on this common header, we use following signatures for identifying BitTorrent traffic:

- The first byte in the TCP payload is the character 19 (0x13).
- The next 19 bytes match the string 'BitTorrent protocol'.

The signatures identified here are 20 bytes long with fixed locations - therefore they are very accurate and cost-effective.

Kazaa protocol

The Kazaa network is a distributed self-organized network. In a Kazaa network, clients with powerful connections and with fast computers are automatically selected as supernodes. Supernodes are local search hubs. Normal clients connect to their neighboring supernodes to upload information about files that they share, and to perform searches. In turn supernodes query each other to fulfill the search.

The request message in a Kazaa download contains the following HTTP request headers:

```
GET /.files HTTP/1.1\r\n
Host: IP address/port\r\n
UserAgent: KazaaClient\r\n
X-Kazaa-Username: \r\n
X-Kazaa-Network: KaZaA\r\n
X-Kazaa-IP: \r\n
X-Kazaa-SupernodeIP: \r\n
```

The Kazaa response contains the following HTTP response headers:

```
HTTP/1.1 200 OK\r\n
Content-Length: \r\n
Server: KazaaClient\r\n
X-Kazaa-Username: \r\n
X-Kazaa-Network: \r\n
X-Kazaa-IP: \r\n
X-Kazaa-SupernodeIP: \r\n
Content-Type: \r\n
```

For higher Kazaa version (v1.5 or higher), a peer may send an encrypted short message before it sends back above response. Note that both messages include a field called X-Kazaa-SupernodeIP. This field specifies the IP address of the supernode to which the peer is connected including the TCP/UDP supernode service port. This information could be used to identify signaling using flow records of all communication.

Using the special HTTP headers found in the Kazaa data download we have the following two steps to identify Kazaa downloads:

1. The string following the TCP/IP head is one of following: ‘GET’, and ‘HTTP’.
2. There must be a field with string: X-Kazaa.

Similar to our Gnutella signatures we include ‘GET’ and ‘HTTP’ to early discard non-HTTP packets, so that we can avoid searching through the whole packet to match ‘X-Kazaa’ if the packet has a low probability to contain HTTP request or response headers.

4.3.4 Other applications

Traffic due to SMTP will have been detected from the port-based classification but only the examination of the protocol semantics within the first 1024 Byte of the flow will allow for the confident characterization of the flow. Network protocol analysis tools, ethereal [7], employ a number of such protocol decoders and may be used to make or validate protocol identification.

Specific flows will still remain unclassified even at this stage and will require inspection of their entire payload. This operation may be manual or automated for particular protocols. From our experience, focusing on the protocol semantics of P2P and FTP led to the identification of a very significant fraction of the overall traffic limiting the unknown traffic to less than 7%. At this point the classification procedure can end. However, if 100% accuracy is to be approached we envision that the last stage of the classification process may involve the manual inspection of all unidentified flows. This stage is rather important since it is likely to reveal new applications. While labour-intensive, the individual examination of the remaining, unidentified, flows caused the creation of a number of new signatures and protocol-templates that were then able to be used for identifying protocols such as PCAnywhere, the sdserver and CVS. This process also served to identify more task-specific systems. An example of this was a host offering protocol-specific database services.

On occasion flows may remain unclassified despite this process; this takes the form of small samples (e.g., 1-2 packets) of data that do not provide enough information to allow any classification process to proceed. These packets used unrecognized ports and rarely carried any payload. While such background noise was not zero in the context of classification for accounting, Quality-of-Service, or resource planning, these amounts could be considered insignificant. The actual amount of data in terms of either packets or bytes that remained unclassified represented less than 0.001% of the total.

4.4 Validation Process

Accurate classification is complicated by the unusual use to which some protocols are put. As noted earlier, the use of one protocol to carry another, such as the use of HTTP to carry peer-to-peer application traffic, will confuse a simple signature-based classification system. Additionally, the use of FTP to carry an HTTP transaction log will similarly confuse signature matching.

Due to these unusual cases the certainty of any classification appears to be a difficult task. Throughout the work presented in this thesis validation was performed manually in order to approach 93% accuracy in our results. Our validation approach features several distinct methods.

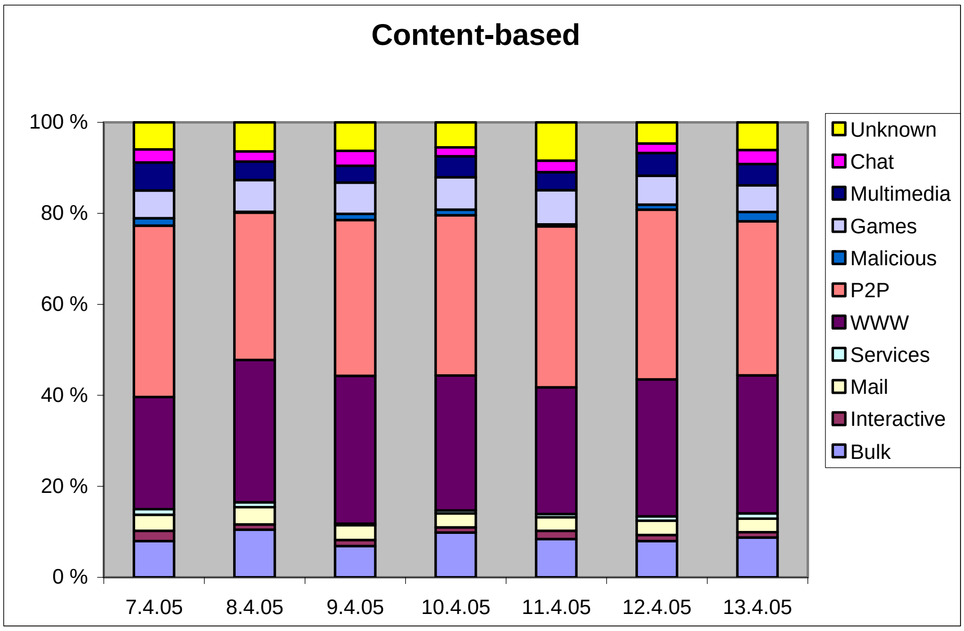
Each flow is tested against multiple classification criteria. If this procedure leads to several criteria being satisfied simultaneously, manual intervention can allow for the identification of the true causal application. An example is the peer-to-peer situation. Identifying a flow as HTTP does not suggest anything more than that the flow contains HTTP signatures. After applying all classification methods we may conclude that the flow is HTTP alone, or additional signature-matching (e.g. identifying a peer-to-peer application) may indicate that the flow is the result of a peer-to-peer transfer.

If the flow classification results from a well-known protocol, then the validation approach tests the conformance of the flow to the actual protocol. An example of this procedure is the identification of FTP PASSIVE flows. PASSIVE flows can be valid only if the FTP control-stream overlaps the duration of the PASSVE flow - such cursory, protocol-based, examination allows an invalid classification to be identified. Alongside this process, flows can be further validated against the perceived function of a host, e.g., an identified router would be valid to relay BGP whereas for a machine identified as (probably) a desktop Windows box behind a NAT, concluding it was transferring BGP is unlikely and this potentially invalid classification requires manual-intervention.

5 Results

5.1 Data

In Table 6 we compare the results of simple port-based classification with content-based classification. The technique of port-analysis, against which we compare our approach, is common industry practice (e.g., Cisco NetFlow or [34]). UNKNOWN refers to applications which for analysis are not readily identifiable. Notice that under the content-based classification approach we had much lower UNKNOWN traffic. We also detected a new traffic-class – MALICIOUS type. The traffic were not able to classify corresponds to a small number of flows. A limited number of flows provides a minimal sample of the application behavior and thus cannot allow for the confident identification of the causal application.



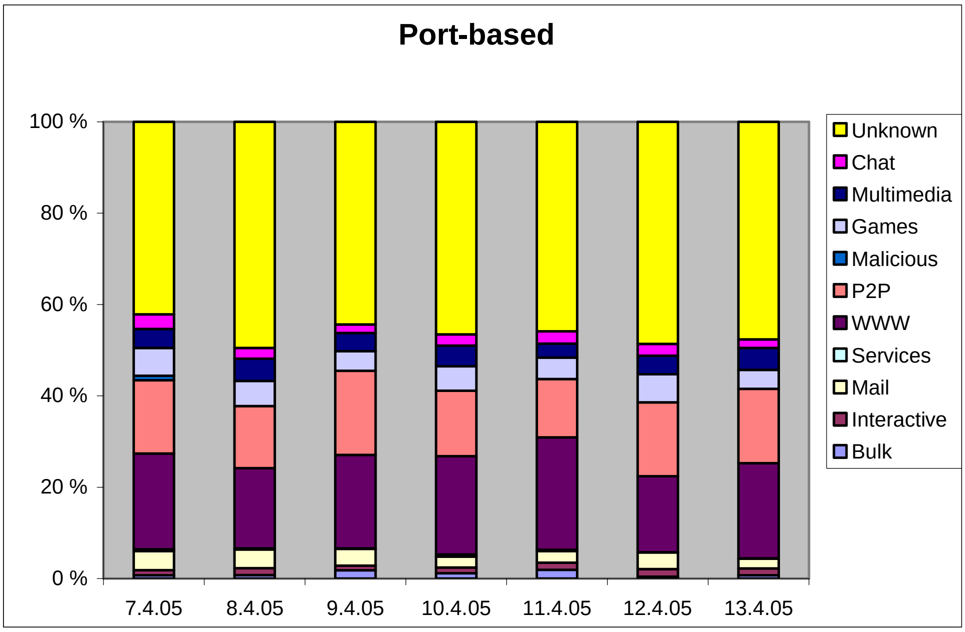


Figure 6: Percentile bar-chart for all services, 7 days, content-based and port-based methods.

After calculating the averages for all services we got these results:

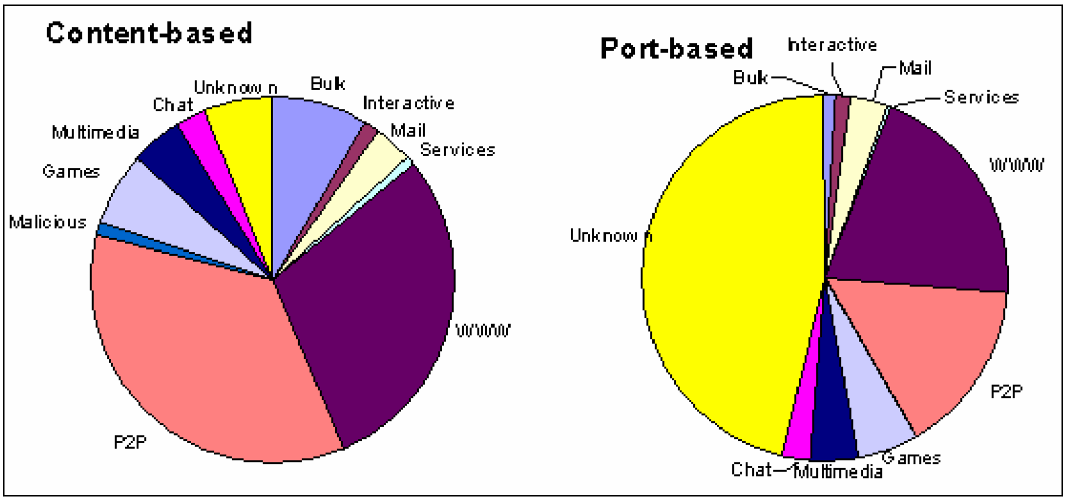


Figure 7: Pie-chart for all services, the averages, content-based and port-based methods.

Classification type	Port-based	Content-based
BULK	1.04 %	8.59 %
INTERACTIVE	1.51 %	1.47 %
MAIL	3.25 %	3.25 %
SERVICES	0.22 %	0.87 %
WWW	20.29 %	29.47 %
P2P	15.35 %	35.25 %
MALICIOUS	0.00 %	1.17 %
GAMES	5.21 %	6.67 %
MULTIMEDIA	4.22 %	4.49 %
CHAT	2.44 %	2.59 %
UNKNOWN	46.47 %	6.18 %

Table 6: Results for port-based and content-based

Table 6 shows that under the simple port-based classification scheme based upon the IANA port assignments 46% of the carried bytes cannot be attributed to a particular application. Further observation reveals that the BULK traffic is underestimated by approximately 7.5% while we see a difference of 9% in the WWW traffic. However, the port-based approach does not only underestimate traffic but for some classes, e.g., INTERACTIVE applications, it may over-estimate it. This means that traffic flows can also be misidentified under the port-based technique. Lastly, applications such as peer-to-peer and mal-ware appear to contribute zero traffic in the port-based case. This is due to the port through which such protocols travel not providing a standard identification. Such port-based estimation errors are believed to be significant.

5.2 Examining under and over-estimation

Of the results in Table 6 we will concentrate on only a few example situations.
The first and most dominant difference is for BULK traffic created as a result of FTP. The reason is that port-based classification will not be able to correctly identify a large class of FTP traffic transported using the PASSIVE mechanism. Content-based classification is able to identify the causal relationship between the FTP control flow and any resulting data-transport. This means that traffic that was formerly either of unknown origin or incorrectly classified may be ascribed to FTP which is a traffic source that will be consistently underestimated by port-based classification.

A comparison of values for MAIL, a category consisting of the SMTP, IMAP and POP protocols, reveals that it is estimated with surprising accuracy in both cases. Both the number of packets and bytes transferred is unchanged between the two classification techniques. We also did not find any other non-MAIL traffic present on MAIL ports. We would assert that the reason MAIL is found exclusively on the commonly defined ports, while no other MAIL transactions are found on other ports, is that MAIL must be exchanged with other sites and other hosts. MAIL relies on common, Internet-wide standards for port and protocol assignment. No single site could arbitrarily change the ports on which MAIL is exchanged without effectively cutting itself off from exchanges with other Internet sites. Therefore, MAIL is a traffic source that, for quantifying traffic exchanged with other sites at least, may be accurately estimated by port-based classification.

Despite the fact that such an effect was not pronounced in the analyzed data set, port-based classification can also lead to over-estimation of the amount of traffic carried by a particular application. One reason is that mal-ware or attack traffic may use the well-known ports of a particular service, thus inflating the amount of traffic attributed to that application. In addition, if a particular application uses another application as a relay, then the traffic attributed to the latter will be inflated by the amount of traffic of the former. An example of such a case is peer-to-peer traffic using HTTP to avoid blocking by firewalls, an effect that was not present in our data. In fact, we notice that under the content-based approach we can attribute more traffic to WWW since our data included web servers operating on non-standard ports that could not be detected under the port-based approach.

Clearly this work leads to an obvious question of how we know that our content-based method is correct. We would emphasize that it was only through the labour-intensive examining of all data-flows. We do not consider that such a laborious process would need to be repeated for the analysis of similar traffic profiles. However, the identification of new types of applications will require a more limited examination of a future, unclassifiable anomaly.

5.3 Describing the results

There three main goals for this sub-chapter:

a) identifying the nature of the phenomenon represented by the sequence of observations,
b) finding the distribution of the intensity of the traffic for different services and
c) forecasting (predicting future values of the time series variable).

All of these goals require that the pattern of observed time series data is identified and more or less formally described. Once the pattern is established, we can interpret the trends of the data. Regardless of the depth of our understanding and the validity of our interpretation (theory) of the phenomenon, we can extrapolate the pattern to predict future events.

5.3.1 Estimating the traffic intensity

The traffic intensity for different services is needed for instance to describe the expectation or the entire distribution of the packets flow on the backbone link.
The data is a sequence of observations which are ordered in time. The observations are made on some phenomenon throughout time, and therefore it is most sensible to display the data in the order in which they arose, particularly since successive observations will probably be dependent.

We have taken into account the fact that the traffic intensity pattern during weekends and holidays differs quite much from the ordinary weekday pattern, (as seen in figure 9 where the dates 12/4 and 13/4 correspond to weekends) in that we estimate one weekday pattern for each of the classifications. The estimation is performed as follows for each one of the classification. First we partition each day into intervals of 60 seconds length and calculate the number of packets for that specific classification in each interval. After that we sort weekday into one group and weekends and holidays into another group. For both these groups we calculate the mean number of packets during each interval. This gives us a somewhat smoother description of the traffic intensity pattern (shows in figure 8). The series values are using content-based method and are plotted on the vertical axis and time on the horizontal axis. Time is called the independent variable (in this case however, something over which we have little control).

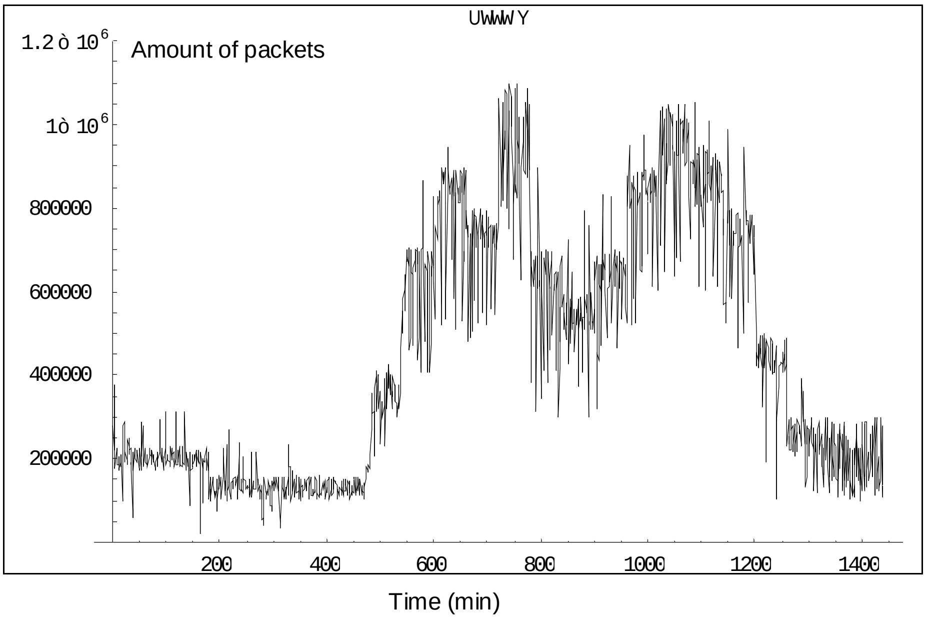
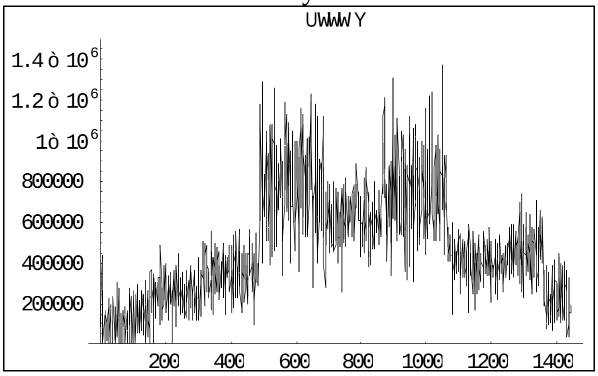
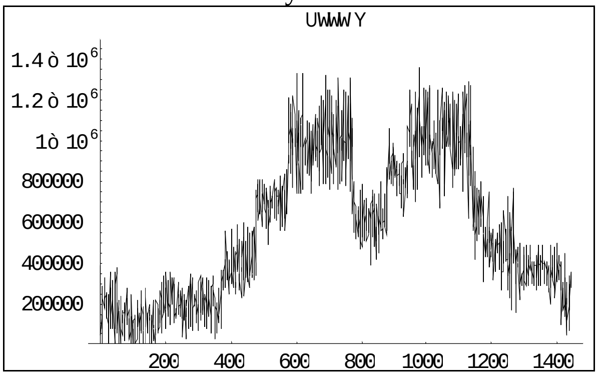


Figure 8: Traffic intensity pattern of WWW traffic, weekdays, using content-based method.

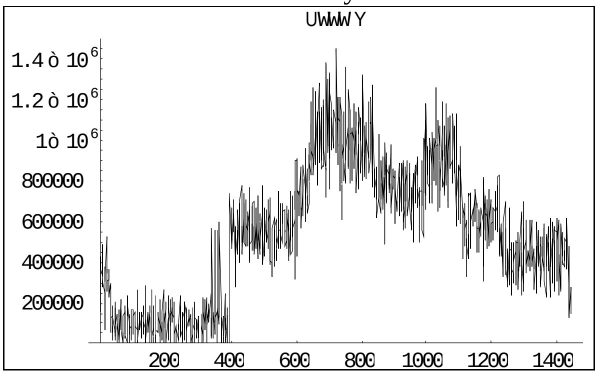
Monday 7/4 2005



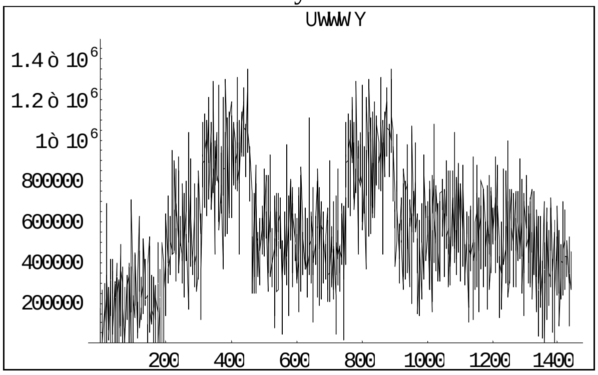
Tuesday 8/4 2005



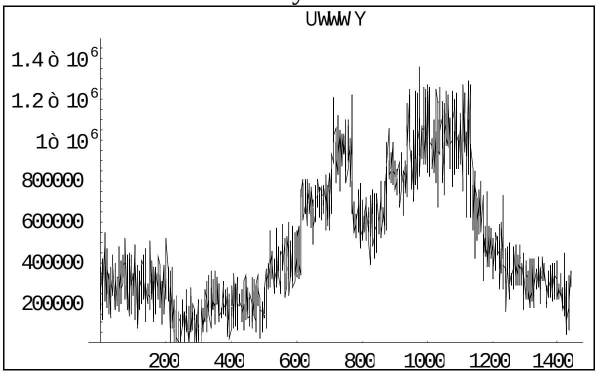
Wednesday 9/4 2005



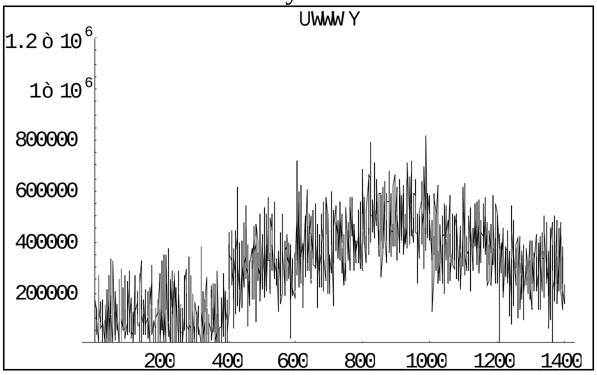
Thursday 10/4 2005



Friday 11/4 2005



Saturday 12/4 2005



Sunday 13/4 2005

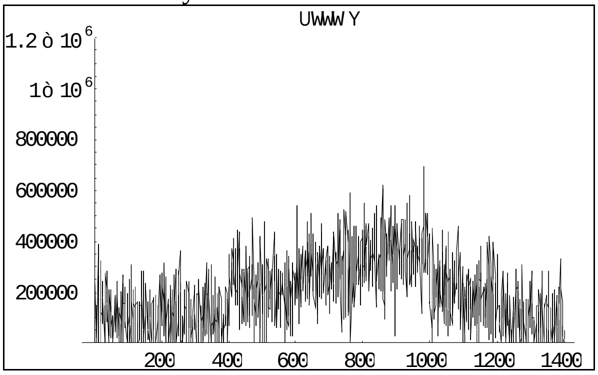


Figure 9: WWW traffic measurements using content-based method during 1 week.

In order to give an even smoother description we uses moving average, where we calculate the local average from our datasets. Moving average is an indicator that shows the average value of the intensity of the traffic over a period of time. To find the 50 moving average we would add up the closing points from the past 50 points and divide them by 50. And because traffics are constantly changing it means the moving average will move as well.

Given a sequence $\{a_i\}_{i=1}^{N}$, an *n*-moving average is a new sequence $\{s_i\}_{i=1}^{N-n+1}$ defined from the a_i by taking the average of subsequences of *n* terms, so the sequences S_n giving *n*-moving average are

$$S_2 = \tfrac{1}{2}(a_1 + a_2, a_2 + a_3, \ldots, a_{n-1} + a_n)$$

$$S_3 = \tfrac{1}{3}(a_1 + a_2 + a_3, a_2 + a_3 + a_4, \ldots, a_{n-2} + a_{n-1} + a_n).$$

and so on.

Smoothing techniques are used to reduce irregularities (random fluctuations) in time series data. They provide a clearer view of the true underlying behaviour of the series. In some time series, seasonal variation is so strong it obscures any trends or cycles which are very important for the understanding of the process being observed. Smoothing can remove seasonality and makes long term fluctuations in the series stand out more clearly. Since the type of seasonality will vary from series to series, so must the type of smoothing. When a variable is graphed against time, there are likely to be considerable seasonal or cyclical components in the variation. These may make it difficult to see the underlying trend. These components can be eliminated by taking a suitable moving average. By reducing random fluctuations, moving average smoothing makes long term trends clearer.

The moving averages that we uses are 20, 30, 50, 100, and 200. Each moving average provides a different interpretation on what the intensity of traffic will be. There really is not just one "right" time frame. Moving averages with different time spans each tell a different story. The shorter the time-span, the more sensitive the moving average will be to traffic changes. The longer the time-span, the less sensitive or the more smoothed the moving average will be. Moving averages are used to emphasize the direction of a trend and smooth out traffic and volume fluctuations or "noise" that can confuse interpretation.

n = 10: provides a very volatile, choppy line, but it has already smoothing out the “noise”.

n = 20: still it is very volatile, choppy line. It is not the most accurate, but is probably the most useful for short term traffics.

n = 30: similar to 20 but provides a bit more certainty for the trend.

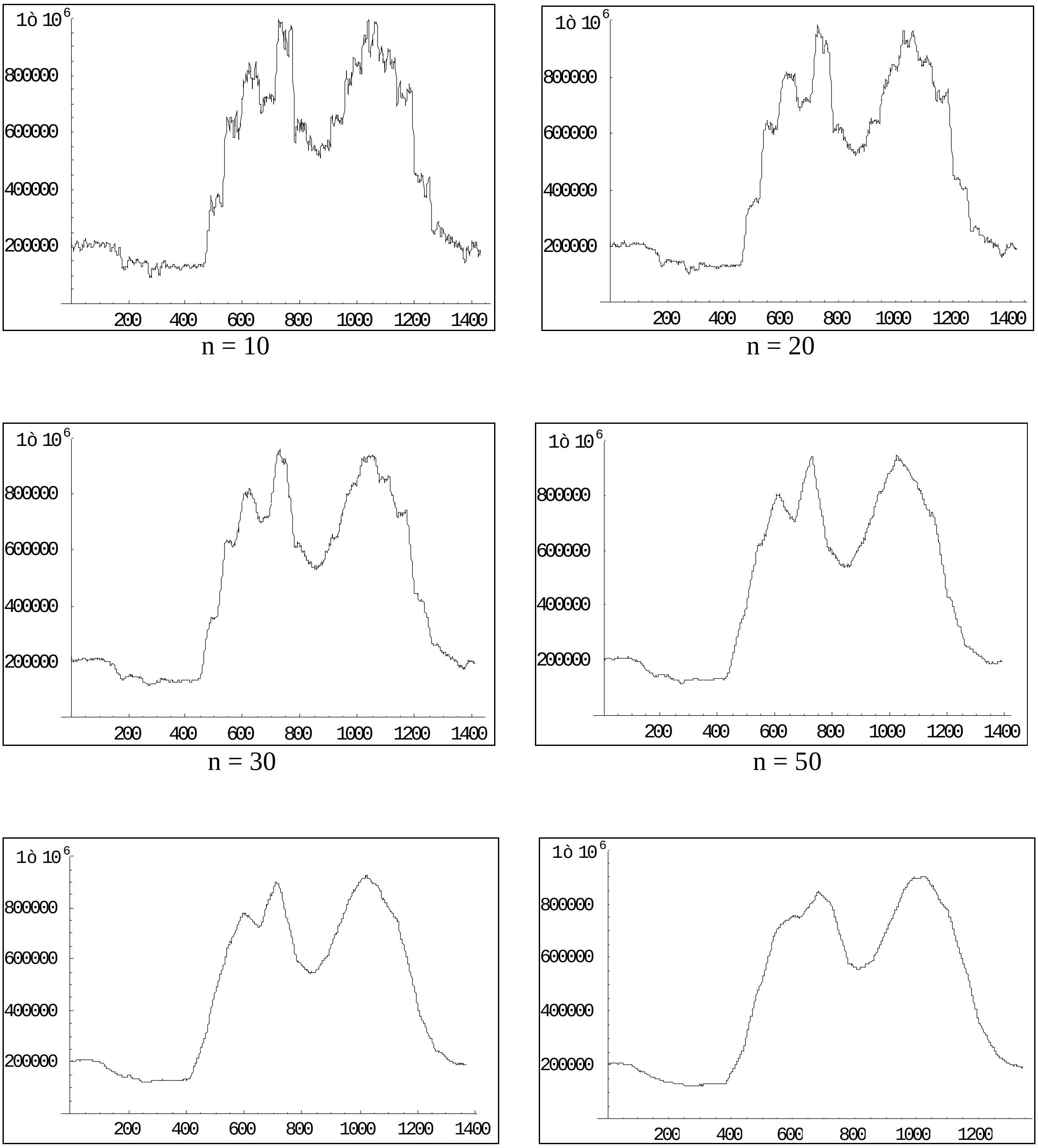
n = 50: moving averages provide a much less volatile, smooth line. This can be used to detect somewhat longer term trends.

n = 70: similar to the 70, it is less volatile, and one of the most widely used for long term trends.

n = 100: even less volatile, more of a rolling chart or smooth line. It does not react to quick movements in the intensity of traffic therefore it is rarely used by us.

Again, there isn't just one "right" time frame. Moving averages with different time spans each tell a different story. The shorter the time span, the more sensitive the moving average will be to traffic changes. The longer the time span, the less sensitive or the more smoothed the moving average will be.

The plot below shows the n = 10, 20, 30, 50, 70 and 100 moving averages for a set of 1440 data points using content-based method.



n = 10

n = 20

n = 30

n = 50

n = 70

n = 100

Figure 10: WWW traffic intensity, with different moving average, using content-based method, weekdays.

Using the procedure described above we arrive at estimators of the intensity pattern that are quite smooth but still give a realistic picture of the traffic behaviour as a function of time (see figure 11).

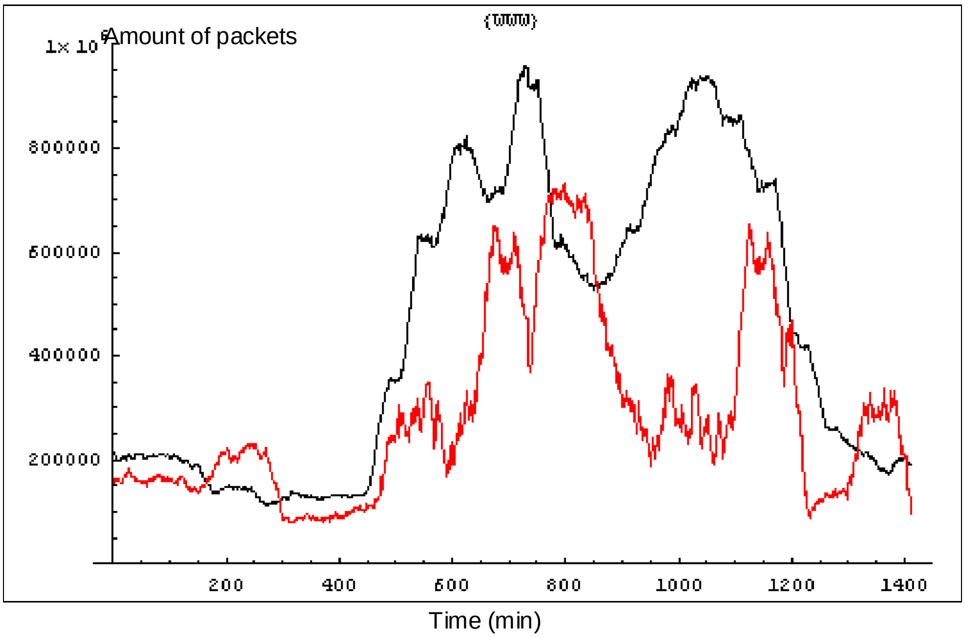


Figure 11: The WWW traffic intensity, using m=30, weekdays. Black line indicates using content-based method while red line is port-based.

The traffic intensity for different classification types

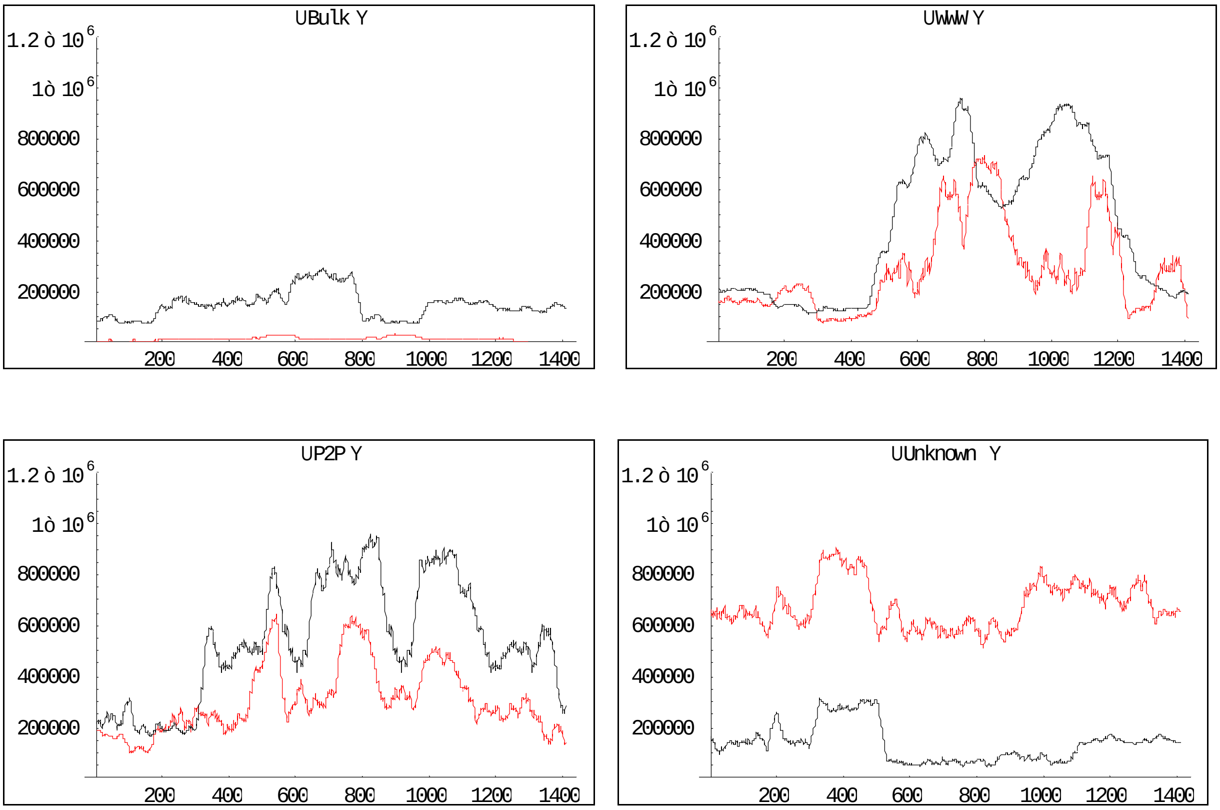


Figure 12: The traffic intensity for Bulk, WWW, P2P and Unknown type, using m=30, weekdays. Black line indicates using content-based method while red line is port-based.

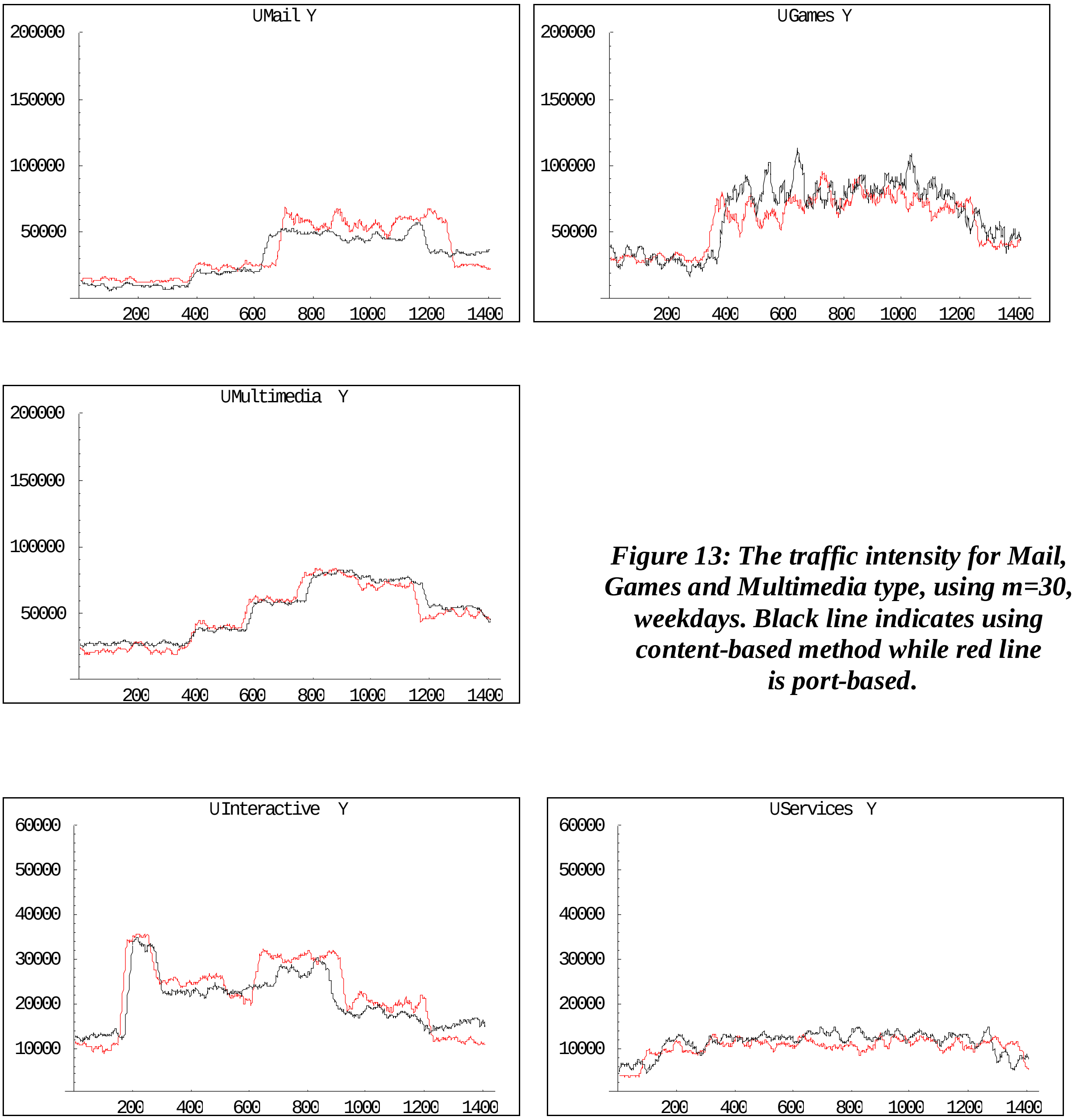


Figure 13: The traffic intensity for Mail, Games and Multimedia type, using m=30, weekdays. Black line indicates using content-based method while red line is port-based.

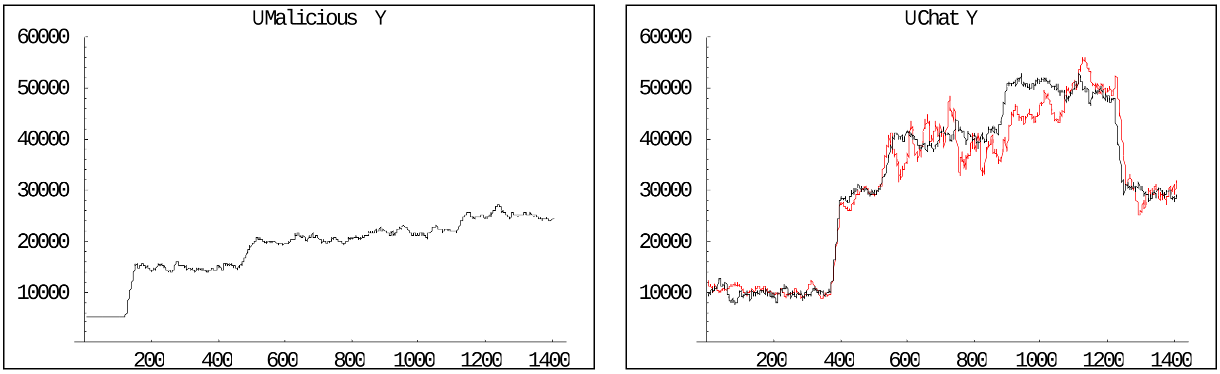


Figure 14: The traffic intensity for Interactive, Services, Malicious and Chat type, using m=30, weekdays. Black line indicates using content-based method while red line is port-based.

5.3.2 Descriptive statistics

While TCP remains the dominant traffic protocol through all hours of the day, a mixture of both well-known (http, ftp, nntp and smtp) and less known applications contribute significant portions to the traffic services.

We want to increase our understanding of our analysis by picking out its main features. Descriptive techniques may be extended to forecast (predict) future values.

Trend is a long term movement in the data series. It is the underlying direction (an upward or downward tendency) and rate of change in data series, when allowance has been made for the other components. A simple way of detecting trend in seasonal data is to take averages over a certain period. If these averages change with time we can say that there is evidence of a trend in the series. There are also more formal tests to enable detection of trend in data series.

Therefore we need to use mathematical summaries to carry out detailed statistical analysis. The aim of these summarizing processes is to allow us to find numbers that sum up the main characteristics of the large collections of data.

Frequency distribution

Statistical data obtained by experiments consist of raw, unorganized sets of numerical values. Before these data can be used as a basis for inferences about the phenomenon under investigation or as a basis for decision, they must be summarized and the pertinent information must be extracted.

We can convey an idea of pattern of the distribution as a whole by “picture it”. The dot-diagram reveals that the packet-rates tend to bunch around some parts of the range than other. We can use the frequency distribution as picture in the dot-diagram to check whether any packet-rate was more frequently recorded than other.
Let us first have looking at the WWW traffic. For instance, packet-rate between 150 000 and 250 000 has been recorded to have bigger rate than other using content-based method. While with port-based we recorded the packet-rate between 0 and 200 000 as the biggest rate. The value in distribution that has been observed with greatest frequency is called the mode of the distribution.

Mode

Mode is the value which occurs most frequency. For grouped data, the mode can be found by first identify the largest frequency of that class, called modal class, and then apply the following formula on the modal class:

$$\text{mode} = l_1 + \frac{f_a}{f_a + f_b}(l_2 - l_1)$$

where:

l_1 is the lower class boundary of the modal class
f_a is the difference of the frequencies of the modal class with the previous class and is always positive
f_b is the difference of the frequencies of the modal class with the following class and is always positive
l_2 is the upper class boundary of the modal class.

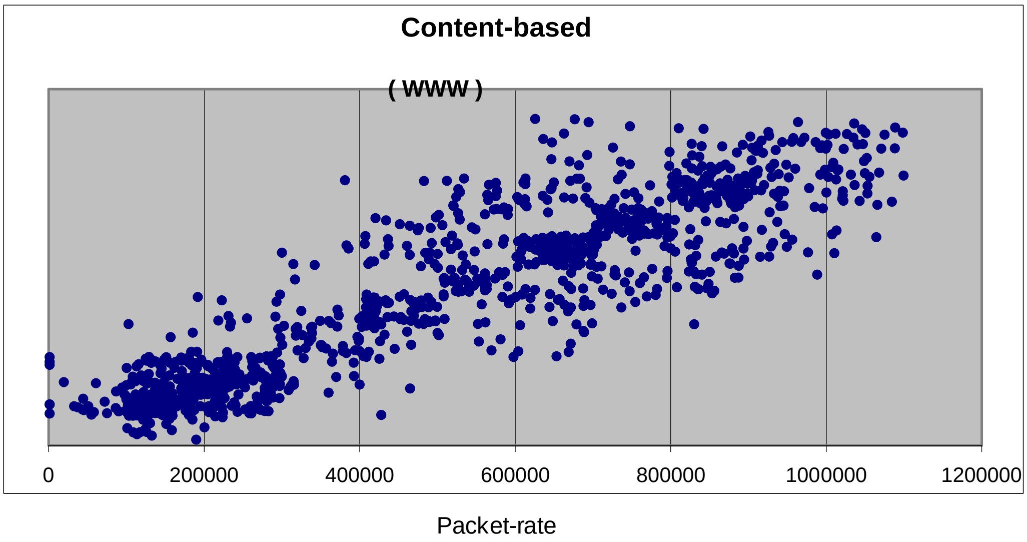


Figure 15: Dot diagram of WWW traffic intensity, weekdays, using content-based method.

Each dot represents amount of capture packets for a 60 seconds time-interval.

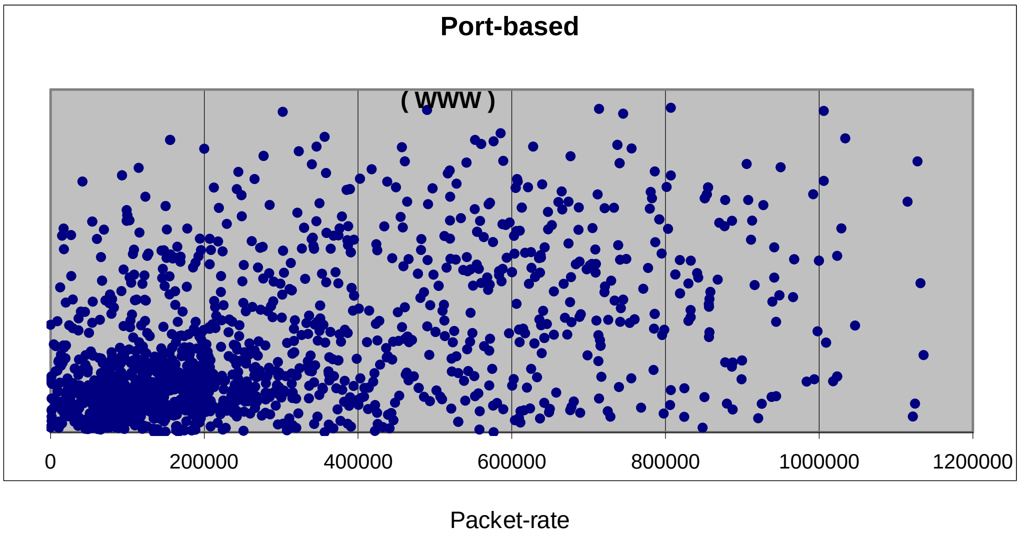


Figure 16: Dot diagram of WWW traffic intensity, weekdays, using port-based method.

What else can we do to bring out the pattern within the distribution?
A useful method for summarizing a set of data is the construction of a frequency table, or a frequency distribution. We cannot create a frequency distribution for continuous data in the same way as we do for discrete data, because no data point will occur more than once. Instead, we divide the overall range of values into a number of classes and count the number of observations that fall into each of these classes or intervals. What we do is

calculating number of observation at least 0 but less than 99 000, at least 100 000 but less than 199 000 and so on. We then get a table using content-based method:

Group number	Packet-rate (captures per min.)	Number of frequency
1	0 – 49 000	6
2	50 000 – 99 000	11
3	100 000 – 149 000	256
4	150 000 – 199 000	176
5	200 000 – 249 000	143
6	250 000 – 299 000	75
7	300 000 – 349 000	30
8	350 000 – 399 000	27
9	400 000 – 449 000	50
10	450 000 – 499 000	45
11	500 000 – 549 000	47
12	550 000 – 599 000	41
13	600 000 – 649 000	70
14	650 000 – 699 000	85
15	700 000 – 749 000	64
16	750 000 – 799 000	53
17	800 000 – 849 000	82
18	850 000 – 899 000	74
19	900 000 – 949 000	41
20	950 000 – 999 000	20
21	1 000 000 – 1 049 000	29
22	1 050 000 – 1099 000	15
		1440 = Total

Table 7: Frequency table of WWW traffic intensity, weekdays, using content-based method.

An arrangement like this is called a grouped frequency distribution. It brings out the overall pattern even more clearly – in this case, a bunching of observed values between 100 000 and 249 000 as we expected from the dot-diagram. But it loses information about the individual values observed. For instance from the first row, the 6 fastest packet-rate recorded could be whatever number between 0 and 49 000. Detail has been sacrificed to clarify the overall pattern.

We can bring out the pattern in grouped frequency distribution even more clearly with a histogram. In the histogram below, the group 12 has twice as many observations as group 17 – and so its block is twice as big. The modal group is the one with the largest frequency. In this case it is group 3.

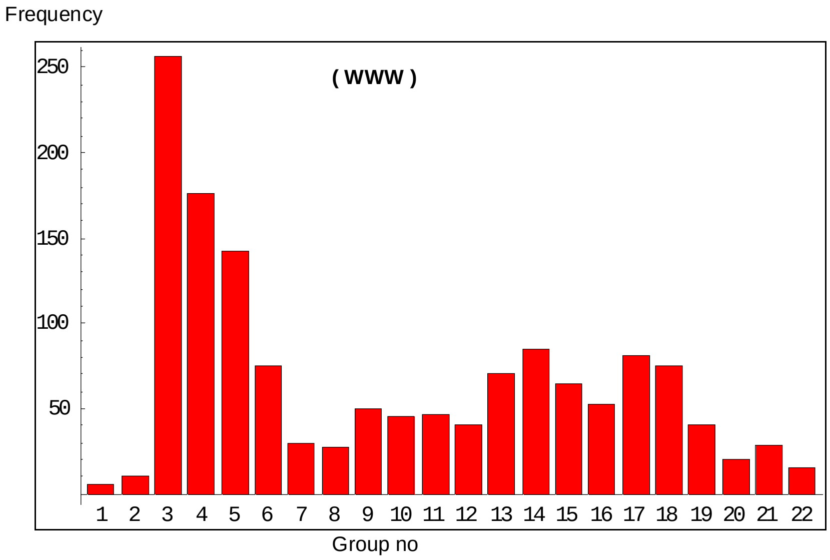


Figure 17: Frequency histogram of WWW traffic, weekdays, using content-based method.

(Notice that we have numbered the horizontal scale with the number of groups corresponding to the packet-rate.)

Another method to represent frequency distribution graphically is by a frequency polygon. As in the histogram, the base line is divided into sections corresponding to the class-interval, but instead of the rectangles, the points of successive class marks are being connected. The frequency polygon is particularly useful when two or more distributions are to be presented for comparison on the same graph.

Frequency polygon for the WWW traffic data

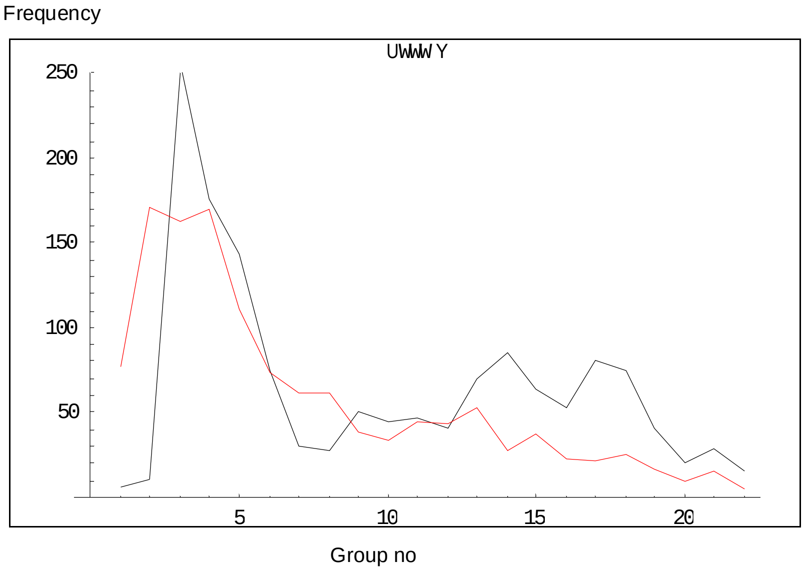


Figure 18: Frequency polygon of WWW traffic, weekdays. Black line indicates using content-based method while red line is port-based.

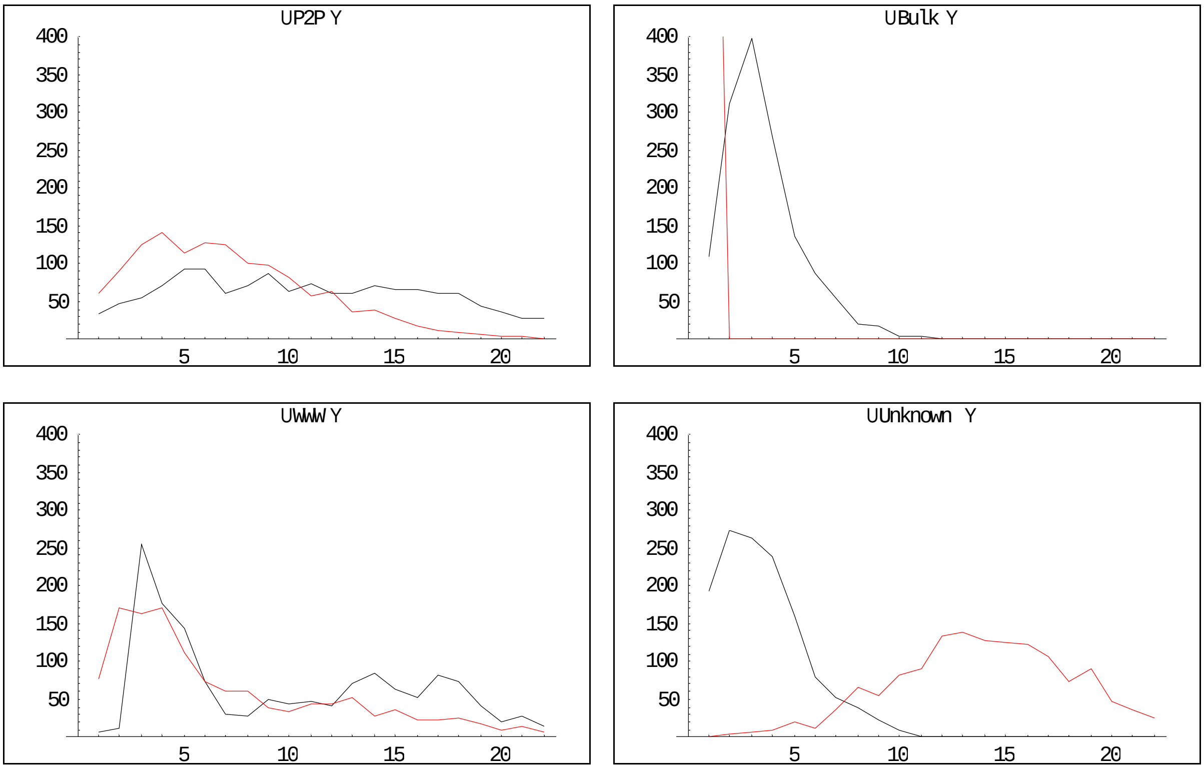


Figure 19: Frequency polygon for Bulk, WWW, P2P and Unknown type, using intervals of 50 000. Black line indicates using content-based method while red line is port-based.

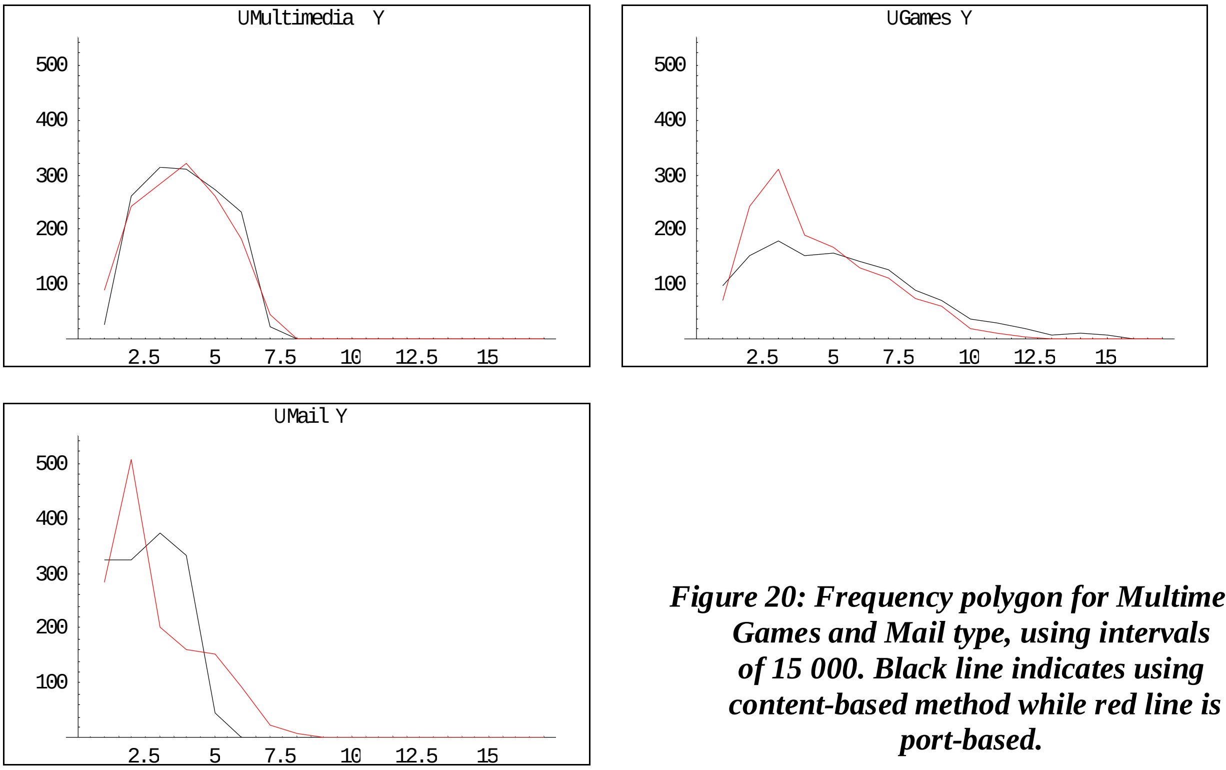


Figure 20: Frequency polygon for Multimedia, Games and Mail type, using intervals of 15 000. Black line indicates using content-based method while red line is port-based.

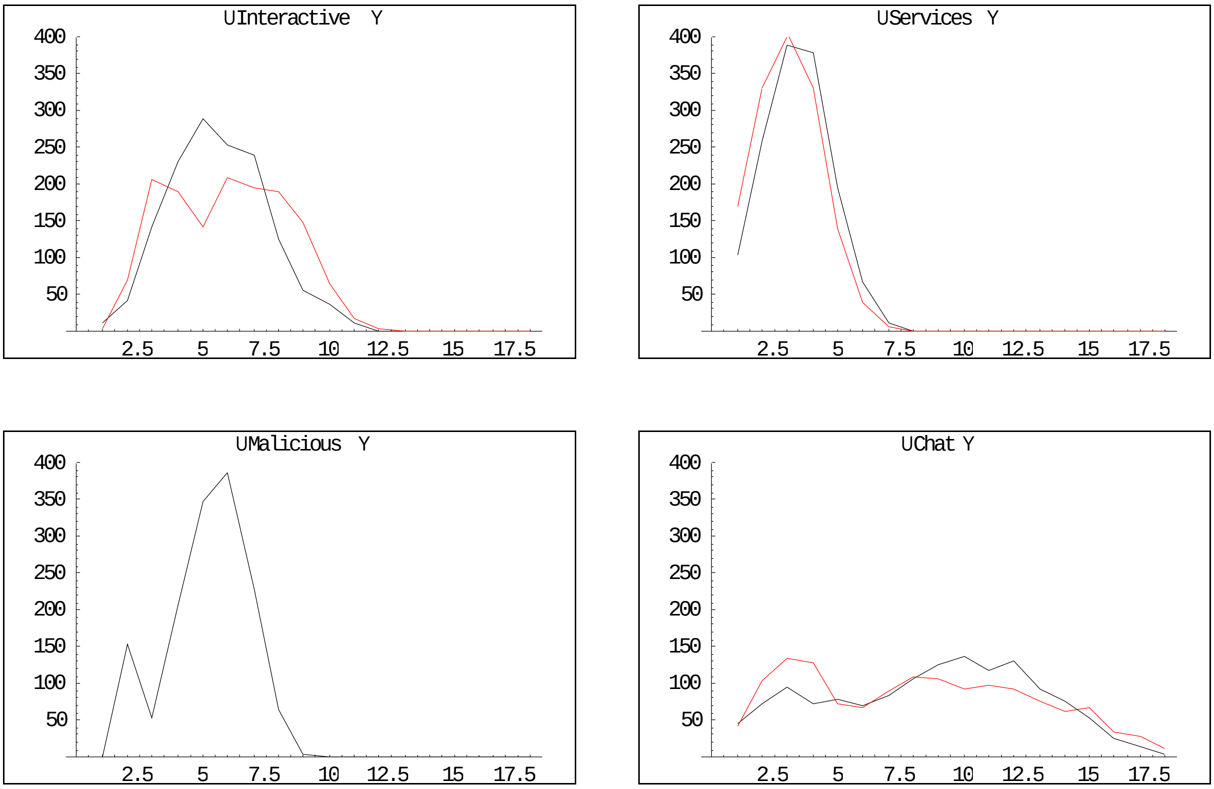


Figure 21: Frequency polygon for Interactive, Services, Malicious and Chat type, using intervals of 4 000. Black line indicates using content-based method while red line is port-based.

Also notices that there are no malicious traffic recorded using port-based method and therefore no red line.

The total traffic

Finally we will have a look at the total traffic using content-based method:

No	Classification type	Amount of packets
1	SERVICES	25 404 731
2	INTERACTIVE	35 215 476
3	MAIL	85 254 588
4	FTP	177 122 533
5	WWW	661 334 866
6	P2P	770 155 533
7	GAMES	162 449 941
8	UNKNOWN	142 384 598
9	MULTIMEDIA	120 031 153
10	CHAT	64 153 841
11	MALICIOUS	34 041 540

2 277 548 800 = Total

Table 8: Frequency table of all services traffic intensity, weekdays, using content-based method.

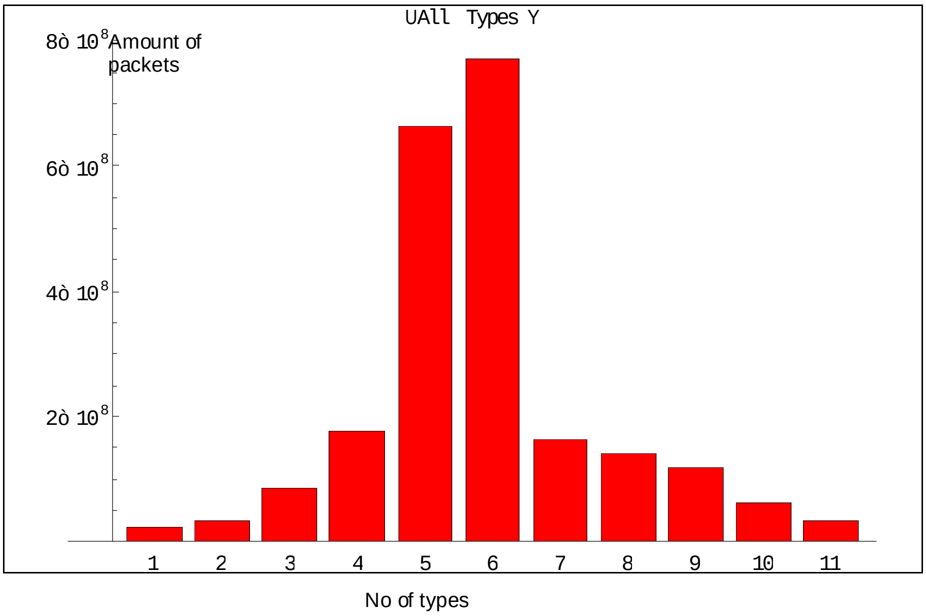


Figure 22: Frequency table of WWW traffic, weekdays, using content-based method.

By convention, we read off the types frequencies by looking at the *height* of the blocks. Another alternative is to let the area of each block stand for the frequency *as a proportion of the total sample size* (an idea which corresponds roughly to *probability*).

Relative Frequency

Relative frequency of a class is defined as:

$$\frac{\text{Frequency of the Class}}{\text{Total Frequency}}$$

If the frequencies are changed to relative frequencies, then a relative frequency histogram, a relative frequency polygon and a relative frequency curve can similarly be constructed.

Relative frequency curve can be considered as probability curve if the total area under the curve be set to 1. Hence the area under the relative frequency curve between a and b is the probability between interval a and b.

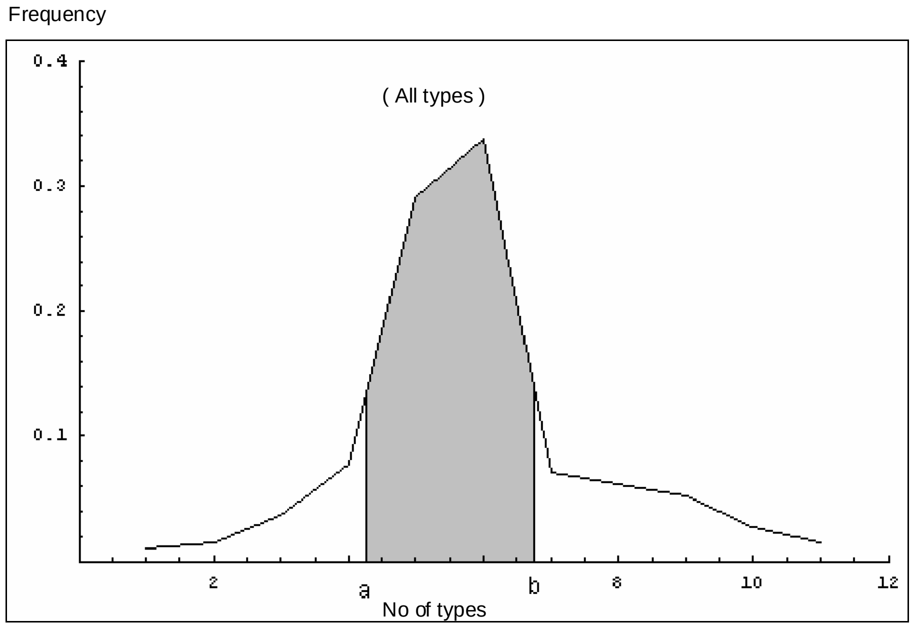


Figure 23: Frequency table of WWW traffic, weekdays, using content-based method.

The total area of a graph of this type is 1. This curve can be described by a function of x, which is known as the probability density function (PDF). The total area under the graph of a PDF is 1.

When we take a measurement in a scientific experiment we sample a continuous variable. If this statement appears surprising, reflect that our measurements will vary around the true value of the quantity we are measuring, and will take on different values with different probabilities: precisely the description of a variable.
The PDF's of experimental measurements are almost all "bell-shaped" like that of the Figure 23. The reasons for this are very deep, but we can assume it in the vast majority of cases. Probability distributions with these "bell-shaped" PDF's are called normal distributions.

So what have we noticed so far about summarizing the results obtained from the experiment? We have seen that a set of raw data, the untreated figures, can be rather obscure. The next step we did was to rearranging them in order of intervals. Grouping may also help emphasize any pattern within the distribution. And diagrams give a better idea of the shape of the distribution than figures alone.

Also, we can begin to look for figures (like average, or the range) that quantify important features of the distribution. In fact, to describe a distribution statistically, or to use it in making inferences or predictions, we must have such figures. One of the most important is central tendency (or average). By central tendency we mean the tendency of the observations to centre around a particular value (or pile up in a particular category) rather than spread themselves evenly across the range or among the available categories. There are 3 types of central tendency: the mode, mean and median. Which one is used will depend on the type of variable. We have already noticed how the mode works.

5.3.2 Central tendency

When we work with numerical data, it seems apparent that in most set of data there is a tendency for the observed values to group themselves about some interior values - some central values seem to be the characteristics of the data. This phenomenon is referred to as central tendency. For a given set of data, the measure of location we use depends on what we mean by middle; different definitions give rise to different measures. We shall consider some more commonly used measures, namely arithmetic mean, median and mode. Ours formulas in finding these values depends on grouped data.

Arithmetic Mean
The arithmetic mean, μ, or simply called mean, is obtained by adding together all of the measurements and dividing by the total number of measurements taken. Mathematically it is given as

$$\mu = \frac{\sum f_i \cdot x_i}{\sum f_i}$$

Where for grouped data: f_i - is the frequency in the ith class,
x_i - is the class mark in the ith class;

Arithmetic mean can be used to calculate any numerical data and it is always unique. It is obvious that extreme values affect the mean.

Median
Median is defined as the middle item of all given observations arranged in order. For grouped data, the median can be found by first identify the class containing the median, then apply the following formula:

$$median = l_1 + \frac{\frac{n}{2} - C}{f_m}(l_2 - l_1)$$

where:
l_1 is the lower class boundary of the median class;
n is the total frequency;
C is the cumulative frequency just before the median class;
f_m is the frequency of the median;
l_2 is the upper class boundary containing the median.

It is obvious that the median is affected by the total number of data but is independent of extreme values. However if the data is ungrouped and numerous, finding the median is tedious. Note that median may be applied in qualitative data if they can be ranked.

Quartiles

Quartiles are the most commonly used values of position which divides distribution into four equal parts such that 25% of the data are ≤ Q1; 50% of the data are ≤ Q2; 75% of the data are ≤ Q3. The first quarter is conventionally denoted as Q1, while the second and third quarters grouped together is Q2 and the last quarter is Q3. Note that Q2 includes the median, contains half of the frequency and excludes extreme values. It is also denoted the value (Q3 - Q1) / 2 as the Quartile Deviation, QD, or the semi-interquartile range.

The five-number summary and boxplots

The five-number summary of a set of observations consists of the smallest observation, the first quartile, the median, the third quartile, and the largest oberservation, written in order from smallest to largest. In symbols, the five-number summary is

Minimum Q1 Median Q3 Maximum

These five numbers offer a reasonable complete description of center and spread.

The five-number summaries:

	Bulk		WWW		P2P		Unknown	
Method	Port-based	Content-based	Port-based	Content-based	Port-based	Content-based	Port-based	Content-based
Median	14441	161307	294764	530878	312667	680724	734642	149307
Q1	3260	125551	156254	182665	99662	199865	441699	110407
Min	411	3544	17478	19602	21988	20741	23477	3095
Max	58602	537665	934896	1199534	914554	1102374	1501447	510477
Q3	26537	265611	546244	736651	407464	806557	1025547	212226

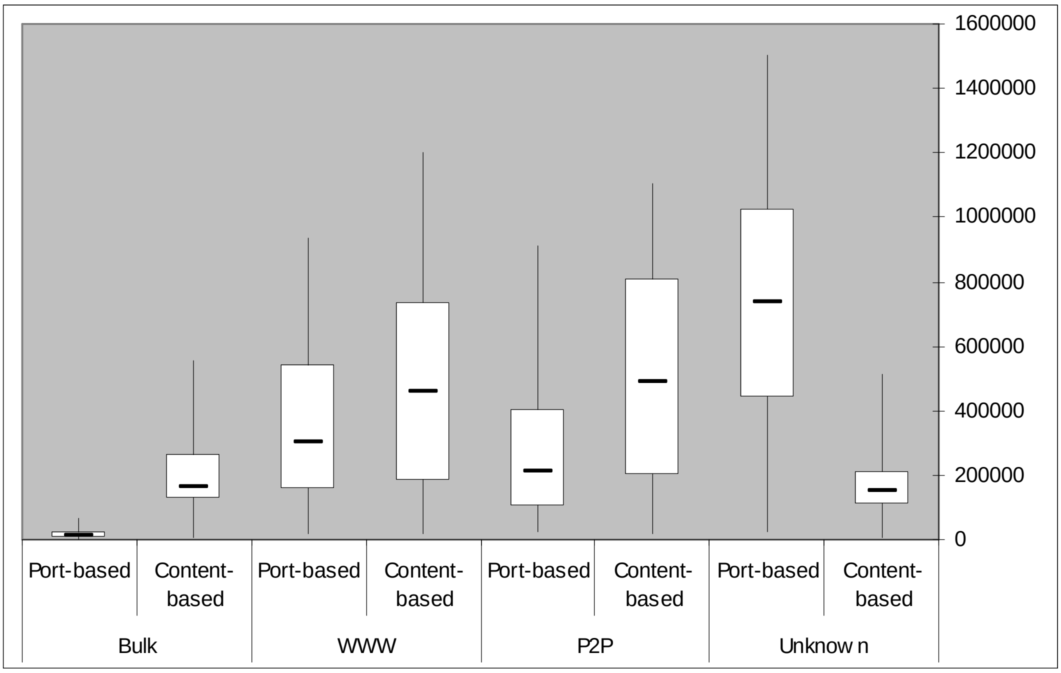


Figure 24: Boxplots of Bulk, WWW, P2P and Unknown traffic, weekdays, both port-based and content-based method.

The candlestick lines show minimum and maximum amount of traffic seen in 24 hours, the bottom and top of the box show the 25th and 75th percentiles, and the line inside the box shows the median value.

	Interactive		Services		Malicious		Chat	
Method	Port-based	Content-based	Port-based	Content-based	Port-based	Content-based	Port-based	Content-based
Median	24341	23043	11277	12521		22351	43978	45043
Q1	19000	19055	4894	5147		10144	38780	40122
Min	3500	3500	478	524		1344	1978	2144
Max	51000	51000	31411	31778		60477	70544	70244
Q3	36000	35149	14789	15144		34334	54599	55203

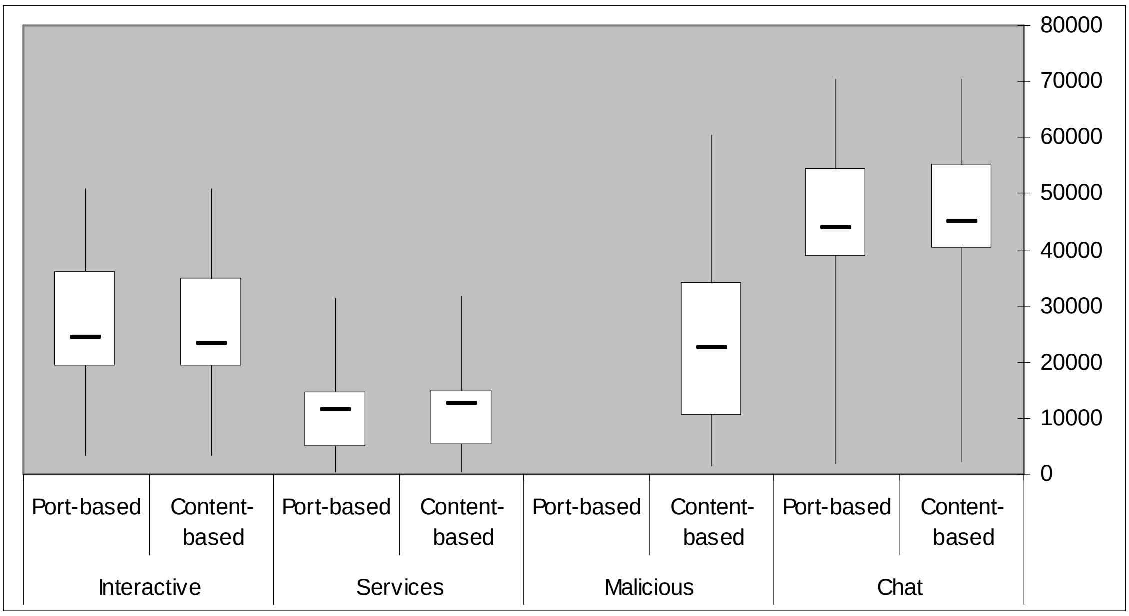


Figure 25: Boxplots of Interactive, Services, Malicious and Chat traffic, weekdays, both port-based and content-based method.

	Mail		Games		Multimedia	
Method	Port-based	Content-based	Port-based	Content-based	Port-based	Content-based
Median	37021	37607	67044	811307	56713	60982
Q1	42495	43456	61542	65551	37547	34168
Min	4929	4854	18514	28415	3471	1514
Max	146846	147485	245149	255665	139657	154698
Q3	91465	91524	121568	135611	74358	84195

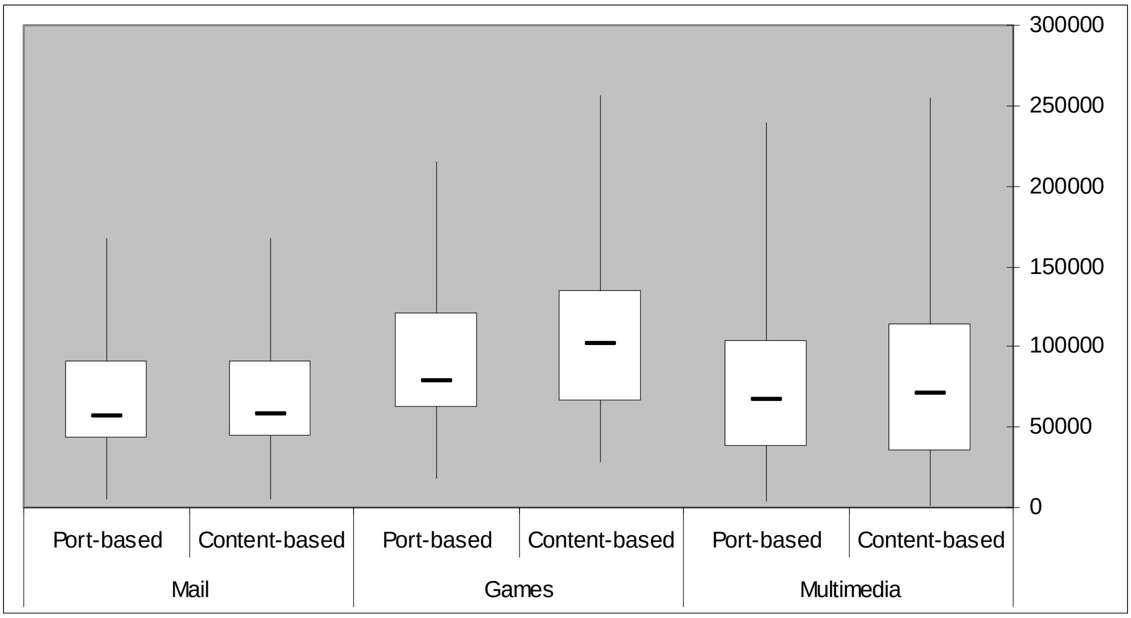


Figure 26: Boxplots of Mail, Games and Multimedia traffic, weekdays, both port-based and content-based method.

5.3.4 Errors

When we record measurements in an experiment we introduce errors. Errors arise from our inaccurate reading of scales, from inaccurate measuring instruments and from imperfect equipment. No experiment is error-free. We can distinguish between two sorts of error:

- Random errors: these have no pattern so we can only strive to minimise them, by careful practice, and to quantify how they affect our confidence in our results.
- Systematic errors: these do have a pattern, for example the times taken from a timer that is simply going too fast.

We are more concerned here with random errors, and how we can quantify their effect on experimental results.

One way to handle errors in experiments is to calculate the “worst case”. If the error in a measurement is ± 5 packets - say we measured 1500 packets but accept that the measurement is somewhere between 1495 packets and 1505 packets - then we can say that the error in subtracting (or adding) two such measurements is at worst ± 10 packets.

But it is usually extremely difficult in practice to estimate the size of “worst case” errors. And worst case analysis does not give a very realistic estimate of how large the error is likely to be, especially if we are combining the results of several measurements. If, for example, we are adding up 100 measurements, each of which has error ± 5 packets, we know that the worst case is for the result to have an error of ± 500 packets. But that would arise only if every one of the 100 measurements was 5 packets too big (or too small). This is extremely unlikely unless the errors are systematic ones.

We are going to derive an analysis of errors that, while it has little to say about “worst cases”, tells us much more about the likely size of errors we might have to deal with.

Two things have a bearing on experimental measurements: the true value of the quantity, and the measuring accuracy. The former determines the position of the bell curve’s maximum point, and is known as the mean of the distribution, μ. The latter determines the “spread” of the bell curve, and is conventionally measured as a quantity called standard deviation, σ . Standard deviation is a more difficult idea than mean, but it is roughly equivalent to “average error size”. A normal distribution is completely determined by its mean and standard deviation.

Variance and Standard Deviation

The variance and standard deviation are two very popular measures of variation. Their formulations are categorized into whether to evaluate from a population or from a sample.

The population variance, σ^2, is the mean of the square of all deviations from the mean. Mathematically it is given as:

$$\frac{\sum f_i (x_i - \mu)^2}{\sum f_i}$$

where: f_i is the frequency of the ith item;
x_i is the value of the ith item or class mark;
μ is the population arithmetic mean.

The population standard deviation σ is defined as $\sigma = \sqrt{\sigma^2}$.

The sample variance, denoted as s^2 gives:

$$\frac{\sum f_i (x_i - \bar{x})^2}{(\sum f_i) - 1}$$

where: f_i is the frequency of the ith item;
x_i is the value of the ith item or class mark;
$\bar{x}$ is the sample arithmetic mean.

The sample standard deviation, s, is defined as s = $\sqrt{s^2}$.

In this experiment we consider the question: given a set of experimental measurements, how may we estimate the true value of the quantity (the mean of the probability distribution) and the average error size (the standard deviation)?
As you may have already worked out for yourself, a good estimate of the true value — the *mean* of the probability distribution — is provided by taking a simple average (also called the **mean**) of the sample itself. Given a set of numbers *x*1, *x*2, . . . *xN*, their mean, *x* , is given by the equation

We can sum up by saying that *the sample mean is an* **estimate** *of the distribution mean*: in other words, *x* is an estimate of μ.
Standard deviations present a slightly harder problem. One might think that a good estimate for σ would be provided by calculating the square root of the average of the squares of the differences from *x*: in other words, by calculating

In fact, though, this figure turns out to be systematically biased: it provides an *underestimate* of the true standard deviation. A fairly subtle argument shows that we can remove this systematic bias from our estimate simply by dividing by *N* –1 instead of *N* (though for this to work, we need to be sure that the errors are *non-systematic*).

This gives us the **sample standard deviation**, *s*:

In summary, *the sample standard deviation is an estimate for the distribution standard deviation*: *s* is an estimate for σ.

Table over mean and standard deviation of Bulk, WWW, P2P and Unknown, weekdays.

	Bulk		WWW		P2P		Unknown	
Method	Port-based	Content-based	Port-based	Content-based	Port-based	Content-based	Port-based	Content-based
Mean	13799	158307	297563	482307	302384	658900	727183	146390
StDev	11582	91813	310684	306883	238126	436205	256029	119524

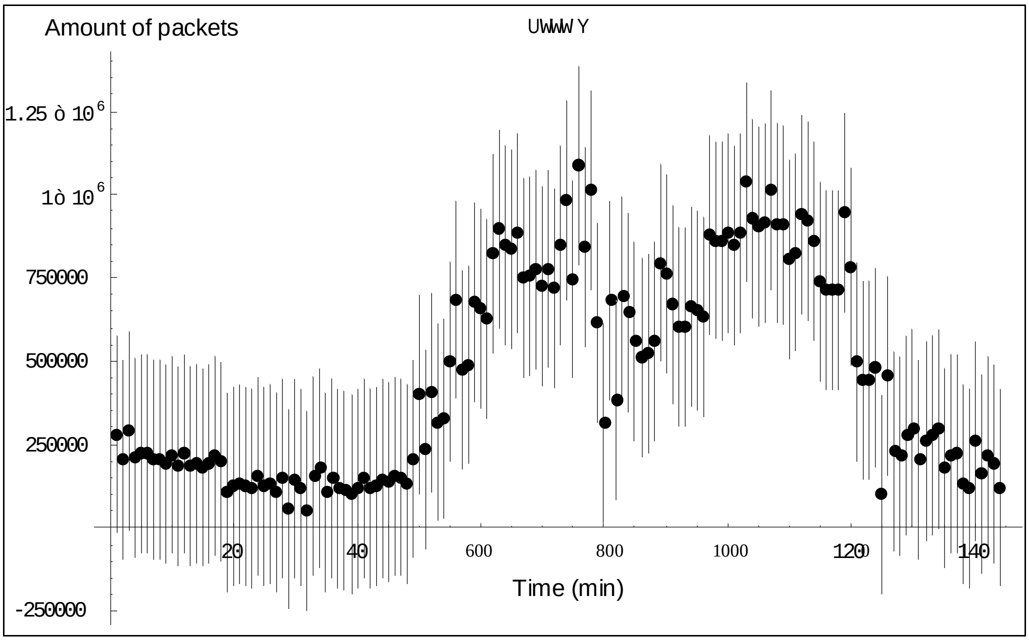


Figure 28: Time series of WWW traffic measurement, error bars are used to represent the width of the scatter. The result is a plot of the observations $\pm\sigma$ ***.***

6 Summary and future work

Motivated by the need for more accurate identification techniques for network applications, we presented a framework for traffic characterization in the presence of packet payload. We laid out the principles for the correct classification of network traffic. Such principles are captured by several individual building blocks that, if applied iteratively, can provide sufficient confidence in the identity of the causal application. Our technique is not automated due to the fact that a particular Internet flow could satisfy more than one classification criterion or it could belong to an emerging application having behaviour that is not yet common knowledge.

We collected a full payload packet traces from a Uninett Gigabit backbone link and compared the results of our content-based scheme against the port-based classification technique. We showed that classifying traffic based on the usage of well-known ports leads to a high amount of the overall traffic being unknown and a small amount of traffic being misclassified. We quantified these inaccuracies for the analysed packet trace.

We then presented an analysis of the accuracy-gain as a function of the complexity introduced by the different classification sub-methods. Our results show that simple port-based classification can correctly identify approximately 55% of the overall traffic. Application of increasingly complex mechanisms can approach 94% accuracy with great benefits gained even through the analysis of up to 1024 Bytes of a traffic flow.

Our work should be viewed as being at an early stage and the avenues for future research are multiple. One of the fundamental questions that need investigation is how such a system could be implemented for real-time operation. We would argue that an adapted version of the architecture described in [33], which currently performs on-line flow analysis as part of its protocol-parsing and feature-compression, would be a suitable system. Such an architecture overcomes the (potential) over-load of a single monitor by employing a method work-load sharing among multiple nodes. This technique incorporates dynamic load-distribution and assumes that a single flow will not overwhelm a single monitoring node. In our experience such a limitation is sufficiently exible as to not be concerning.

We clearly need to apply our technique to other Internet locations. We need to identify how applicable our techniques are for other mixes of user traffic and when our monitoring is subject to other limitations. Examples of such limitations include having access to only unidirectional traffic or to a sample of the data.
Both these situations are common for ISP core networks and for multi-homed sites. We already identify that the first phase of identification and culling of simplex flows would not be possible if the only data available corresponded to a single link direction.

We emphasise that application identification from traffic data is not an easy task. Simple signature matching may not prove adequate in cases where multiple classification criteria seem to be satisfied simultaneously. Validation of the candidate application for a traffic flow in an automated fashion is an open issue.

Further research needs to be carried out in this direction. Moreover, we envision that as new applications appear in the Internet there will always be cases when manual intervention will be required in order to gain understanding of its nature.

Lastly, in future work we intend to address the issue of how much information needs to be accessible by a traffic classifier for the identification of different network applications. Our study has shown that in certain cases one may need access to the entire flow payload in order to arrive to the correct causal application. Nonetheless, if system limitations dictate an upper bound on the captured information, then the knowledge of the application(s) that will evade identification is essential.

References

[1] http://dag.cs.waikato.ac.nz/networkMCards.htm

[2] “DAG 4.2GE INSTALLATION GUIDE”, Endace Measurement Systems

[3] RFC 1413 Identification Protocol. http://www.ietf.org/rfc/rfc1413.txt

[4] IANA, Internet Assigned Numbers Authority (IANA). http://www.iana.org/assignments/port-numbers.

[5] K. Claffy. ”Internet traffic characterization”, PhD thesis, UC San Diego, 1994.

[6] B. Krishnamurthy and J. Rexford. ”Web Protocols and Practice”, Chapter 10: Web Workload Characterization. Addison-Wesley, 2001.

[7] J. E. Pitkow. Summary of WWWcharacterizations. W3J, 2:3–13, 1999.

[8] P. Barford and M. Crovella. ”Generating Representative Web Workloads for Network and Server Performance Evaluation”, In Proceedings of ACM Sigmetrics, June 1998

[9] J. Almeida, J. Krueger, D. Eager, and M. Vernon. ”Analysis of educational media server workloads”, In Proc. Inter. Workshop on Network and Operating System Support for Digital Audio and Video, June 2001.

[10] M. Chesire, A. Wolman, G. M. Voelker, and H. M. Levy. “Measurement and analysis of a streaming media workload”, In Proc. USENIX Symposium on Internet Technologies and Systems, March 2001.

[11] J. van der Merwe, S. Sen, and C. Kalmanek. ”Streaming video traffic : Characterization and network impact”, In 7th International Web Content Caching and Distribution workshop (WCW), Boulder, Colorado, August 14th-16th 2002.

[12] V. Paxson. ”Empirically derived analytic models of wide-area TCP connections”, IEEE/ACM Transactions on Networking, 2(4):316–336, 1994.

[13] V. Paxson and S. Floyd. ”Wide-area traffic: The failure of Poisson modeling”, IEEE/ACM Transactions on Networking, 3(3):226–244, June 1995.

[14] C. Dewes, A. Wichmann, and A. Feldmann. ”An analysis of Internet chat systems”, In Proceedings of ACM SIGCOMM Internet Measurement Conference,
Oct 2003.

[15] P. Barford and D. Plonka. ”Characteristics of Network Traffic Flow Anomalies”, In Proceedings of ACM SIGCOMM Internet Measurement Workshop, Oct 2001.

[16] P. Barford, J. Kline, D. Plonka, and A. Ron. ”A Signal Analysis of Network Traffic Anomalies”, In Proceedings of ACM SIGCOMM Internet Measurement Workshop, Nov 2002.

[17] Y. Zhang and V. Paxson. ”Detecting backdoors”, In Proc. USENIX, Denver, Colorado, USA, 2000.

[18] D. Moore, G. Voelker, and S. Savage. “Inferring Internet Denial of Service Activity”, In Proc. of the USENIX Security Symposium, Washington D.C., August 2001. Available at http://www.cs.ucsd.edu/˜savage/papers/UsenixSec01.pdf.

[19] Jan Coppens, Steven Van den Berghe, Herbert Bos, Evangelos Markatos, Filip De Turck, Arne Øslebø and Sven Ubik. ”SCAMPI: A Scalable and Programmable Architecture for Monitoring Gigabit Networks”, Proceedings of E2EMON Workshop, Belfast, 7 September 2003. Available at http://www.ist-scampi.org/publications/

[20] Luca Deri. “Passively Monitoring Networks at Gigabit Speeds using Commodity Hardware and Open Source Software”, PAM 2003, April 2003.
Available at http://www.ist-scampi.org/publications/

[21] J. Coppens, E.P. Markatos, J. Novotny, M. Polychronakis, V. Smotlacha and S. Ubik. “SCAMPI - A Scaleable Monitoring Platform for the Internet”, Proceedings of the 2nd International Workshop on Inter-Domain Performance and Simulation (IPS 2004), Budapest, Hungary, 22-23 March 2004.
Available at http://www.ist-scampi.org/publications/

[22] NLANR/MOAT Active Measurement Program, http://amp.nlanr.net/

[23] J. Apisdorf, K. Claffy and K. Thompson. “OC3MON: Flexible, Affordable, High-Performance Statistics Collection”, Proceedings of INET ’97, Kuala Lumpur, Malaysia, June 1997.
http://www.isoc.org/isoc/whatis/conferences/inet/97/proceedings/F1/F1_2.HTM

[24] T. Oetiker, “MRTG - Multi Router Traffic Grapher”.
http://people.ee.ethz.ch/~oetiker/webtools/mrtg/paper/

[25] Cisco Systems, “NetFlow Services and Applications”, White Paper, June 2002.
http://www.cisco.com/warp/public/cc/pd/iosw/ioft/neflct/tech/napps_wp.pdf

[26] Amdahl, G.M., “Validity of single-processor approach to achieving large-scale computing capability”, Proceedings of AFIPS Conference, Reston, VA. 1967. pp. 483-485.

[27] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, W. Wiess, “An Architecture for Differentiated Services”, RFC 2475.

[28] A. Gerber, J. Houle, H. Nguyen, M. Roughan, and S. Sen, ”P2P: The Gorilla in the Cable”, In National Cable & Telecommunications Association (NCTA) 2003 National Show, Chicago, IL, June 2003.

[29] S. Saroiu, K. P. Gummadi, R. J. Dunn, S. D. Gribble, and H. M. Levy, “An Analysis of Internet Content Delivery Systems”, In Proceedings of the 5th Symposium on Operating Systems Design and Implementation, 2002.

[30] K. P. Gummadi, R. J. Dunn, S. Saroiu, S. D. Gribble, H. M. Levy, and J. Zahorjan, “Measurement, modeling, and analysis of a peer-to-peer file-sharing workload”, In Proceedings of the 19th ACM Symposium on Operating Systems Principles (SOSP-19), October 2003.

[31] A. Feldmann, ”BLT: Bi-layer tracing of HTTP and TCP/IP”, WWW9 / Computer Networks, 2000.

[32] V. Paxson, ”Empirically derived analytic models of wide-area TCP connections”, IEEE/ACM Trans. Netw., 1994.

[33] W. Moore, A. Hall, J. Kreibich, C. Harris and E. Pratt, “Architecture of a Network Monitor. In: Passive & Active Measurement”, Workshop 2003 (PAM2003).

[34] W. Moore, D. Keys, K. Koga and R. Lagache, “CoralReef software suite as a tool for system and network administrators”, In: Proceedings of the LISA 2001, 15th Systems Administration Conference, 2001.

All reference last checked 22th May 2005

Appendices

A.1 Running the DAG data capture software

The various tools used for data capture are in the *tools* sub-directory.

To make a trace we use the dagsnap tool:

```
dagsnap -d <device> -o <filename> -s <runtime in seconds>
```

On scampi1 there is only one card, /dev/dag0, with two interfaces so we capture both directions at the same time.
To capture a 5 minute trace we would write:

```
scampi1:~$ /root/dag/tools/dagsnap -d /dev/dag0 -o trace.dag -s 300
```

The traces are in erf format, the native format for dag. The program to convert from erf to pcap:

```
dagconvert -T:pcap -i <erf file> -o <pcap file>
```

To check which interfaces are enable with the command dagfour:

```
scampi1:~$ /root/dag/tools/dagfour
linkA   noreset nonic nofcl noeql enablea
linkB   noreset nonic nofcl noeql enableb
packet  varlen slen=100
packetA drop=3454682
packetB drop=3366184
pci     66MHz 64-bit rxonly
```

Here we can see that both interfaces A and B are indicated as “enable”, i.e. packets are captured from both interfaces. To switch on or off on an interface we used the command

```
dagfour disablea or dagfour disableb
```

Her ser du at både interface A og B er markert som "enable", noe som vil si at pakker blir fanget fra begge to. For å skru av eller på et interface kan du f.eks. skrive "dagfour disablea" eller "dagfour enablea".

```
phonghp@scampi1:~$ more cronmaleDAG
#! /usr/bin/perl

#ARGV[0]=sted

$var="@";
$var2=phonghp.$var.cia;

#`mkdir /traces/phonghp/$ARGV[0]`;
#`ssh $var2.uninett.no mkdir $ARGV[0]`;
`rm /home/phonghp/cronfile`;
`touch /home/phonghp/cronfile`;

open(ADD, ">>cronfile");
for ($i=0; $i<24; $i++)
{
    print "1. løkka\n";
    for ($j=10; $j<60; $j+=30)
    {
       print "2. løkke\n";
        $filname = traces.$i.$j;
        $tid=$i-1;
        $gammelfil = traces.$tid.$j;
        print ADD "$j $i * * * /root/dag-2.4.14/tools/dagsnap -d
/dev/dag0 -o /t
races/phonghp/$ARGV[0]/$filname.dag -s 240\n";
        $nytid = $j+5;
        print ADD "$nytid $i * * * /root/dag-2.4.14/utils/dagconvert -
T:pcap -i
/traces/phonghp/$ARGV[0]/$filname.dag -
o/traces/phonghp/$ARGV[0]/$filname.pcap\n
";
        $nytid2 = $j+10;
        print ADD "$nytid2 $i * * * /bin/rm
/traces/phonghp/$ARGV[0]/$filname.da
g\n";
        print ADD "$nytid2 $i * * * /usr/bin/scp
/traces/phonghp/$ARGV[0]/$filna
me.pcap $var2.uninett.no:$ARGV[0]\n";
        print ADD "$nytid2 $i * * * /bin/rm
/traces/phonghp/$ARGV[0]/$gammelfil.
pcap\n";
        print ADD "\n";

    }
}
close(ADD);
```

A.2 Scripts

```
#! /bin/bash
# run.bash

/usr/sbin/tcpdump -n -nn -r $1 | /home/phong/perl/script4

#/usr/sbin/tcpdump -n -nn host 193.69.165.21 -C 66289411 -r
/home/phong/splitac | /home/phong/perl/script3
```

```
[phong@medusa]$ more script2
#! /usr/bin/perl

#lists all ports

#list of ports
#http://www.chebucto.ns.ca/~rakerman/port-table.html

$firstT=1;
$pktCounter=0;

while ($line = <STDIN>)
{
    @array = split(' ',$line);
    if ($firstT == 1)
    {
        $startTime = $array[0];
        $firstT = 0;
    }
    else
    {
        $endTime = $array[0];
    }

    $port = "$4" if ($array[1] =~ /(.*)\.(.*)\.(.*)\.(.*)/);

    $port_in = "$4" if ($array[3] =~ /(.*)\.(.*)\.(.*)\.(.*):(.*)/);

    $antall{$port}++;
    $antall_in{$port_in}++;

    $pktCounter++;



}
#close(FIL);

print "***************************************************\n";
print "***************************************************\n";

#$antall{$port} = sort $antall{$port};
print "Number of packets $pktCounter\n";
```

```
sub hashValueAscendingNum
{
   $antall{$a} <=> $antall{$b};
}

print "Outcoming traffic\n";

foreach $port_in (sort hashValueAscendingNum (keys(%antall_in)))
{
   print "$port_in: $antall{$port_in}\n";
}

print "\n";
print "\n";
print "Incoming traffic\n";

foreach $port (sort hashValueAscendingNum (keys(%antall)))
{
   print "$port: $antall{$port}\n"; # \t\t $port_in:
$antall_in{$port_in}\n";
}

print "\n";
print "\n";

print "Start time: $startTime\n";
print "End time: $endTime\n";

print "****************************************************\n";
print "****************************************************\n";
```

```
[phong@medusa perl]$ more script4
#! /usr/bin/perl

#leser fra output'en fra tcpdump
#skriver output til skjerm + filer

#outcoming traffic

$bulk=0; #ftp_data=20 and ftp=21
$interactive=0; #ssh=22, telnet=23
$mail=0; #smpt=25, pop2and3=109 and 110, imap=143
$services=0; #dns=53, ldap=389, ntp=123, auth=113
$www=0; #http=80, https=443, http_alt=8080
$multimedia=0; #rtsp=554, real=7070
$p2p=0; #kaaza=1214, directConnect=412, eDon=4661-4672,
        #napster=6699-6701, bittorent=6881-6889,
        #gnutella=6346-6347,
$chat=0; #yahoo=5050, aol=5190, msn=1863, irc=2337
$games=0; #halflife=27005-27030, warcraft=6112-6119


$snmp=0; #161 NY

#incoming traffic


#protocols
$icmp=0;
$udp=0;
$tcp=0;

$teller=0;

$dato=`date '+%d%m'`;
$path="/home/phong/output/$dato";


$firstT=1;

while ($line = <STDIN>)
{
    $teller++;
    @array = split(' ',$line);
    if ($firstT == 1)
    {
        $startTime = $array[0];
        $firstT = 0;
    }
    else
    {
      $endTime = $array[0];
    }

    $port = "$4" if ($array[1] =~ /(.*)\.(.*)\.(.*)\.(.*)/);

    $port_in = "$4" if ($array[3] =~ /(.*)\.(.*)\.(.*)\.(.*):(.*)/);
    $proto = $array[4];
```

```
#services
    if(($port == 20) or ($port == 21))
    {
       $bulk++;
    }
    elsif(($port == 22) or ($port == 23))
    {
        $interactive++;
    }
    elsif(($port == 25) or ($port == 109) or ($port == 110) or
         ($port == 143))
    {
        $mail++;
    }
    elsif(($port == 53) or ($port == 389) or ($port == 123) or
         ($port == 113))
    {
        $services++;
    }
    elsif(($port == 80) or ($port == 443) or ($port == 8080))
    {
        $www++;
    }
    elsif(($port == 554) or ($port == 7070))
    {
        $multimedia++;
    }
    elsif((($port == 412) or ($port == 1214) or
         (($port > 4660) and ($port < 4673)) or
         (($port > 6698) and ($port < 6702)) or
         (($port > 6880) and ($port < 6890)) or
         (($port > 6345) and ($port < 6348)))
    {
        $p2p++;
    }
    elsif(($port == 5050) or ($port == 5190) or ($port == 1863) or
         ($port == 6901) or ($port == 2337))
    {
        $chat++;
    }
    elsif((($port > 27004) and ($port < 27031)) or
         ((port > 6112) and ($port < 6119)))
    {
        $games++;
    }

#protocols
    if($proto =~ /icmp/)
    {
        $icmp++;
    }
    elsif($proto =~ /udp/)
    {
        $udp++;
    }
    elsif($proto =~ /S|F|R|P|./)
```

```
        {
            $tcp++;
        }

    }
    close(FIL);

    print "***************************************************\n";
    print "***************************************************\n";

    #making hour to 60 minutes
    @array5 = split(':',$sTime);
    if($array5[0] != 0) #to avoid dividing on zero
    {
        $startTime = ($array5[0]*60)+($array5[1]);
        print "IKKE NULL: $startTime\n";
    }
    else
    {
        $startTime = $array5[1];
        print "NULL: $startTime\n";
    }

    print " Numbers of packets $teller\n";
    print "Andre løkke: $startTime\n";

    print "Outcoming traffic\n";
    print "BULK: $bulk \t Percent: $bulk/$teller*100\n";
    print "INTERACTIVE: $interactive \t Percent: /
    $interactive/$teller*100\n";
    print "MAIL: $mail \t Percent: $mail/$teller*100\n";
    print "SERVICES: $services \t Percent: $services/$teller*100\n";
    print "WWW: $www \t Percent: $www/$teller*100\n";
    print "P2P: $p2p \t Percent: $p2p/$teller*100\n";
    print "GAMES: $games \t Percent: $games/$teller*100\n";
    print "MULTIMEDIA: $multimedia \t Percent: /
    $multimedia/$teller*100\n";
    print "\n";

    print "Protocols\n";
    print "ICMP: $icmp\n";
    print "UDP: $udp\n";
    print "TCP: $tcp\n";
    print "\n";

    print "Start time: $startTime\n";
    print "End time: $endTime\n";
```

A.3 Cluster

```
[phong@medusa]$ more alle.bash
#! /bin/bash

dato=`date '+%d%m'`
path=/home/phong/output/$dato

if [ ! -d $path ]
then
    echo her $path
    mkdir $path
fi


for host in `cat nodes`
do
        #echo "Prosesser på ${host}:"
        # {hots} for at : ikke blir del av variabel

        for fil in $1/*
        do
          nyfil=`basename $fil .pcap`
          #echo $nyfil

          if [ "$host" == "$nyfil" ]
          then
              echo her
              ssh $host /home/phong/perl/run $fil >> $path/$host &

          #else
          #    echo "her\n";
          fi

        done

       # ssh $host ps aux | grep $USER
       # if ["$host" == ""]
done


[phong@medusa phong]$ more nodes
compute-0-0
compute-0-1
compute-0-2
compute-0-3
compute-0-4
compute-0-5
compute-0-6
compute-0-7
compute-0-8
compute-0-9
compute-0-10
compute-0-11
compute-0-12
compute-0-13
```

A.4 Snort

We want to run a binary log file through Snort in sniffer mode to dump the packets to the screen:

```
snort -r /data/traces.pcap –c /local/snort-2.0.2/etc/snort.conf –l ./log
```

We are on high speed network and we want to log the packets into a more compact form for later analysis, therefore we consider logging in binary mode. Binary mode logs the packets in tcpdump format to a single binary file in the logging directory:

Once the packets have been logged to the binary file, we can read the packets back out of the file with sniffer that supports the tcpdump binary format (tcpdump or Ethereal).