BUDAPEST UNIVERSITY OF TECHNOLOGY AND ECONOMICS
DEPT. OF TELECOMMUNICATIONS AND MEDIA INFORMATICS

METHODS FOR EFFICIENT CLASSIFICATION OF NETWORK TRAFFIC

Géza Szabó

Ph.D. Dissertation

Supervised by

Dr. Sándor Molnár PhD
High Speed Networks Laboratory
Dept. of Telecommunications and Media Informatics
Budapest University of Technology and Economics

Budapest, Hungary
2010

Abstract

Accurate classification of traffic flows is an essential step for administrators to detect intrusion or malicious attacks, forbidden applications, or simply new applications which may impact the future provisioning of network resources. In this work four major topics are considered in connection with traffic classification activities.

First, a *novel validation method* is proposed for characterizing the accuracy and completeness of traffic classification algorithms. The main advantages of the new method are that it is based on realistic traffic mixtures, and it enables a highly automated and reliable validation of traffic classification. The validation method is used to create reference traces and the classification performance of the existing traffic classification methods is measured. Using this information a *combined traffic classification method* that includes the advantages of different approaches is introduced, in order to provide a high level of classification completeness and accuracy.

The second part of our work focuses on gaming traffic. Gaming traffic depends on two main factors, the game protocol and the gamers' behavior. By understanding the nature of the impact of the later one *user behavior detection algorithms* are introduced to grab specific events and states from passive traffic measurements. The algorithms focus on the characteristics of the traffic rate, showing what information can be gathered by observing only packet header information.

In the third part of our work, *a novel model and an algorithm* are introduced to extend the Deep Packet Inspection traffic classification method with the analysis of non-fix byte signatures, which are not considered in current methods.

Finally, we focus on the performance of traffic classification methods in terms of speed. To *support high speed traffic classification* the majority of the tasks are reformulated as parallel algorithms and pushed to the Graphical Processing Unit to offload the CPU, which then may serve other processing intensive tasks, e.g., traffic capture. The performance tests of the proposed methods showed that traffic classification is possible up to approximately 6 Gbps with a commodity PC.

Contents

List of Figures

List of Tables

Abbreviations

- 3G – Third Generation
- ASIC – Application-Specific Integrated Circuit
- CPU – Central Processing Unit
- DFA – Deterministic Finite state Automate
- DNS – Domain Name System
- DPI – Deep Packet Inspection
- fBm – fractional Brownian motion
- FPGA – Field-Programmable Gate Array
- FPS – First Person Shooter
- FTP – File Transfer Protocol
- GPU – Graphical Processing Unit
- IDS – Intrusion Detection System
- IPS – Intrusion Prevention System
- HTTP – HyperText Transfer Protocol
- HSL – Hue, Saturation, Lightness
- IP – Internet Protocol
- LRD – Long-Range Dependence
- ML – Machine Learning
- MMORPG – Massively Multiplayer Online Role Playing Game
- MMORTS – Massively Multiplayer Online Real-Time Strategy game
- MTU – Maximum Transmission Unit
- NDIS – Network Driver Interface Specification
- NIC – Network Interface Card
- P2P – Peer to Peer
- QoS – Quality of Service
- RFC – Request For Comments
- RTP – Real-time Transport Protocol
- RTS – Real-Time Strategy game
- RTT -Round Trip Time
- SCP – Secure CoPy
- SIMD – Single Instruction Multiple Data
- SMTP – Simple Mail Transfer Protocol

- SSH – Secure Shell
- SVM – Support Vector Machine
- TCP – Transmission Control Protocol
- VoIP – Voice over IP
- XFA – Extended Finite state Automate

Acknowledgements

I would like to express my gratitude to all those who gave me the possibility to complete this thesis.

I am thankful for the invaluable help of my research supervisor, Dr. Sándor Molnár and all the members of the High Speed Networks Laboratory including members of the Department of Telecommunications and Media Informatics and also the Department of Telecommunications.

I would like to thank the friendly and inspiring atmosphere of all the colleagues in Ericsson TrafficLab, especially for István Szabó with whom we started this work, and the others who I worked with including Dániel Orincsay, András Veres, Szabolcs Malomsoky and István Gódor who all made the rough research way smoother for me.

This is a prominent opportunity to express my thank to my former teachers. Among them, I am particularly grateful to my math and physics teachers in secondary school Attila Horty and Péter Sáró, and my English teachers István Szabó and Márta Merza who made a solid basis for this work.

Most importantly, none of this would have been possible without the love and patience of my family, especially my mother, father and my brother. They have been a constant support of love, concern and strength during all these years.

Introduction

Detailed knowledge about the traffic mixture is essential for network operators and administrators, as it is a key input for numerous network management activities. For example, it can be the input for network planning and capacity provisioning, fine-tuning of charging schemes or security monitoring. Further, a good understanding of the traffic mix and the capability of observing trends in the traffic volume of the different applications can provide important input to network equipment design.

The aim of traffic classification is to find out what type of applications are run by the end users, and what is the share of the traffic generated by the different applications in the total traffic mix. The communication between IP network nodes can be organized into flows and the task of traffic classification is to assign a specific application to each individual flow. A flow is a collection of IP packets sent from a given port at one IP address to a given port at another IP address using a specific protocol. A flow is identified by its five-tuple flow identifier: source IP address, destination IP address, source port, destination port, protocol identifier.

The objective of the dissertation is to design and develop robust and efficient traffic classification tools for IP networks. **Our first goal was to establish a framework to make it possible to compare the traffic classification mechanisms available in literature.** We proposed a method for traffic classification validation. The further goal was to utilize the results of the measurement of classification metrics to propose a new combined method for traffic classification.

During the validation of traffic classification methods, it turned out that the presumptions used in traffic characteristics based classifiers in current literature is obsolete when the identification of currently popular gaming traffic is needed. Most of current internet traffic consists of proprietary Real Time Strategy (RTS) and Massively Multiplayer Online Role Playing Game (MMORPG) traffic beside the former modeled First Person Shooter (FPS) games. FPS traffic consists of fixed sized packets sent in fixed rate. Current popular gaming traffic is special as the traffic characteristics are highly influenced by user interactions. **Our second goal was to analyze the traffic characteristics of these new emerging game types.**

Gaming traffic usually uses proprietary non-documented protocols, or further even

Protocol Header Encryption to make protocol recognition unfeasible with common Deep Packet Inspection (DPI) methods. During our work with gaming traffic, we experienced that in most cases gaming traffic consists of the communication of player related coordinates between peers. We analyzed the possibility of the utilization of this information. **The third goal was to extend current DPI methods to make them also capable of recognizing fields with non-fixed byte signatures in a protocol structure.**

The first part of our work focused on traffic classification accuracy and completeness and not considered the traffic classification speed. However, in most cases this is basis of a go or not go decision whether a method can be applied in a real-time traffic classification system. **The fourth goal was to focus on wire-speed traffic classification and to reformulate traffic classification parts to utilize efficiently massively parallel architectures.**

All the algorithms presented in this work were implemented and used in real network conditions. In Section 2.1 the applied method were *measurement* and implementation of the algorithms to work in real network conditions. The proposed validation method was implemented as a network driver. This made it possible to create measurements in real network conditions. The measurement of the accuracy of the traffic classification engines was possible either by obtaining the original source code of the papers or by implementing ourselves. The proposed classification algorithm was also implemented and used on the reference trace and traces obtained from real networks. Results reported in Section 3 were obtained by *statistical analysis* of active measurements. In Section 4 the model parameters were obtained through the statistical analysis of measured data from real networks. The analysis of the effects of process decomposition was analyzed through custom-built *simulations* implemented in Matlab [39]. In Section 5 the proposed algorithms were implemented on a proper hardware and tested on measurements obtained from real world measurements. The achievable compression ratio was analyzed through custom-built simulations implemented in Matlab [39].

Chapter 1

Related work

Currently, there are a couple of fundamentally different approaches for traffic classification. In this section we browse through the state-of-the-art traffic classification methods. We discuss briefly the accuracy of these methods, which is relevant here, because in most cases a classification method is validated by another classification method ([91], [127], [124]).

The most accurate traffic classification would obviously be complete protocol parsing. However, many protocols are ciphered due to security reasons (SSH [30], SSL [29]). Also some are proprietary, thus there is no public description available (Skype [31], MSN Messenger [17], World of Warcraft [38], etc.). In general, it would be difficult to implement every protocol which can occur in the network. In addition, even simple protocol state tracking can make the method so resource consuming that it becomes practically infeasible.

1.1 Port based classification

In TCP/IP and UDP networks, a port is an endpoint to a logical connection and the way a client program specifies a specific server program on a computer in a network. Some ports have numbers that are preassigned to them by the IANA [12] first in 1994. Port numbers range from 0 to 65536, but only ports numbers 0 to 1024 are reserved for privileged services and designated as well-known ports. The Registered Ports are those from 1024 through 49151 and above this port number are the Dynamic and/or Private Ports.

In the simplest and most common method the traffic classification is based on associating a well-known port number with a given traffic type, e.g., web traffic with TCP port 80 [12]. This method needs access only to the header of the packets. One of

the first network monitoring systems using port based classification was [45] published in 1998.

The advantage of port based method is that it is simple and fast to compute. This is the main reason why several papers use port based traffic classification for the validation purposes [73, 127, 124]. However, the port based method becomes insufficient in many cases [57], since no specific application can be associated to a dynamically allocated port number, or the traffic classified as web may easily be something else tunneled via HTTP. The port based method is a standard and common method but due to the above problems it cannot be considered to be reliable.

1.2 Signature based classification - Deep Packet Inspection

Intrusion Detection Systems (IDS) have enjoyed a spectacular rise. Since Dorothy Denning first proposed the concept in 1986 [75], there were relatively few products until 1998, when a number of vendors began offering commercial network monitoring IDS, e.g., [44, 43]. The IDS market grew rapidly through 2004 when major virus outbreaks occurred on Internet [56]. The IDS vendors realized this time that supporting only port based analysis of the traffic is insufficient to defend against virus spreading and they began to look deep into the packets. That is how the term 'Deep Packet Inspection' was born.

To make protocol recognition feasible, only specific byte patterns are searched in the packets in a stateless or stateful manner (see Figure 1.1). These byte signatures are predefined to make it possible to identify particular traffic types, e.g., web traffic contains the string 'GET', eDonkey peer-to-peer (P2P) traffic contains `'e3 38'`. The common feature of the signature based heuristic methods is that in addition to the packet header, they also need access to the payload of the packets. Current papers use DPI techniques mainly for the validation of their proposed algorithms [110, 91].

DPI-algorithm types. Due to their ability to make finer distinctions than simple string matching, regular expressions are today the primary method for defining Intrusion Prevention System (IPS) [88] signatures. Regular expressions provide space-efficient representation of the signatures. If they fit in the cache close to the arithmetic unit, the one-by-one comparison to the traffic stream can be fast if the rule set is small.

Matching the expression individually is too slow thus a compact representation of the whole signature set is beneficial in terms of signature matching speed. One

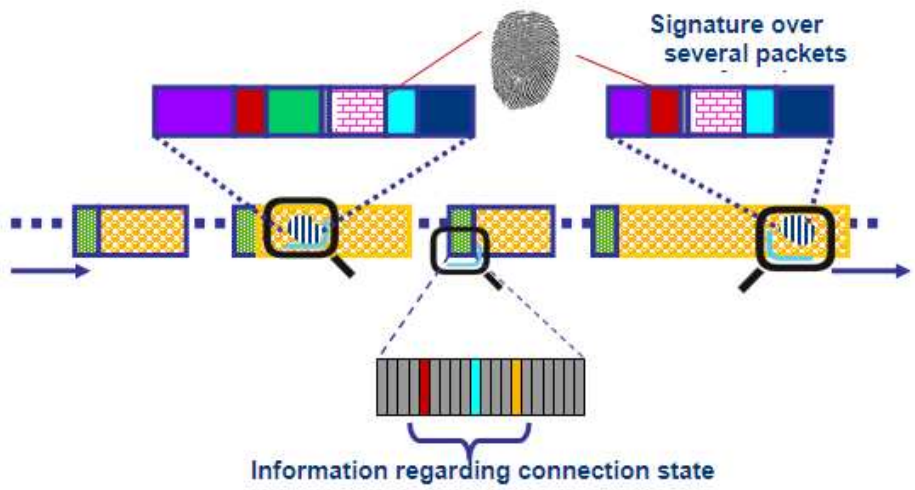


Figure 1.1: Deep Packet Inspection – analysis of encapsulated content over many packets [55]

representation is the **Deterministic Finite State Automate** (DFA) [111]. DFAs match a set of regular expressions by consisting of a number of states each of which has a transition table with 256 pointers to other states. During matching a 'current state' variable is repeatedly updated and it moves between the states of the DFA. Some of these states are marked as 'accepting states' and indicate that the signature matches. DFAs are not a compact way of representing regular expressions, but they have two major advantages: there is a fast matching procedure, and it is possible to compose the DFAs corresponding to multiple signatures into a combined DFA that recognizes all signatures in a single pass. The basic processing steps of a DFA are very simple. The DFA reads a byte from the input, uses it to index into a transition table pointed to by the current state variable, and the value it reads out is the next state. If this is an accepting state the DFA raises an alert which is handled by other parts of the system. These steps are repeated until the end of the input is reached.

When complex DFAs representing individual signatures are combined, the composite DFA is much larger than the sum of the sizes of the DFAs for individual signatures and often exceeds available memory. One approach is to reduce the memory footprint combines signatures into multiple DFAs rather than one [126]. This induces a space-time trade-off between memory and inspection time: the larger the memory budget, the fewer DFAs are required (and thus the faster the matching).

The core reason for the state space explosion of DFAs is that the only way for them to differentiate between two input strings is to have them lead to distinct states. **Extended Finite State Automates** (XFAs) [7] are similar to DFAs, but they also use a small scratch memory in which they store bits and counters that indicate the

progress of the matching operation. Individual states have small programs associated with them that update the scratch memory. When the XFA reaches an accepting state it checks the scratch memory and it raises an alert only if certain conditions are met. The use of the scratch memory allows XFAs to eliminate all major causes of state space explosion.

The per byte processing of an XFA is more complex than that of a DFA, but when compared against multiple DFAs representing the entire signature set, XFAs are typically still faster. The basic scratch memory operations performed by XFAs are setting or resetting a bit in a bitmap and setting, incrementing or testing a counter. When processing a byte the XFA may perform multiple such operations on separate bits and counters. To improve performance XFAs also use a special type of counter: the offset counter. For some types of signatures the counter would need to be incremented for each input byte (for example the signature may need to see whether the number of non-newline characters after an SMTP keyword is above a threshold required to trigger a buffer overflow). Instead of using a traditional counter that would need to be explicitly incremented on each byte, XFAs use offset counters that work as follows: after the keyword is recognized, the packet offset at which the alert should trigger is stored in a sorted list; if a newline is seen before that offset, the program associated with that state disarms the offset counter, otherwise the alert is triggered when that offset is reached.

The advantage of DPI method is that in the case of well documented open protocols, the method can work well. Thus an accurate decision can be ensured with well-defined signatures. On the other hand, only extensive experiences with real traces provide enough feedback to select the best performing byte signatures. The signatures have to be kept up to date, otherwise some applications can be missed, or the method can produce false positives. Finally, this method cannot deal with encrypted content.

The architectures of network devices performing signature matching have to balance a large number of often contradictory goals when designing the signature matching module: meeting performance targets, minimizing the unit cost, staying within the power budget and form factor imposed by the overall architecture of the device, minimizing time to market and maximizing the flexibility to update functionality after the device is deployed. The platforms typically considered are: ASICs, FPGAs, network processors, and general purpose processors. Table 1.1 shows a comparison of different metrics for these platforms.

ASICs are typically used in the highest speed devices because they are the only platform able to meet the high performance requirements. At slightly lower speeds FPGAs become a viable alternative if flexibility is valued much, but they consume

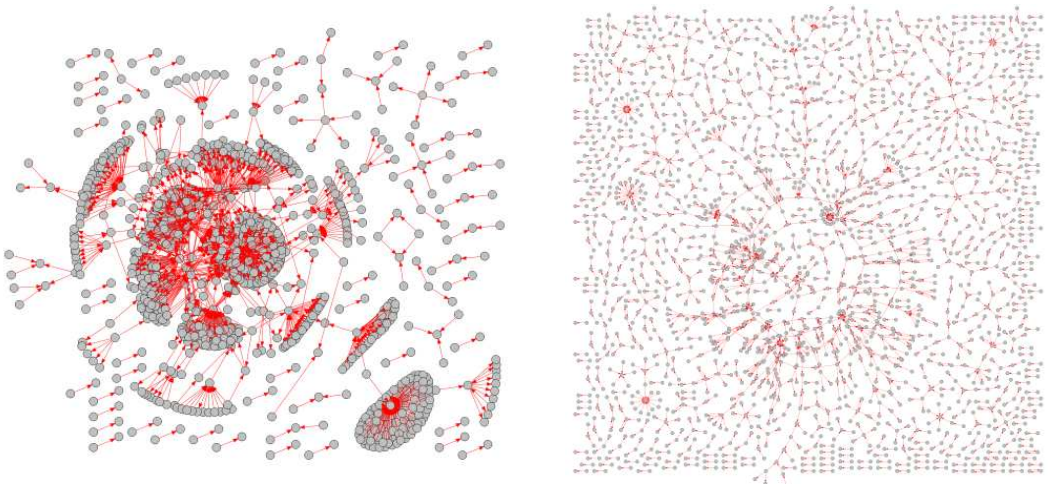
Parameters	Application-Specific Integrated Circuit (ASIC)	Field-Programmable Gate Array (FPGA)	Single Instruction Multiple Data (SIMD) architectures	Network processors	General purpose microprocessors
Cost	Highest	Medium	Low	Medium-Low	Low
Power Efficiency	Highest	Low-Medium	High	Medium	Lowest
Area Efficiency	Highest	Worst	High	Medium	Low
System design					
Flexibility	Worst	Medium	High	Medium	Best
Design time	Highest	Medium	Medium	Low	Lowest
Performance					
Peak performance	Highest	Medium	Medium	Medium	Lowest
Application Performance	Highest	Medium	High	Medium	Lowest

Table 1.1: Implementation alternatives for signature matching [19]

significantly more power than ASICs. Network processors and multicore processors targeted to network applications offer lower throughput, but they have cost and power advantages. General purpose processors are the platform with the lowest performance, but best in terms of cost, flexibility and time to market. Note that the price of a general purpose processor may be higher than that of a network processor or an ASIC, but all networking devices that have intrusion prevention functionality already contain a general purpose processor, so if the existing processor can be used for signature matching, the added cost is zero.

1.3 Connection pattern based classification

The basic idea in connection pattern based classification is to look at the communication pattern generated by a particular host and to compare it to the behavior patterns representing different activities or applications. For instance Figure 1.2(a) shows a large number of clients connecting to e-mail servers that is how groups are formed, while Figure 1.2(b) shows P2P hosts connecting to each other. The first work in this field is presented in [90] describing the working mechanism of the classification algorithm on P2P traffic. That work was extended in [91] (BLINC) in which a general method applicable on a wide range of applications was presented. The connection patterns describe network flow characteristics corresponding to different applications by capturing the relationship between the use of source and destination ports, the relative cardinality of the sets of unique destination ports and IPs as well as the magnitude of these sets.



(a) E-mail traffic (b) P2P traffic

Figure 1.2: Traffic Dispersion Graphs [16]

Iliofotou et al. [16] argue that they took one step further than host-based granularity in [91] and their method operates on aggregated sets (graphs) of related nodes (see Figure 1.2(a), 1.2(b)). Practically they take the whole network traffic and construct graphs of nodes being the communicating hosts and edges constructed on different filters. Such filters can be edge on first packet, edge on first SYN packet, etc. The whole graph is later analyzed with quantitative graph description properties, e.g., node degree distribution, joint degree distribution, rich club connectivity metric, etc.

The advantage of the connection pattern based methods is that packet payload is not needed during classification thus it can deal with encrypted content. These methods imply straightforward visualization of the data. On the other hand, the application specific behavior patterns are often difficult to find, especially if multiple application types are used simultaneously on the same host. The methods require long warm-up time and lack local decision. In order to identify a communication pattern reliably, the methods need a lot of flows coming from and going to a host. There is no chance to identify hosts by the time their communication flows begin to accumulate. These methods have high memory usage as every occurred host is collected. For example, in case of P2P applications the number of seen hosts per user can accumulate e.g., by 20 in every second. The CPU load can be also considerable during the calculation of certain graph metrics.

In current practical solutions [16] the data is sampled to reduce memory requirements. An other common practice is the usage of simplified graph metrics: in [91] node degree is collected to a wired-in limit, thus limited variable sizes can be applied.

1.4 Statistics based classification

Machine Learning (ML) algorithms are used to map instances of network traffic flows into different network traffic classes. Each flow is described by a set of statistical features and associated feature values. A feature is a descriptive statistic that can be calculated from one or more packets - such as mean packet length or the standard deviation of inter-arrival times. Each traffic flow is characterized by the same set of features, though each will exhibit different feature values depending on the network traffic class to which it belongs.

1.4.1 ML-algorithm types

ML algorithms that have been used for IP traffic classification generally fall into the categories of being **supervised** or **unsupervised**. *Unsupervised* [80] (or clustering) algorithms group traffic flows into different clusters according to similarities in the feature values. These clusters are not pre-defined and the algorithm itself determines their number and statistical nature. This is useful in case of the analysis of a yet unknown/uncommon traffic.

For *supervised* algorithms [103, 127, 102, 78, 64] the class of each traffic flow must be known before learning. A classification model is built using a training set of example instances that represent each class. The model is then able to predict class membership for new instances by examining the feature values of unknown flows.

1.4.2 Feature reduction

In ML-based techniques feature reduction is an important issue. In practical IP classification tasks we need to decide which features are most useful given a set of working constraints. For instance, calculating Fourier transform statistics for thousands of simultaneous flows may not be feasible. In addition, the representative quality of a feature set greatly influences the effectiveness of ML algorithms. Training a classifier using the maximum number of features obtainable is not always the best option, as irrelevant or redundant features can negatively influence algorithm performance.

The process of carefully selecting the number and type of features used to train the ML algorithm can be automated through the use of feature selection algorithms. Feature selection algorithms [114] are broadly categorized into the **filter** or **wrapper** model. *Filter* model algorithms rely on a certain metric to rate and select subsets of features. The *wrapper* method evaluates the performance of different features using specific ML algorithms, hence produces feature subsets 'tailored' to the algorithm used.

Feature Description	Abbreviation
Minimum forward packet length	**minfpktl**
Mean forward packet length	**meanfpktl**
Maximum forward packet length	**maxfpktl**
Standard deviation of forward packet length	**stdfpktl**
Minimum backward packet length	**minbpktl**
Mean backward packet length	meanbpktl
Maximum backward packet length	maxbpktl
Standard deviation of backward packet length	stdbpktl
Minimum forward inter-arrival time	minfiat
Mean forward inter-arrival time	meanfiat
Maximum forward inter-arrival time	maxfiat
Standard deviation of forward inter-arrival	stdfiat
Minimum backward inter-arrival time	minbiat
Mean backward inter-arrival time	meanbiat
Maximum backward inter-arrival time	maxbiat
Standard deviation of backward inter-arrival	stdbiat
Protocol	**protocol**
Duration of the flow	duration
Number of packets in forward direction	**fpackets**
Number of bytes in forward direction	fbytes
Number of packets in backward direction	bpackets
Number of bytes in backward direction	bbytes

Table 1.2: Table of flow features recommended for use by [120]

In [103] Moore et al. used 248 flow features to differentiate between application types. With feature reduction about 20 features (see Table 1.2) were selected which are required for traffic classification. In [120] it was examined what happens with the accuracy of ML algorithms if the feature set is even more reduced. It was found that reducing the features set to 7 (see Table 1.2 in bold), the algorithms suffer only minor accuracy degradation.

It is interesting to note that there are theoretical limitations of some of the features, however it is neglected in most of the papers or the feature reduction mechanisms are happened to filter them out by chance. For example, in [10] Hjelmvik mentions that using inter-arrival times is very common in academic literature for the traffic classification field, probably because it is a flow measure that doesn't require access to the application layer data. Hjelmvik [10] suggests not using inter-arrival times for several reasons. One reason is that the same type of session would have different inter-arrival times depending on the bandwidth and latency properties of the link between the client and server, hence the measurement might in some cases tell the observer more about the link properties than about the used protocol. Another reason is that inter-arrival times do not really have much to do with the design of a protocol, i.e. no RFC or protocol specification defines what would be acceptable values for inter-arrival times. Every single client or server application will therefore

have somewhat unique inter-arrival times. This might be good if one wanted to do application identification (for example to see if a web server runs Apache or IIS) but not in order to do protocol identification. The inter-arrival times can also vary depending on the CPU speed and load of the client and server machines.

1.4.3 Class weighting

The number of samples in a training data set for a given traffic class influences the ML-methods to optimize their weights toward the more frequent samples [67]. If samples contain an equal number of traffic flows for each of the network application classes then class probabilities become equally weighted. Although this may negatively impact algorithms that rely on prior class probabilities, it prevents algorithms from optimizing towards a numerically superior class when training (leading to overly optimistic results). It is a trade-off between optimizing ML-algorithms to general applicability or trace-dependent traffic mixes.

1.4.4 ML-algorithm performance

According to [120] the difference in the accuracy of the ML-based methods is negligible comparing to the difference in their computational performance. As the complexity of the ML-algorithms differs a lot, their performance regarding build time and classification speed can also differ a lot. In [120] Williams et al. found that algorithms has to take a trade-off between build-time and classification performance: the fastest algorithm during traffic classification was C4.5 [26] but its built-time is the one of the highest in class.

The advantage of ML-based traffic classification is that flow-level information is enough during classification. This type of classifier is a best-effort classifier with high classification completeness. The drawback of these methods is that pre-classified input trace is needed for supervised learning algorithms.

1.5 Information theory based classification

A useful aid in traffic classification is introduced in [124] and [95] which are based on information theoretic approaches and can group the hosts into typical behaviors e.g., servers, attackers. The main idea is to look at the variability or randomness of the set of values that appear in the five-tuple of the flow identifiers, which belong to a particular source or destination IP address, source or destination port (e.g., see

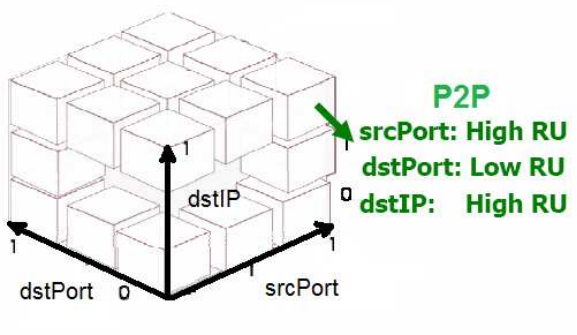


Figure 1.3: Behavior clusters based on Relative Uncertainity values [124]

Figure 1.3). The information theoretic approach cannot be used for flow level traffic classification in the same way as the other methods. It is just an aid in traffic classification and arises the problem that it can only specify very broad application types but not capable of classifying specific applications. This method intensively uses the five-tuple identification of the flows without other additional information.

1.6 Combined methods

A couple of different approaches have been proposed in the literature but none of them performs well for all different application traffic types present on the Internet. Thus, a combined method that includes the advantages of different approaches is proposed in [C1] in order to provide a high level of classification completeness and accuracy. The pros and cons of the classification methods are discussed based on the experienced accuracy for different types of applications. The classification method in [C1] is based on the classification results of the individual methods and uses a complex decision mechanism to conclude the end result of the classification. As a consequence, the ratio of the unclassified traffic becomes significantly lower. Further, the reliability of the classification improves as the various methods validate the results of each other.

Kim et al. [92] repeats the same comparison of different traffic classification methods including the evaluation ML-based techniques. They found that port-based classification still works for a number of protocols. Thus they extended the common traffic characteristics feature list (see Table 1.2) with port numbers and used Support Vector Machine (SVM) classifier to combine them. Zhang et al. [128] describe the implementation issues of a multi-phase traffic classification system, how the data-structures are constructed to store the output of several traffic classification methods.

Method	Advantage	Disadvantage
Port based	fast	inaccurate
DPI	accurate	resource consuming, signature database need constant update, privacy issues, encrypted content
Connection pattern	no need for packet payload, can deal with encrypted content	long warm-up time, lot of memory needed
Statistics based	no need for packet payload, can deal with encrypted content	pre-classified trace is needed for training
Information theory based	no need for packet payload, can deal with encrypted content	inaccurate due to overlapping application clusters
Combined methods	high accuracy and completeness	time frame synchronization issues, different input requirements

Table 1.3: Table of advantages and disadvantages of traffic classification methods

Hjelmvik [10] presents a protocol identification scheme called the Statistical Protocol Identification (SPID) algorithm, which identifies the application layer protocol by using statistical measurements of flow data as well as application layer data (payload analysis). The SPID algorithm utilizes Kullback-Leibler divergence measurements to compare probability vectors created from observed network traffic to probability vectors of known protocols. In [10] the traffic characteristic features are complex derived measures compared to the feature list presented in Table 1.2.

The advantage of the combined methods is that the accuracy of these methods is significantly higher than stand-alone traffic classification systems. The difficulty during the application of these methods is that the applied traffic classification methods may require different input and all of these has to be available during the process e.g., packet level information for DPI and flow-level information for connection pattern based classification. An other issue is that the traffic classification methods may work in different time-frame and their results has to be synchronized. For instance, DPI is processor intensive task, while a port matching is not.

The summary of the advantages and disadvantages of the methods can be found in Table 1.3.

1.7 Off-the-shelf products

There are off-the-shelf traffic shaping products in the market [9, 20, 22, 21]. They provide wide range of functionality including firewalls, traffic prioritization based on QoS classes and also charging e.g., in the case of [9]. To support these functionalities these products imply traffic classification methods. One of the main requirements

Vendor	Traffic Management, Blocking & Shaping	Per-Service, App & Customer Policy-Based QoS	Tiered Services	Network Security	Per-Sub & Application Policy-Based Charging	Lawful Intercept	Targeted Advertising	DRM	Data Leakage Prevention (Regulatory Compliance)	Content Filtering (e.g., Parental Control)	Audience Measurement	Marketing Campaign Measurement
Alcatel-Lucent	Y / P	Y / P	Y / P	Y / P	Y / P	D	P	Y	N	N	D	D
Allot	Y	Y	Y	Y	Y	P	P	P	N	Y	P	N
Anagran	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
Arbor	Y	Y	Y	Y	Y	N	D	D	N	D	N	N
Bivio	Y / P	Y / P	Y / P	Y / P	Y / P	P	P	N	P	P	N	N
Cisco	Y	Y	Y	P	Y	D	D	D	N	P	N	N
CloudShield	Y / P	Y / P	D	Y / P	D	Y / P	N	N	Y / P	Y / P	N	N
Continuous Computing	P / D	P / D	P	Y / P	P / D	D	D	N	N	D	D	D
Ericsson	Y	Y	Y	D	Y	Y	D	N	N	Y	Y	Y
Juniper	Y / P	Y / P	P	Y / P	P	N	N	N	Y	N	N	N
Nokia Siemens	Y	Y	Y	Y	Y	Y	N	N	N	N	N	N
Procera	Y	N	Y	Y	Y	D	N	N	N	Y	N	N
Qosmos	P	P	P	P	P	P	P	P	P	P	P	P
Radware	Y	Y	Y	Y	P	P	P	N	N	P	N	N
Sandvine	Y	Y	Y	Y	Y	Y	P	P	N	P	N	N

Key: Y = Yes; N = No; P = Yes, through partner; D = In development
Source: Light Reading

Figure 1.4: Typical applications Offered by DPI vendors [14]

of these products is that they have to operate in real-time introducing the least possible latency in the trespassing traffic. In this meaning they have to perform in more restricted constraints, e.g., they have to apply fast algorithms, they cannot keep much information in memory about previous activities of a specific user, etc. It is also important that they can hardly rely on unreliable guesses on a given traffic type as in case of misclassification the users would complain. For instance, there are network operators which enforce P2P traffic shaping on their network. If a non-P2P traffic is classified as P2P and filtered by the operator the user feels uncomfortable.

To fulfill the above requirements it is very likely that most of the products use DPI methods during traffic classification. It is out of the scope of this work to 'reverse-engineer' how these products work and try to compare them to each other.

A brief overview of current off-the-shelf products and their functionalities are shown in Figure 1.4.

Chapter 2

Novel framework for traffic classification and validation [J1, C1, C2, C3]

First, we deal with the problem of the validation of traffic classification algorithms. Several different classification approaches can be found in the literature. However, the validation of these methods is weak and ad hoc, because neither a reliable nor widely accepted validation technique nor reference packet traces with well-defined content are available. The validation is typically done with another specific classification method.

2.1 Validation of traffic classification mehods [C3]

A classification method can be convincingly validated only by an active test, for which a number of requirements are fulfilled, such as:

- It should be **independent** from classification methods, i.e. the validation of a classification method by another one must be avoided,
- The test should provide **reference information** about each packet that can be compared to the result of the classification method under study,
- The test should be **deterministic**, meaning that it should not rely on any probabilistic decisions,

- **Feasibility**: it should be possible to create large tests in a highly automated way, and

- The environment where the active measurements are collected should be **realistic**.

2.1.1 Related work

Sen et al. [110] validated their constructed signature database by manually checking the false positive ratio of their technique. Their approach was to investigate TCP connections which were identified as P2P connections. If in fact the content of the connection did not belong to a P2P protocol they counted the connection as a false positive. By the term 'active measurements' they mean that specific traffic type is generated on purpose, thus the kind of traffic is expected can be exactly known at a certain point in the measurement. This is the most common way of developing signature databases as this method ensures that the traffic is sterile, i.e., only a specific application is measured at a time. The measurements they used are not public, therefore others cannot use them as reference.

Karagiannis et al. [91] validated their method by using signature based classification. As there are no commonly accepted and well performing byte-signatures, authors constructed their own signature database.

In [73] Crotti et al. used port based classification method while in [79] Erman et al. used payload based classification method to create training and evaluation data set from the measurements collected at their campus network. In [127] Zander et al. used port based classification for validation of their method. They assume that for the ports they use in the study the majority of the traffic is from the expected application. In this case, it is most likely that few 'wrong' flows would decrease the homogeneity of the learned classes. Therefore their evaluation results can be treated as lower bound of the effectiveness. They also do not consider traffic of the selected applications on other than the standard server ports. Xu et al. [124] validated the identified clusters by checking the found dedicated port of the hosts with the port-application database used for port based classification.

Authors of [78] worked with commonly available traffic traces, but these traces contained only packet headers which excludes such reliable validation methods which are based on packet payload. In [64] the traffic classification method was applied online without capturing the original data due to the lack of capacity to store the massive amount of data which is the consequence of high traffic speeds. This makes impossible to validate the traffic classification by others.

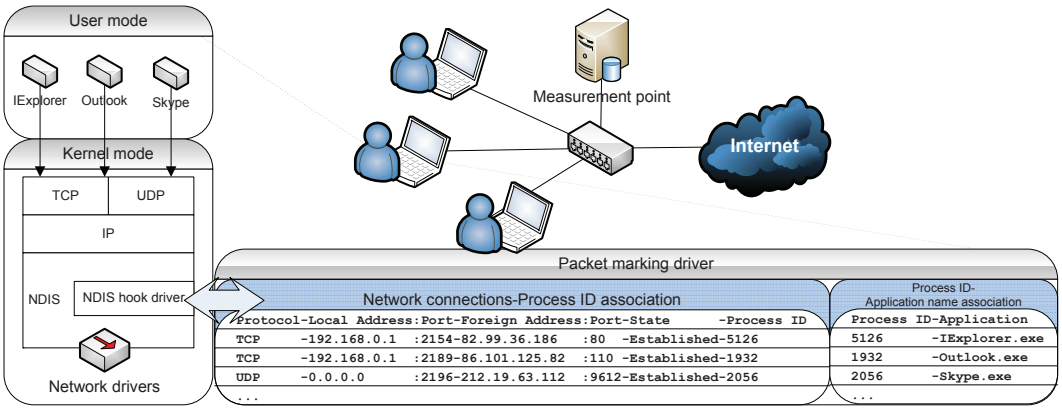


Figure 2.1: The position of the proposed driver within the terminal

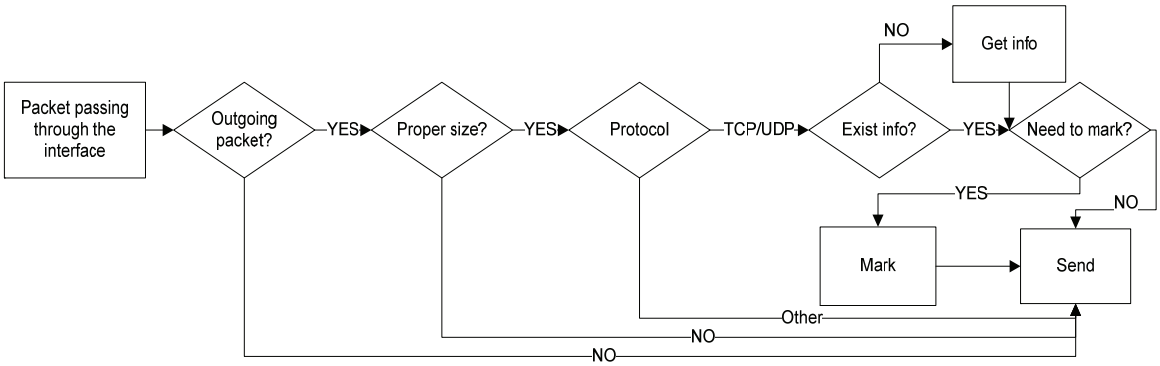


Figure 2.2: The working mechanism of the introduced driver

2.1.2 Proposed validation method

The two main requirements on the realization of the method are that it should not deteriorate the performance of the terminal and the byte overhead of marking should also be negligible. The preferred realization is a driver that can easily be installed on terminals. The position of the introduced driver can be seen in Figure 2.1. It takes place right before the network interface thus each packet exchanged between the terminal and the network has to pass through it. We have implemented a prototype, which is a Windows XP driver based on the Network Driver Interface Specification (NDIS) library. The kernel NDIS library abstracts the network hardware from network drivers and provides an API through which intelligent network drivers can be efficiently programmed. If the sending and receiving functions of the NDIS IP protocol driver are hooked, all TCP and UDP packets can be intercepted and filtered.

To meet our requirements the driver is designed to work in the following way. In the case of a passing through packet the following process takes place (see Figure 2.2):

1. The packet is examined whether it is an incoming or outgoing packet. In case of an incoming packet the process ends without marking the packet as it is not beneficial to mark incoming packets.

2. In case of an outgoing packet the size of the packet is examined. If the current packet size is already the size of Maximum Transmission Unit (MTU), the extension of the packet with marking would lead to IP fragmentation. To avoid this the process continues with only those packets which are smaller than the MTU decreased with the size of marking. Initiating messages in protocols are typically small e.g., the SYN packet of a TCP packet is only a flag, thus there is practically no loss (unmarked flow) with the introduction of this condition.

3. As there is no information in the operating system about those 'network connections' which use other protocols than TCP or UDP, the process continues with only TCP or UDP packets.

4. According to the five-tuple identifier of the packet it is checked whether there is already available information about which application the flow belongs to. The driver has to cache this information because querying the operation system about the existing network connections is very resource consuming and can not be done at high network speeds. We used a hash as the data structure for the cache as it can be directly addressed by the searched data. If there is no information on the flow in the cache yet, the operating system is queried to supply the established network connections and the process IDs of the responsible applications. The process IDs are state specific information in the operating system. To get a universal name about the application, the process IDs are connected to the application's executable name as can be seen in Figure 2.1.

5. When all information is prepared for the marking of the packet, there is a final chance to decide whether the driver should mark the packet or not. The packet marking can be done for all of the packets in the flow, randomly selected packets of the flow, only the first packet of the flow or it is also possible to switch off the marking for specific applications. There is an option for the random selection of packets to be marked to enforce the first packet of the flow to be marked or avoid the first packet to be marked. The sense of these options is to make an optimal trade-off between performance, network transparency and to ensure high chance of recordable marked packets in the case of network loss.

The marking is done by extending the original IP packet with one option field. We selected the Router Alert option field, because the existence of this field is transparent for both the routers on the path and also for the receiver host (according to RFC 2113 [28]). If one uses another option field, it should be carefully checked whether

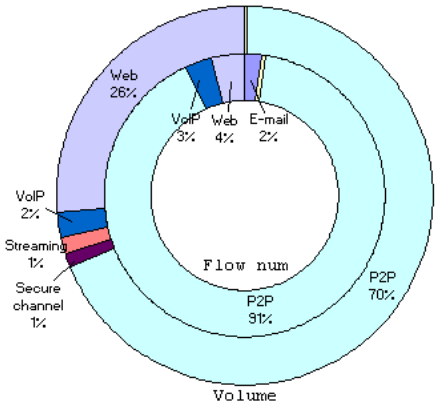


Figure 2.3: The traffic mix of the measurement

the marking is conform to the security policy of the given network, otherwise the marking can be easily removed by an edge router in the border of the access network. In the option field, the first two characters of the corresponding executable file name are added, thus increasing the size of the packet with 4 bytes. The packet size field in the IP header is also increased with 4 bytes and the header checksum is recalculated. As already discussed above, the driver does not mark packets larger than (MTU-4 bytes).

2.1.3 Validation

The validation was done by using the method in real network conditions. The method worked transparently for the communication parties.

A reference measurement [35] has been created as a proof-of-concept of the introduced validation method. For the sake of simplicity, the measurement took place in a separated access network. Our driver has been installed onto all computers on this network. The duration of the measurement was 43 hours. The captured data volume in the network is 6 Gbytes, containing 12 million packets. The measurement contains the traffic of the most popular P2P protocols: BitTorrent, eDonkey, Gnutella, DirectConnect; VoIP and chat applications: Skype, MSN Live; FTP sessions, file-transfer with download manager; e-mail sending, receiving sessions; web based e-mail (e.g., Gmail); SSH sessions; SCP sessions; FPS, MMORPG gaming sessions; streaming radio; streaming video and web based streaming. In Figure 2.3 the traffic mix of the measurement can be seen. Both the volume and the flow number ratio of different applications are presented.

In Figure 2.4 an example of the marked packets can be seen. The IP header shows the increased size of the packet (without the option field, the value where currently is

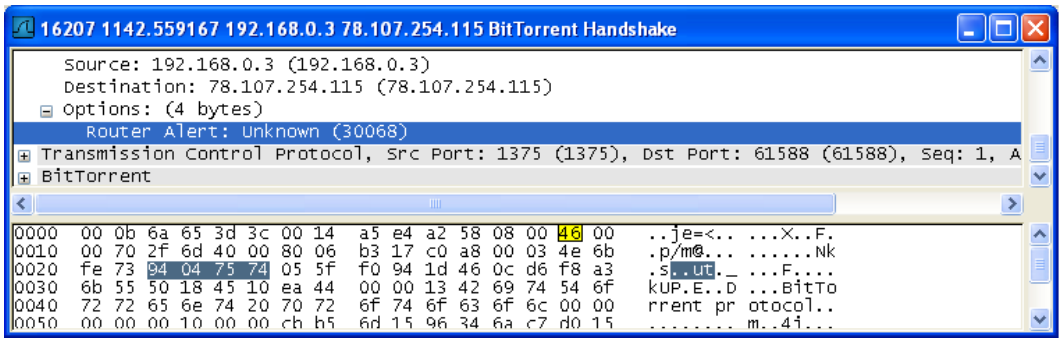


Figure 2.4: A marked packet of the BitTorrent protocol

46 would be 45) and the option field is highlighted, where the last two fields could be used to place the marking. The marking shows that the generating application was the uTorrent [37] BitTorrent client (by the first two characters in its name).

2.1.4 Application

The two key metrics that can characterize the performance of a traffic classification method are accuracy and completeness [91]:

- *Accuracy* means the fraction of traffic identified correctly in the sense that it is associated to the application type which indeed generated it. The classification accuracy values of an examined method show the ratio of the correctly classified traffic compared to all the traffic classified by the examined method for the given application. This can be calculated by Algorithm 1. Using a more common but less precise denotion: $\mathcal{A} = \frac{TP}{TP+FP}$, where TP and FP are representing the number of true positive and false positive decisions for a flow considering an examined method.

- *Completeness* means the fraction of traffic which is categorized by a particular method. It is the fraction of the traffic which the method is capable of associating to one of the application types. The classification ratio values shows the ratio of flows classified as a specific application type by the examined method and the application marked in the validation trace. This can be calculated by Algorithm 2. Using a more common but less precise denotion: $\mathcal{C} = \frac{TP+FP}{TP+FN}$ with the same denotations as in $\mathcal{A}$.

The perfect classification is the $(\mathcal{A}, \mathcal{C}) = (1, 1)$ case. During the classification both of these metrics are intended to be kept around the perfect classification values. The following example shows the accuracy and completeness of two applications classified with one specific method (e.g., `A`) comparing to the reference trace (e.g., `A*`).

```
A - A*, A - A*, A - B*, B - B*, ? - B*
Accuracy: A: 2/3 B: 1/1, Completeness: A: 3/2 B: 1/3
```

Algorithm 1: Calculation of Accuracy

Input: Input flows F, Examind method M, Examined application a, Application marked in the validation trace R

```
1 TP = 0;
2 forall f ∈ F do
3     if M_f^a == R_f^a then
4         TP++;
5 TPFP = 0;
6 forall f ∈ F do
7     if M_f^a == 1 then
8         TPFP++;
9 return TP/TPFP;
```

Algorithm 2: Calculation of Completeness

Input: Input flows F, Examind method M, Examined application a, Application marked in the validation trace R

```
1 TPFP = 0;
2 forall f ∈ F do
3     if M_f^a == 1 then
4         TPFP++;
5 TPFN = 0;
6 forall f ∈ F do
7     if R_f^a == 1 then
8         TPFN++;
9 return TPFP/TPFN;
```

The accuracy and completeness of the classification methods measured on the validation trace can be seen in Table 2.1. The payload based and port based method were implemented by us, using an extended signature database of [2] to increase the true positive hits, while the used port-database is a reduced version of [12] to reduce the false positive hits (similar to [25]). BLINC is the original implementation of the authors of [91]. In the case of chat, e-mail, filetransfer, P2P, streaming, web the payload based method outperformed the other traffic classification methods. In the case of system and tunneled traffic, the port based method was the most complete.

The experienced accuracy and completeness about the traffic classification methods is depicted in Figure 2.5. The most accurate methods are the strict protocol parsers and their lightweight implementation, the payload based traffic classifier. The most complete solutions are those methods which rely mostly on weak heuristics. In this way, such methods can always classify the traffic thus their completeness will be

Application		Payload		Port		BLINC [91]	
		#flow	vol	#flow	vol	#flow	vol
chat	$\mathcal{A}$	0.99	0.99	0.92	0.94	0.02	0.01
	$\mathcal{C}$	0.99	0.97	1.12	1.04	4.05	0.12
e-mail	$\mathcal{A}$	0.99	1.00	0.99	0.99	0.01	0.01
	$\mathcal{C}$	0.99	0.99	1.00	1.01	0.01	0.01
filetransfer	$\mathcal{A}$	0.99	1.00	0.53	0.56	0.01	0.01
	$\mathcal{C}$	0.99	0.99	0.23	0.30	0.01	0.01
P2P	$\mathcal{A}$	0.07	0.77	0.08	0.14	0.01	0.01
	$\mathcal{C}$	0.08	0.77	0.22	0.14	0.01	0.01
streaming	$\mathcal{A}$	0.78	0.89	0.20	0.25	0.01	0.01
	$\mathcal{C}$	0.88	0.92	0.34	0.21	0.01	0.01
system	$\mathcal{A}$	0.64	0.70	0.99	0.99	0.85	0.57
	$\mathcal{C}$	1.03	1.07	0.99	0.99	1.15	0.76
web	$\mathcal{A}$	0.99	0.99	0.91	0.99	0.28	0.41
	$\mathcal{C}$	6.57	3.31	6.52	3.32	0.96	0.23
tunneled	$\mathcal{A}$	0.99	0.99	0.72	0.90		
	$\mathcal{C}$	0.62	0.11	1.01	0.92		

Table 2.1: Accuracy & completeness of the classification methods measured on the validation trace

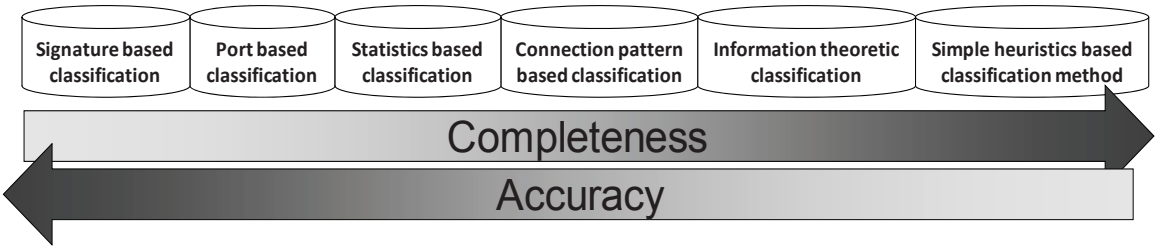


Figure 2.5: The accuracy and completeness rank of different traffic classification methods

high, however their accuracy is low. It should be noted that Figure 2.5 is a general overview of the accuracy and completeness of current methods but it is not intended to mean that the trade-off between these metrics always exist. E.g., in case of a payload based classifier the short byte signatures may decrease the accuracy but increase the completeness. On the other hand well-constructed simple heuristics which are really typical of the traffic – e.g., P2P traffic has parallel UDP and TCP connections –, can be an accurate method. To handle these cases in general is difficult.

2.2 Combined traffic classification method [J1, C1, C2]

Before this work, we have found that papers searched for one ultimate solution to cope with the traffic classification problem in general. We proved that traffic classification methods vary in their accuracy per application types. A complex decision mechanism is needed to provide high accuracy and completeness in all real-world situations.

To combine existing traffic classification mechanisms decision trees were chosen to exploit their following advantages: a) decision trees can be implemented and used efficiently b) decision trees are simple to understand and interpret. The ML technique for inducing a decision tree from data is called decision tree learning. During decision tree construction algorithms for discrete data sets the cost function are information gain related in general which are not directly related to the traffic classification problem. The main problem with them that application types having less flows than other are neglected, e.g., gaming traffic comparing to P2P.

2.2.1 The proposed combination of classification methods

The idea is to combine the results of multiple independent traffic classification modules with a decision mechanism in order to more accurately classify a flow. The suggested system consists of several modules as it can be seen in Figure 2.6.

The input of the system is a packet level network traffic trace. This trace is the direct input of the statistics based classification method and the signature based method as information such as e.g., packet interarrival time, packet size distribution, packet payload, etc. are available in the packet level trace but it is not available in the flow level trace. The input of the port based method, the information theoretic approach and the connection pattern based method is the flow table, which contains a record for every flow present in the trace. It is possible to classify traces without payload but in that case the results of the signature based method can not be taken into account during the final classification mechanism, which can decrease the accuracy of the final decision.

The flows in the flow table are classified using all of the methods separately. After the flow level classification with the classification modules, the final decision mechanism combines the results with an intelligent decision. Beside these modules the system can be extended with other classification modules as well, in order to handle the continuously emerging new applications. In that case the decision mechanism may need to be modified based on the information about the accuracy of the newly added specific classification module.

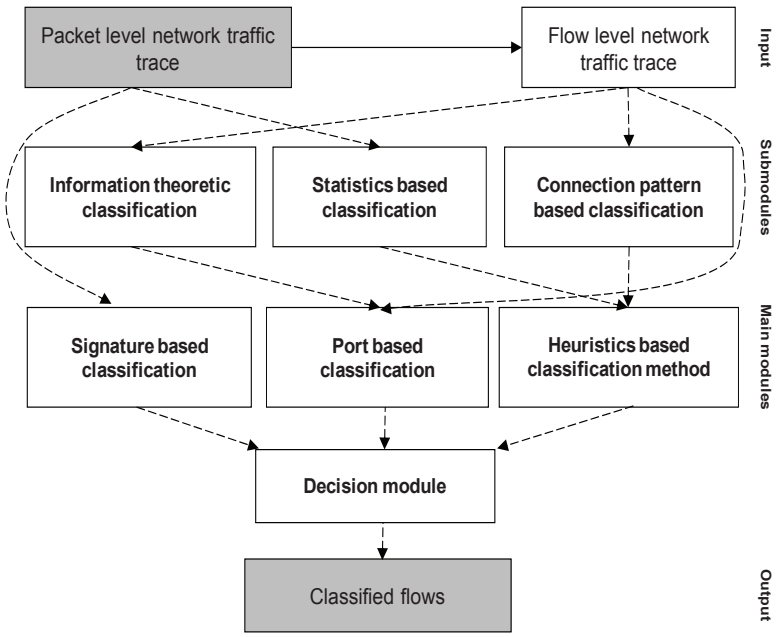


Figure 2.6: The traffic classification modules used in the combined method

2.2.2 Heuristics based modules

The classification process starts with classifying all the flows with the independent modules. The output of the statistics based classification and the connection pattern based classification methods are not the direct input of the final intelligent decision mechanism, instead they are preprocessed, 'fine-tuned' with additional heuristics to reach the most accurate application type without examining application specific port or bitstring information.

Statistics based classification. The statistics based classification method is one of the methods which can serve information corresponding the application without taking into consideration the five-tuple of the flow. As our presumption is that no previous information such as preclassified trace segment is available during classification we can not use Bayesian classification which requires an accurately classified training trace. We used a collection of basic heuristics which can give hints about the different application types based on statistical properties of an examined flow, such as e.g. the packet interarrival time distribution, packet size distribution, average packet size, etc. The following application types are the possible outcome of this method:

- interactive traffic (e.g. telnet session)
- control flows (e.g. part of such protocols as streaming, FTP, P2P–BitTorrent,

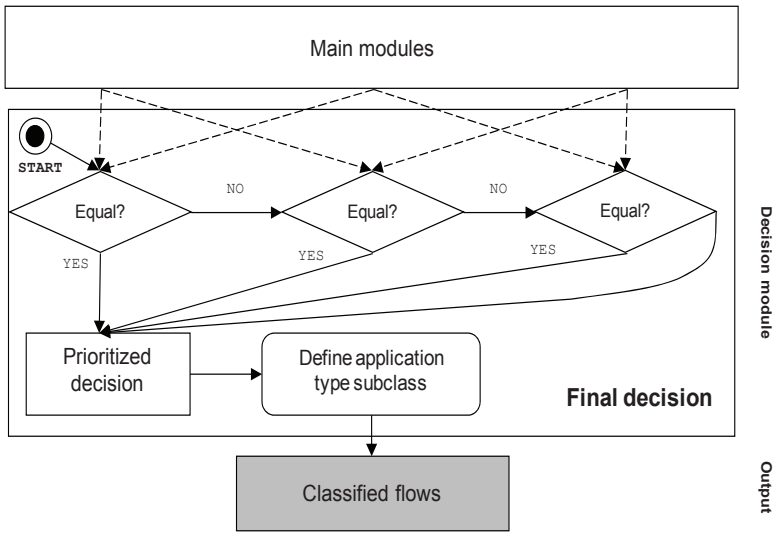


Figure 2.7: The concept of the decision module of the combined traffic classification method

DirectConnect, where the clients maintain a flow with the server during the whole communication)

- streaming (e.g. Windows Media Player on-line media broadcast)
- VoIP (e.g. H.323, Skype)
- file transfer (e.g. download from an FTP server)

We came across with the following special cases when the statistical method fails to accurately determine an application type. If the statistical method is probed against P2P applications, it may classify P2P traffic as file transfer, streaming or VoIP application. From the statistical method's point of view, the difference between streaming and file transfer is small, unless we consider the protocol (UDP for streaming, TCP for filetransfer), so we usually examine the two application types together. The other case is when a flow which has similar upload and download traffic volume in both directions can make our statistical method give such a hint that the flow is VoIP traffic though actually it belongs to P2P traffic. This can be explained by the situation that in several P2P protocols the peers have a value representing the upload/download ratio. Those peers are more likely to be accepted by other peers to connect which have high upload/download ratio. To improve the data distribution efficiency, the protocols (e.g. BitTorrent) enable peers to upload a file chunk to the same peer in the same time where they download from thus results in a VoIP like communication among the peers. We can see that the application specific heuristics have serious role

in the classification of these application types. We constructed the decision mechanism to take these cases into consideration during the processing of the output of the statistical method.

Connection pattern based classification. Another non-classical method which uses the degree of parallelism of flows between hosts is the connection pattern based classification method (BLINC [91]). BLINC can classify web, e-mail, DNS, P2P, chat, filetransfer, streaming, gaming and attacking network activity.

Application specific heuristics. We use the statistics based classification and connection pattern based classification methods together with further heuristics to combine their results and get as exact application type as possible without examining application specific port or bitstring information. As the statistical method is usually capable of classifying the traffic but the result frequently happens to be too general or fall into other application type which exhibits similar statistical properties as the one which has been selected, thus it needed to be further classified with BLINC and additional heuristics.

Classification of the traffic of emerging applications e.g. P2P or streaming is very important. To identify ***P2P applications*** we introduced a connection pattern based heuristic, which uses a novel approach compared to BLINC [91] and [90]. If we look through the P2P applications we can find that nearly none of them are completely P2P, namely the clients register themselves at server like nodes (supernodes). E.g. in case of BitTorrent, a tracker server is operated which tracks the file parts in the torrent system. As every client has a control flow with the P2P supernode, and flows between each other, triangles can be identified when looking at the communication pattern of the nodes.

The main difference between BitTorrent and DirectConnect is that BitTorrent does not maintain the tracker updating flow, but after the updating process it drops the flow and periodically establishes a new one. This mechanism gave a hint to introduce a heuristic for P2P classification which uses the characteristics of the periodical behavior of control flows. Thus, control flows can be used for classification in the following way: those IPs are collected together which have started flows between two IPs repeatedly with any frequency e.g., higher than a minute. As e-mail applications also check mail servers regularly, e-mail has to be excluded from this examination. Similarly, presence protocols also communicate with a frequency but they use one single flow. The packet sending frequency appears inside the flow between the packets, and do not create new flows.

Other useful heuristic in P2P application identification is to search for those flows that go between two IPs both on UDP and TCP as it is very typical for P2P applications (this heuristic is used with success in other works as well [90]). File/user

searches and status/queuing updates go on UDP protocol, while the filetransfer goes on TCP between the peers. In P2P applications the source port of every peer, where the UDP flows originate from, is typically one dedicated port. Such a heuristic which finds the dedicated P2P ports is also used with success.

To detect flows belonging to ***streaming application***, we found the following heuristic: if a streaming data flow goes from srcIP A, srcPort B, dstIP C, dstPort D (UDP) then its control flow can be found at srcIP C, srcPort X, dstIP A, dstPort B+1 (UDP). We mark the flows that match for this criteria, and if we find that the control flow is also a streaming flow then the streaming data flow is also classified as streaming traffic.

If more specific application class can not be determined, a possible output of the heuristical methods can be the ***relayed flow class***. The introduction of the relayed flow search is a novel connection pattern based heuristic compared to BLINC [91]. A relayed flow is such a flow that a host communicates with another host but the communication goes through an intermediate node. To collect these communications we mark flows if a flow exists with srcIP A, dstIP B, transferred byte length C, transferred packet number D, then there is another flow close in time which has a srcIP B, dstIP E, transferred byte length C, transferred packet number D. In our prototype the transferred byte length C was considered to be equal if the transported data volume through the examined flows has a difference below 5%. (The same was used for the number of packets.) The search was executed only among those flows which have transmitted more than 10 Kbyte of data. The presence of relayed flows is a very characteristic feature of several chat and VoIP applications which are able to connect two clients behind firewalls. To solve this problem they both connect to a relay node with no firewall and this node relays the data from one client to the other. E.g. Skype, the P2P VoIP application works like this.

In this section we combined the results of the statistical classification method with BLINC and further heuristics. During the classification the methods can confirm each other's results and the combined method further specifies the application types.

2.2.3 The final decision of the combined classification method

After the execution of all the independent classification modules and having managed to process the statistics based classification and connection pattern based classification output, we are ready to reach a final decision on the application type of every flow. In this section we introduce how the heuristic, the port and the payload based classification results can be combined in the most appropriate way.

The novelty of the final decision mechanism is that it has been constructed on the basis of the empirical accuracy of the classification methods, where the more accurate method is taken into account with higher priority. This final decision mechanism involves majority decision and novel heuristics which select the most appropriate result.

The operation of the final decision mechanism can be seen in Figure 2.7. The signature based method is capable of finding the most specific feature of an application, the signatures of the application layer protocol, so its results are considered with the highest priority during classification. The connection pattern based method is the less accurate one, that is why its results are considered the weakest sign during classification. The decision mechanism is constructed to use these signs from the strongest to the weakest direction as it can be seen on the connections of the comparison boxes in Figure 2.7. We accept the decisions of the modules if the majority of the modules state the same for the given traffic type.

If the classification result is a very common application type (e.g. web, P2P) then the port list and the signatures belonging to this common application type are checked. This step may reveal the specific application which is responsible for generating the flow (e.g. identifying Kazaa as the subtype within the P2P application type).

During classification the simplest case is that if only one method's result is available then we have to accept that. As a preprocessing of the output of the signature based and port based classification modules we do the following. As several P2P protocols use the HTTP protocol for communication, if the payload based method result is P2P and the result of the port based method is web or vica-versa, then we convert both of the results to P2P.

Packet payload available. We compare the results of the port based and the payload based methods and if several matches occur as flows may have been constructed from different types of packets, we decide on the strictest one. This case can occur e.g., when the inspection of the first packet of a P2P application suggests that it is a simple HTTP request with the 'GET' signature then it is revealed in later packets from the 'hash' signature that it is actually a P2P application. Thus two types of applications appeared on the same flow in different packets and we decide for the stricter one. We used the following sequence regarding the strength of support of the results of the decisions:

- the statistical and connection pattern based method have a common sequence: (the strictness in ascending order) filetransfer, streaming, VoIP, interactive applications, web, P2P, other

- the payload and port based method have a common sequence: (the strictness in ascending order) web, chat, P2P, other

If the outcome of this comparison is web, we tag it with the result of the heuristical method to suggest a subclass. As an example, if a chat application uses HTTP protocol and port 80 to communicate to the chat server, and the heuristical method (e.g., BLINC) classifies it as chat, then the final classification will contain this result. If the result of the payload based method exists but the final class can not be decided because the port based result and the payload based result differ then we leave the decision to the advanced port based classification mechanism.

Server and dedicated port identification. The usual port based classification method has been extended with a decision strength to save the information on the accuracy of the port based classification for every decision. The decision strength can be introduced by the extension of the port based system with the result of the information theoretical classification method presented in [124]. This method can be used to search for servers and server ports as well. BLINC can find the servers as well: it uses simple thresholds, therefore it can find servers only with plenty of flows while the information theoretic approach can work with smaller number of flows as well. The port based classification results are extended with a number which tells how strong the belief is:

1. The flow is marked with code 1 if the information theoretical classification method finds either the source or destination IP of the flow as a server IP.

2. The flow is marked with code 2 if the information theoretical classification method finds neither the source nor the destination as a server IP, but one of the ports is under 1024 (as these ports are usually belong to services)

3. The flow is marked with code 3 if the information theoretical classification method finds neither the source nor the destination as a server IP, and both of the source and destination ports are over 1024

4. The flow is marked with code 4 if the information theoretical classification method finds neither the source nor the destination as a server IP, and both of the source and destination ports are over 1024 and our port-application database contains both ports

To find the application of a dedicated port, the execution of the port based method is followed by an additional heuristic which collects those flows where the port is marked with strength 1 (as server), but the application is not known based on the port number. For every IP, for every given port which has such a flow, the results of the payload based method is collected and the most specific application among the

results of the payload based method carrying the highest transported data volume is selected. The result of the port based method is considered as the previously selected application with strength 1 from now on. With this heuristic every dedicated port of an IP which is used by e.g., P2P application or Skype and have any flow classified by the payload based method can be marked as the port of that specific application.

There are two possible cases when the port based classification is used in different ways: if the strength is 3 or 4 than the result of the payload based method is compared to the result of the heuristic method. If the strength is 1 or 2, the method uses the port based method output instead of the payload based method output. If the port based method has no specific output but the strength is either 1 or 2, as the information theoretic approach suggested it as a server port, then the algorithm advises it for manual inspection, and it classifies it according to the result of the heuristic method.

After the previous steps, the next task is to loop through on all the bidirectional flows and if the two directions of a bidirectional flow are classified for different types of application then the stricter is chosen and both directions of the flow are set to that.

Inter P2P user traffic classification. In the case of P2P applications, Windows SMB service, and the passive FTP protocol the high volume of data goes through a flow having dynamically allocated port numbers. To classify them we would need to parse the whole protocol, that is why we use a simpler method: we collect those IPs where P2P, Windows messages and filetransfer occurred and the unknown flows which go among a group of these IPs are classified as the specific application that the IPs belong to. This procedure is similar to the one presented in [90] to mark the possible P2P peers. With this method we use again the concept that the unreliable heuristics are only accepted if it is supported by the outcome of other methods as well or if no other information is available at all.

2.2.4 Automatic decision tree construction and the evaluation of the final decision tree

The evaluation of the final decision tree was performed by creating decision trees with machine learning methods. In data mining and machine learning, a decision tree is a predictive model, a mapping from observations about an item to conclusions about its target value. In a decision tree structure, leaves represent classifications and branches represent conjunctions of features that lead to those classifications.

We evaluated the proposed final decision module in the following way:

1. The validation trace was traffic classified by all the methods independently.
2. For every flow the transmitted data volume has become the weight of the flow.
3. For every flow, the generating application was extracted from the packet markings in the validation trace.
4. A decision tree based on these data was created by the method of [27] in which case the input was the results of the different classification modules and the expected output was the result of the generating application based on the validation trace.
5. For every flow, the decision of the proposed final decision module was extracted.
6. A decision tree was created by the method of [27], in which case the input was the results of the different classification modules and the expected output was the result of the proposed final decision module.
7. The two resulting decision trees were compared.

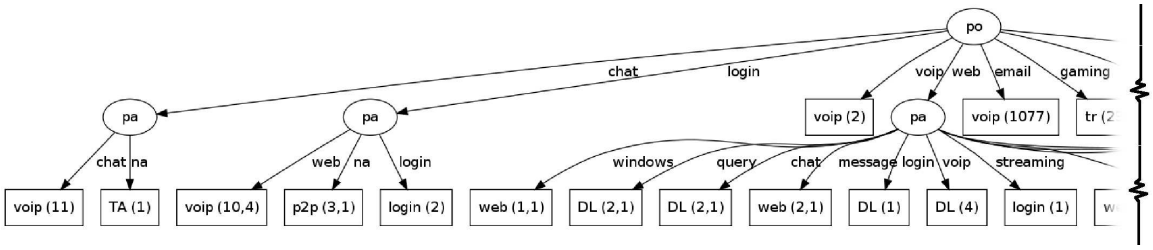


Figure 2.8: A part of the constructed decision tree

We used a machine learning based decision tree construction method of [27] which was set to optimize on information gain during the partitioning of the output clusters based on the results of the different traffic classification results. The constructed decision tree by the method of [27] can be seen in Figure 2.8. The entry point of the decision tree is the port based classifier module. Etc.

The constructed decision tree differs from our proposed algorithm. The reason for the difference is that the method of [27] practically neglected any other traffic than P2P and optimized the decision tree for P2P traffic. The weighting of flows by the transmitted data resulted in that P2P traffic dominated the learning set of the algorithm. We repeated the construction of the decision tree without using weights but in that case the number of P2P flows caused the same result.

To overcome this problem, papers [103], [127] use same number of input flows of different classes or the statistically oversampling – creating flows artificially– of

those application classes where the input sample is low. The problem with this approach that some of the applications has very limited number of flows. Statistical oversampling helps only in those cases where the magnitudes are in the same order of magnitude. In our case the number of flows of e.g., gaming and P2P may not even fall into the same magnitude.

An other method was used to evaluate the final decision mechanism. The decision tree was constructed in the same way as above in this section (Steps 1-3) but the output of the final decision was also fed into as input parameter. The idea behind this validation step was that if our proposed method comes up as the root of the constructed decision tree than it means that the information gain is the highest from it comparing to all the separate methods.

Figure 2.9 shows the result of the decision tree construction. The result meets our expectations that the final decision module was placed to the root of the constructed decision tree.

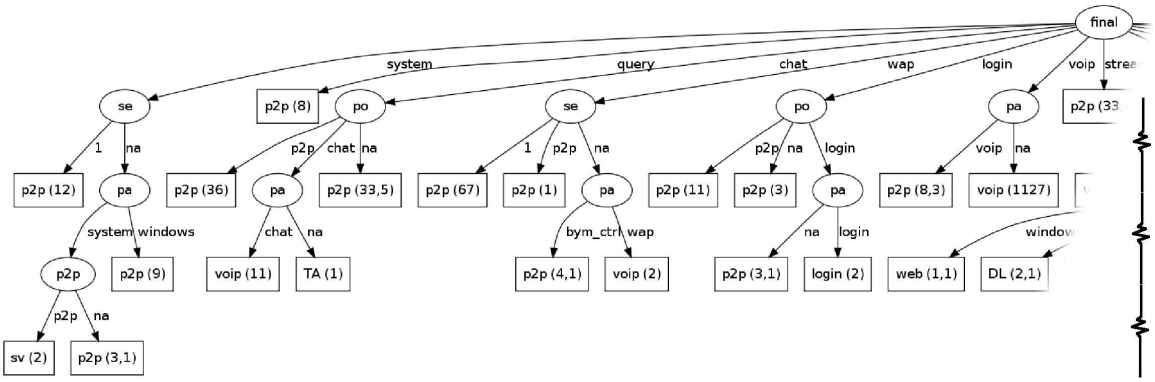


Figure 2.9: A part of the constructed decision tree with the proposed combined decision included

2.2.5 Automatic decision tree construction

In the previous section we used the common decision tree construction methods for discrete data. The common among these methods is to consider the distribution of values of the candidate subclasses and optimize for information gain resulting in more and more homogeneous classes. As the distribution of the values in the subclasses depends on the number of different values, the application types with few samples get neglected.

We propose to optimize on traffic classification metrics directly during decision

tree construction. Our proposal is to regard all applications and traffic classification methods with equal weight. To optimize on both accuracy and completeness we define the target function as the Euclidean-distance from the perfect classification $(\mathcal{A}, \mathcal{C}) = (1, 1)$ of the calculated average accuracy and completeness of a specific traffic classification method $m \in M$ on a set of flows F containing application set A: $f(F, M, A) = \min_{m_i \in M}\{((1,1) - \frac{1}{|A|}\sum_{a \in A}(\mathcal{A}^a_{m_1}(F), \mathcal{C}^a_{m_1}(F))), ((1,1) - \frac{1}{|A|}\sum_{a \in A}(\mathcal{A}^a_{m_2}(F), \mathcal{C}^a_{m_2}(F))), ...\}$, with the same denomination as in Section 2.1.

Algorithm 3: Automatic Decision Tree Construction function (ADTC)

Input: flows F, Methods M, Applications A

1 $m_{min} = f(F, M, A)$;
2 **if** $|M \setminus m_{min}| > 0$ **then**
3 $A\_tmp = D(m_{min}(F))$; `/* Collect the possible outputs of the input set of flows` F `of method` m_{min} `*/`
4 **forall** $a \in A\_tmp$ **do**
5 $F\_tmp \leftarrow F \cap m^a_{min}$; `/* Collect the flows of` F `having output` a `regarding method` m_{min} `into` F_tmp `*/`
6 ADTC(F_tmp, $M \setminus m_{min}$, A_tmp);

The decision tree construction algorithm can be followed in Algorithm 3. This is a recursive process, the conventional way as decision tree construction is done for discrete values. In Step 1 the optimal method out of M with the minimum Euclidean-distance from the ideal classification is determined for the input set of flows F. The selected method is saved to m_{min}. If the set of remaining methods is not empty (Step 2) then the possible decisions for the input flows F of method m_{min} is gathered to A_tmp in Step 3. The input flow set F is grouped into several flow sets F_tmp each containing one specific decision for the possible application type with m_{min}. A new automatic decision tree construction iteration is started for these set of flows F_tmp.

Figure 2.10 shows the constructed decision tree with the automatic decision tree construction process for the validation trace. (The numbers in the bullets are ids and have no other meaning.) The first edge from the root towards the 'rate' node shows that first all the traffic is clustered via the traffic characteristics based method. Its result is later grouped into 'VoIP' and 'na' ('result is not available') classes which are further clustered by the 'pa' (payload based DPI method) and 'p2p' (inter-p2p traffic classification module). etc. The leaves shows the most frequent application type of the validation trace for the specific cluster, while the edges going to the leaves show the percentage of the specific application comparing to the total number of other applications in that cluster. This ratio is calculated based on the flow number. The overall average hit ratio of the clusters considering all the applications with equal weight is 93%.

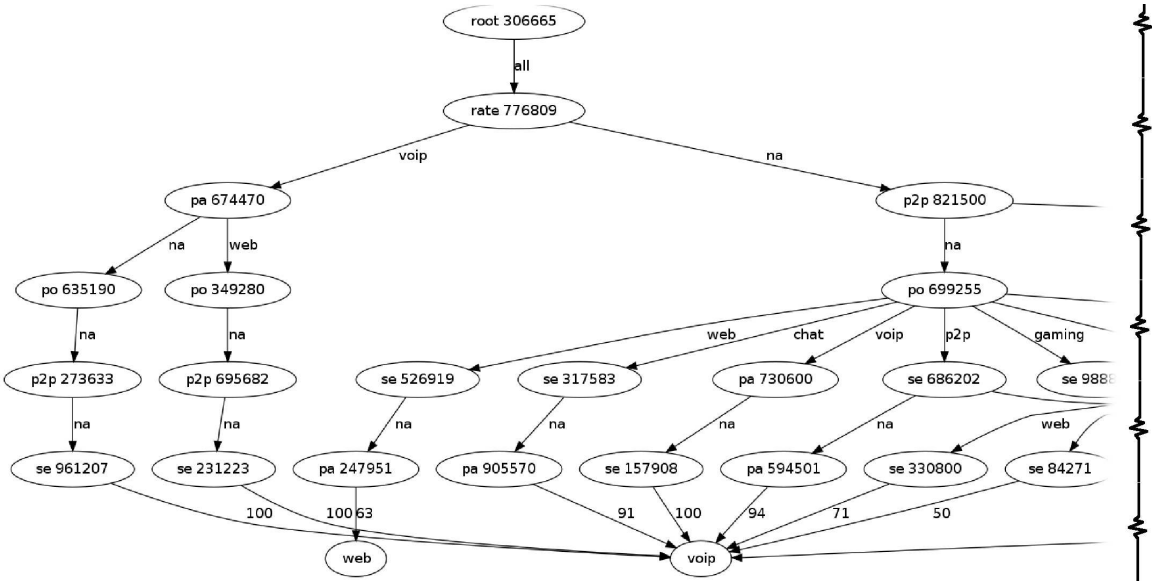


Figure 2.10: A part of the constructed decision tree with Automatic Decision Tree Construction /po – port based classification, pa – payload based classification, se – server/dedicated port search, p2p – inter p2p communication tagging heuristic, rate – traffic characteristics based classifier/ (the numbers in the bullets are ids and have no other meaning)

Note that the most important contribution of the proposed decision tree construction algorithm is the introduced target-function. The decision tree construction algorithm converges to a locally best decision tree. This is not the globally best decision tree most likely. All of the possible decision combinations should be evaluated to achieve a globally best decision. This would be a very processor-intensive and time consuming solution. One possible approach to speed-up the search and to try out several locally best solutions choosing the best performing among them is the usage of random forest algorithms [86].

2.2.6 Validation

We validated the combined traffic classification method with the proposed validation mechanism in Section 2.1, with the addition that the classification of VoIP applications has been extended with ideas from [106]. The improvement of both classification completeness and accuracy can be observed in all application types. Figure 2.11 shows that e-mail, filetransfer, streaming, secure channel, and gaming traffic has been

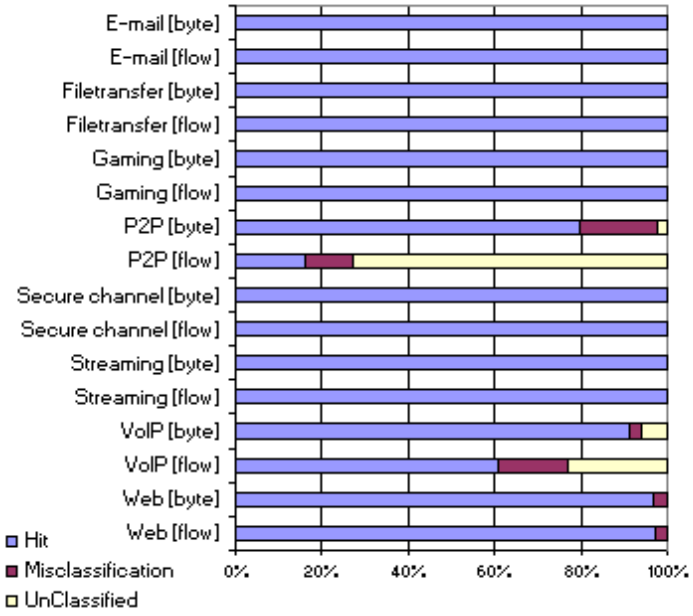


Figure 2.11: The results of the combined classification method compared to the reference measurement

identified very accurately. This is due to the fact that these applications use well-documented protocols, open standards, and do not constantly change. In the case of those protocols which use encryption, the session initiation phase is critical as this phase can be identified the most accurately. In such common protocols as SSH or SCP it can be done with full success, however in such proprietary protocols like Skype the identification fails for several flows.

In the case of classification of P2P applications there are several problems: one thing to note is that P2P applications created plethora of TCP flows containing 1-2 SYN packets probably to disconnected peers. This is the primary reason of the large number of unclassified P2P flows, while the unclassified P2P volume is low. As there is no payload in these packets, the signature based methods can not work. The flows are initiated from dynamically allocated source ports towards not well-known destination ports, thus the port based methods also fail. The server search and P2P communication heuristic methods (see Section 2.2.3) also fail because there are no other successful flows to such IPs.

Some small non-P2P flows were also misclassified into the P2P class. Fortunately, the number of such flows is small both in flow number and byte volume. We realized that the reason behind is the not fully proper content of the port-application database. Creating too many port-application associations easily results in the rise

of the misclassification ratio.

The constant change of P2P protocols also causes some inaccuracy in the classification: there are new features added to P2P clients day-by-day, and their working mechanism can be typical for a selected client not the whole protocol itself.

Another problem of traffic classification is a matter of philosophy. There is traffic which is the derivation of other traffic: the simplest case is the DNS traffic which is the result of any traffic which uses domain names instead of specific IP addresses. For example, web creates DNS traffic though users do not want to create DNS traffic on purpose. There are more complicated cases: e.g., MSN uses HTTP protocol for transmitting chat messages, which do not need to be considered as web. Furthermore, the MSN client transmits advertisements over HTTP, but this cannot be recognized as deliberate web browsing. This raises the question whether such HTTP flows from the MSN application which are classified as web would have to be considered as misclassification, or it is acceptable that they are classified as web. In this comparison, to be fully objective, only that kind of traffic was considered as hit where the classification outcome and the generating application type (the validation outcome) agreed. For example, the chat on the DirectConnect hubs which has been classified as chat could have been considered as actually correct but in this comparison it was considered as misclassification.

The high VoIP hit ratio is due to the successful identification of both MSN Messenger and Skype. Skype is difficult to identify: for some of the Skype flows the problem is the same as in the case of P2P applications, further Skype is a proprietary protocol designed to ensure secure communication thus it is difficult to obtain a good protocol description. However, authors of [106] found a characteristic feature of Skype: the application sends packets even when there is no ongoing call with an exact 20 sec interval. In Section 2.2.3, there is a P2P identification heuristic which was designed to track any message which has a periodicity in packet sending thus the extension of the original method in Section 2.2.3 for the specific 20 sec periodicity of Skype was straightforward. The validation showed us the deficiency of the classification of Skype, thus with a simple extension of the algorithm it became proper for accurate Skype traffic identification as well. In this way the idea of Section 2.2.3 has been validated as it proved to be robust for the extension with new application recognition, and also the validation mechanism proved to be useful.

As a summary, Figure 2.12 shows the improvement of both accuracy and completeness of the introduced combined classification comparing to the individual methods (see Section 2.1.4 for details). The improvement of both of the measures can be observed in all application types.

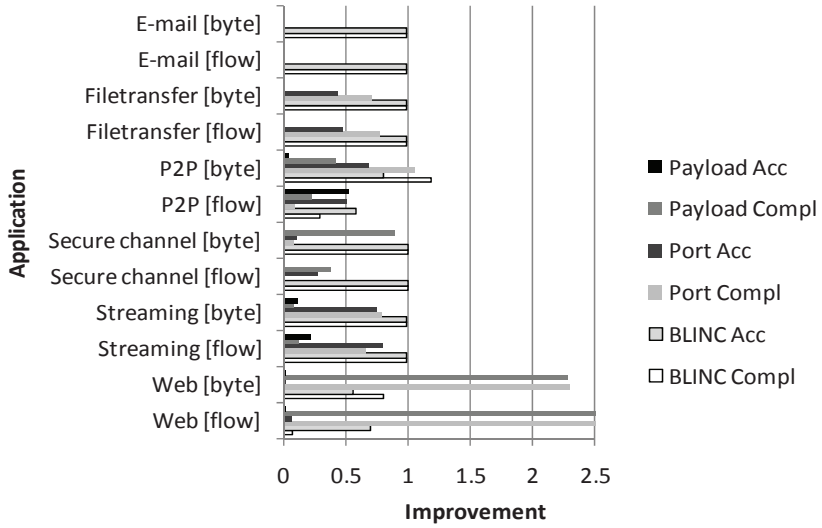


Figure 2.12: The improvement of accuracy and completeness of the introduced combined classification comparing to the individual methods

2.2.7 Application

The method is applicable for analysis of traffic mixture of network traces. Due to the speed limitation of some applied algorithms the traffic classification can be done mainly offline.

The presented methods can be used for analysis of traffic mixture of operational networks. A result of such analysis is given below. The analyzed trace is a three day long measurement collected in an operational mobile broadband network. Figure 2.13 shows the traffic mix in volume. The types of applications used by subscribers have a large influence on the network traffic. It can be seen that in the network the web browsing and P2P application take most of the bandwidth. Due to the fact that web browsing is downlink dominant, it contributes with fewer amounts to the uplink direction.

The share of the P2P traffic and the web traffic stays constant during daytime and the share of the P2P traffic comparing to other traffics grows for the night and early morning hours. The hours of the growth of P2P traffic volume share coincide the hours of the growth of the uplink traffic volume share. Apart from the web and the P2P traffic, there is a considerable amount of streaming traffic. Due to the fact that the streaming traffic is downlink dominant, it only contributes to the downlink traffic share. Other applications such as e-mail or FTP give only a few percent of the total traffic.

Figure 2.14 shows the upload/download traffic ratio over time. The uplink/downlink volume share shows the ratio of data sent and received by subscribers. The share of

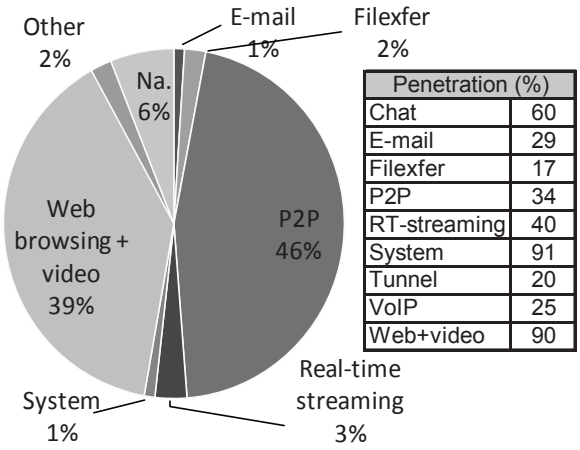


Figure 2.13: Application volume share

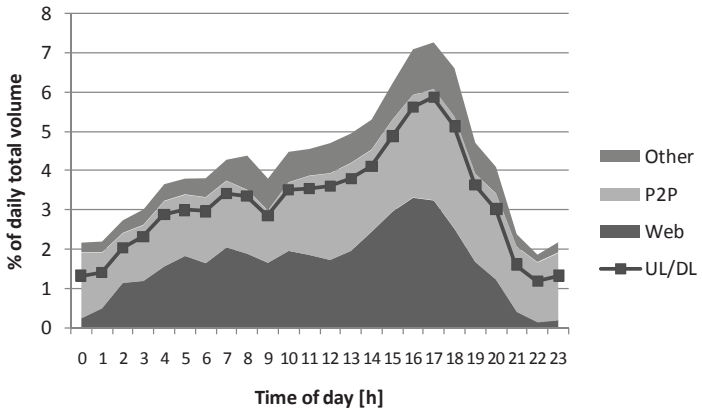


Figure 2.14: Upstream/downstream profile over time

the uplink traffic stays constant slightly above 1% of the total traffic during daytime and grows for the night hours. This is the effect of the missing of web application during night which is downlink dominant, thus its traffic does not contribute to the downlink traffic at night.

2.3 Summary

In this chapter we introduced a new active measurement method which can help in the validation of traffic classification methods. The introduced method is a network driver which can mark the outgoing packets from the clients with an application specific marking. The main advantages of the new method are that it is based on

realistic traffic mixtures, and it enables a highly automated and reliable validation of traffic classification.

With the introduced method we created a measurement and used this to validate the state of the art traffic classification methods. Examining the results of the different traffic classification methods, their accuracy varies for different application types. Basically, none of them can provide a solution that is capable of identifying correctly all traffic types present in various networks. We have showed the advantages and drawbacks of the different types of traffic classification methods.

Using this knowledge we introduced a novel traffic classification method. The novel traffic classification method combines the different traffic classification methods and introduces a set of new heuristics, aiming at improving both the completeness and the accuracy of the traffic mixture identification process. The main benefit of the novel approach from the point of view of operators and administrators is that the ratio of the unclassified traffic decreases significantly. As another advantage, the reliability of the classification process improves, since the various methods can confirm the results of each other. Moreover, the application types can be identified more specifically, by applying the various methods successively. The proposed traffic classification mechanism was validated by our introduced validation method. The traffic classification method has been proved to be working accurately but also some deficiencies in the classification of P2P applications and Skype has been identified.

Chapter 3

Characteristics of gaming traffic [J2, J4, J5, C4, C5]

Gaming traffic analysis is important for traffic classification due to its emerging penetration, and for simulator supported server architecture design. Game traffic depends on two main factors, the game protocol and the gamers' behavior. Based on popular real-time multiplayer games in this chapter we investigated the latter factor showing how a set of typical game phases – e.g., player movement, changes in the environment – impacts traffic on different observation levels. The nature of human behavior has such a high impact on traffic characteristics that it influences the traffic both at a macroscopic – e.g., traffic rate – and at a microscopic – payload content – level. The role of player activity can be important for modeling status update message intensity coming from game server while environment changes can imply server exchange.

Increasing online game popularity causes increasing loads on game servers. It is believed that understanding player behavior can aid in the understood of their impact on Quality of Service (QoS), network traffic characteristics, and it also helps in developing efficient protocols and architecture optimization techniques. The design of an accurate player model for Massively Multiplayer Online Games (MMOGs) makes it possible for future research to predict and simulate player behavior and population fluctuations over time. Further it makes it feasible of the evaluation of existing traffic models and architecture designs.

Obtain player behavior and gaming environment information is difficult. As far as we know before our work there were no papers dealing with this problem obtaining this information from Massively Multiplayer Online Role Playing Game (MMORPG) traffic based on non-intrusive measurements. Other works modify the game itself, turn on some client side logging which is later collected or reverse engineer the protocol

messages and parse the packets. Our traffic characteristics based method can be applied for a wide range of games as it is protocol format independent, it builds on the common traffic characteristics of this game genre.

Emerging Massively Multiplayer Online Real Time Strategy games (MMORTS) require complex game server architecture to make the transmission of the state information of a huge number of units generated by a lot of players feasible. This architecture design is supported by network traffic simulations based on the accurate characterization of player behavior [125]. However, these characteristics of player behavior in Real Time Strategy (RTS) games have not been investigated yet, thus, this is the main motivation of this work. In particular, we introduce a method that can identify the war periods in real-time strategy game sessions based on non-intrusive measurements thus, it is possible to analyze a vast number of game plays.

3.1 User behavior detection algorithm for MMORPG traffic [J2, J4, J5, C4]

In [82] Fernandes et al. showed what happens when an avatar performs different actions in the virtual world, at different places and under different network conditions. Our work extends the investigation of gaming traffic of [82] in the function of time gaining information about gaming environments, further we redefine the player activity applied in [68] to gain information about player behavior. Our work aims at deriving detailed character movement and gaming environment information as e.g., [82] but obtaining this information from non-intrusive measurements without the requirement of the detailed knowledge of the whole protocol as in e.g., [115] or [99].

This chapter discusses in details how player movement and gaming environment changes affect the traffic characteristics at a macroscopic level. Detection of player actions would make it possible to answer such interesting questions as e.g., whether the self-similar property of human behavior appears in the traffic as well due to the player actions. We need a method to collect the necessary statistics on player actions from data sources containing bulk information. As there are a number of different games with proprietary protocols, a deep packet inspection method is not generally applicable. A statistical method would be desirable as it is general and works for even new upcoming games as well. It would make it feasible to analyze vast number of gaming sessions in non-intrusive measurements obtained at high network aggregation level. In current literature there is no such method which can collect the necessary statistics, thus, we propose a method which builds on heuristics to deduce the movement or idle state of the player and its surrounding area conditions from

passive measurements.

We observed the internal structure of the network traffic when the player is in moving or stalling state and the environment around him when it is crowded with other players or deserted. This observation has been examined and validated with several methods. It was found that even if a heuristic method based on statistical properties of the traffic has several fallbacks it can be applied on the recognition of game states with high hit ratio. The proposed method was thoroughly validated on network traffic of World of Warcraft [38] and Silkroad Online [54]. It was also tested on Guild Wars [51], Eve Online [47] and Star Wars Galaxies [49] to check how generally applicable the method is. Although this list does not contain every possible MMORPGs but we argue that the observed traffic characteristics are common among MMORPGs, thus the proposed player state detection method may be applicable for most MMORPGs.

We examined the similarity of ingame player and real world human behavior. It cannot be expected that the two environments would completely be the same, but some properties which were found earlier in connection with human activities are expected to appear in the gaming environment as well.

3.1.1 Related work

Recent research of networked games focuses on modeling low time-scale characteristics, such as packet inter-arrival times and packet sizes, to develop efficient control methods of network traffic and optimally utilize gaming servers. Since the popularity of multiplayer online games rapidly increases there is an urgent need to develop more realistic traffic models.

Recent MMORPGs follows historically after First Person Shooter (FPS) games. They are close to each other in gaming experience and server status update mechanisms. Claypool et al. [71] discovered the dependence of server traffic bandwidth on ingame events. In their later work [62] they managed to refine their measurements and divided UT2003 [46] into the fundamental user interaction components of movement and shooting, sub-dividing movement up into simple and complex movement, and sub-dividing shooting based on the precision of the weapons being shot. In [112], Svoboda et al. created a traffic model differentiating game-states focusing on such problems which will be critical to networks with shared resources. Basically they differentiated active and idle game periods which are related to the problem of proper radio resource allocation. Cricenti et al. [72] condensed the result of ingame events as a correlation in packet sizes of Quake4 [53] traffic. They proposed an ARMA(1,1) model to capture the time series behavior of the gaming traffic. In [115] Tan et al. proposed the Networked Game Mobility Model (NGMM), for synthesizing mobility in FPS networked games. NGMM utilizes application level aspects of networked game

traces to statistically model FPS games.

Today a large part of the gaming traffic is generated by MMORPGs, thus several works dealing with this type of traffic appeared. In [94] Kim et al. analyzed Lineage II. It was one of the world's largest MMORPGs in terms of the number of concurrent users at that time. They found that the bandwidth usage of the server traffic is about ten times larger than the client traffic. This asymmetry is due to the fact that the server transmits all the information to construct the visual environment for the clients in the same region.

Kim et al. [107] provided the first look onto player behavior analysis through a measurement study on World of Warcraft [38]. Their goal was to answer the following questions: how does the population of the virtual world change over time, how are players distributed in the virtual world and how do they move in the virtual world. They found that player distribution appears to occur on a power-law distribution, and players move to only a small number of zones during each playing session. In particular, certain locations which have rare equipments tend to be popular places to visit. Further, large cities which have a wide range of services also tend to be popular. Thus, players will tend to congregate in popular locations.

In [82] Fernandes et al. showed what happens when an avatar performs different actions in the virtual world, at different places and under different network conditions. The Second Life [48] client makes intensive use of network resources. They identified the reasons of high bandwidth consumption e.g., ingame music, numerous unique character clothes, etc. deriving from several game servers. Liang et al. [99] collected mobility traces of 80k avatars spanning more than 20 regions over two months in Second Life. They analyzed the traces to characterize the dynamics of the avatars mobility and behavior. They extracted character position information by parsing update packets from the servers. They discussed the implications of their findings in connection with the design of peer-to-peer networked virtual environments, mobility modeling of avatars. Their measurements suggest that a hybrid mobility model that incorporates both random way-point mobility model (for outdoor) and pathway mobility model (for indoor) would be suitable.

Chen et al. [68] found that the characteristic features of MMORPGs are the strong periodicity, temporal locality, irregularity, and self-similarity. The self-similarity property was explained by the existence of a hidden on-off process according to the authors. Their hypothesis was that on-off periods are due to the player active and idle periods. They defined a player active if the player sends packets above a threshold and idle if the packet sending rate of the player is below a threshold. Authors of [69] found that MMORPG bots traffic is distinguishable from human players by the regularity in the release time of client commands, the trend and magnitude of traffic burstiness in multiple time scales, and the sensitivity to network conditions.

Our work can extend the above studies with several aspects. [69] provided us with

the proof that human behavior is present in the network traces at macroscopic level. We argue that the self-similarity property of the gaming traffic, which was found in [68] and was modeled with an on-off model does not fit due to the impact of human behavior. In this work we show that player behavior in the MMORPG traffic has Long-Range Dependence (LRD)[1] properties. [68] hints that the traffic rate changes of the game may be due to player actions, but the detection heuristic has not been validated and is not precise as will be shown later in this section.

Our work extends the investigation of gaming traffic of [82] in the function of time gaining information about gaming environments, further we redefine the player activity applied in [68] to gain information about player behavior. Our work aims at deriving detailed character movement and gaming environment information as e.g., [82] but obtaining this information from non-intrusive measurements without the requirement of the detailed knowledge of the whole protocol as in e.g., [115] or [99].

3.1.2 The proposed user behavior analysis method

We introduce methods for user behavior analysis in this section. Our constructed methods are based on the analysis of active measurements. In the term *active measurement* we mean that we measured the traffic on the client side generated by a player, thus the user actions can be controlled. Chen et al. [68] studied game traffic characteristics with the help of power spectrum diagrams to reveal periodicity in the traffic. However, the power spectrum does not grab the characteristics of the traffic in the function of time. We decided to apply wavelet analysis [101] on the MMORPG traffic to exploit the following features: (a) one major advantage afforded by wavelets over power spectrum analysis is the ability to perform local analysis, (b) the other advantage is the capability of revealing aspects of data like trends, breakdown points and discontinuities in higher derivatives.

We created several active measurements in a controlled environment by capturing the generated network traffic of World of Warcraft [38] and Silkroad Online [54] and manually log the time of the different states of the player. After the measurement, we compared network characteristics and the ingame activity of the player. Examples from this experiment are shown in Figure 3.1 and Figure 3.2. The figures depict the bandwidth characteristics of the traffic coming from the server in the function of time. The two games have similar characteristics though World of Warcraft has about the twice the network bandwidth consumption as Silkroad. Observations on the traffic characteristics could be made in connection with player activity and the environment

[1]Practical meaning: the rate of decay of statistical dependence in a time series is slower than an exponential decay. See [63] for details.

changes:

- *Moving/Stalling*: The player moving actions imply constant location changes, which implies constant update of the environment information around the player. The player moves on areas with refreshed state and with obsolete information, thus update information occurs intensely but with variable frequency. During stalling, the state update regularity is independent from the player and synchronized by the server with more regularity. Our observation was that the character movements increase the noise in the traffic rate time series.

- *Inside/outside city*: The density of the gaming environment of other players influences the number of independent actions which need state update at the player in their vicinity. Entering to a crowded environment induce a level-shift upward in the traffic rate, while leaving it results in a level-shift downward. Note that 'city' naming convention refers to the observation that the densely populated areas are usually in a city in the gaming environment, however it refers to any densely populated area.

To grab both of these characteristics in time we used wavelet decomposition (see Algorithm 4 for the Matlab [39] skeleton code of the algorithm in details):

- Reconstruction of the signal from the low frequency component gives hint about the gaming environment (densely populated city/extinct desert)

- The high frequency component gives hint about user activity (stall/run)

In Figure 3.3 the wavelet decomposition of the traffic presented in Figure 3.1 can be seen. It can be noted that the moving and stalling states are well-distinguishable on the high-frequency component: in case of moving, it becomes noisier. The environment changes can be well-tracked on the low-frequency component.

To improve the accuracy of the method we extended the wavelet decomposition with a state-machine which follows the changes in the environment. It tracks whether the player is inside city or outside city and does not let sudden changes in the environment if temporal transients occur in either of the frequency components. Figure 3.4 shows the defined states and state transmissions in the proposed detection algorithm. The skeleton code follows these states and state transitions. The high frequency component change makes a state transition between stalling and moving, while the low frequency component change results in a state transition between the environments. The environment change is only possible with the player moving. This model lacks the possible state transition between stalling in and outside the city. This could occur when a player uses teleport. In this case the state machine gets into the proper state in a few additional steps, but keeps the false state hit ratio low.

Algorithm 4: MMORPG state recognition algorithm

```
   Input: rate
   Output: out
 1 nx = length(ina); out = zeros(size(ina));
 2 [C5,L5]=wavedec(ina,5,'db1');A5 = wrcoef('a',C5,L5,'db1',5);
 3 [C1,L1]=wavedec(ina,1,'db1');A1 = wrcoef('d',C1,L1,'db1',1);
 4 medF=mean(abs(A1));maxF=max(A1);
 5 phase_length=30 /depends on the wrcoef paratmeteres/;
 6 state=0 /init=stalling, just where?/;
 7 if (abs(A5(1)-min(A5)) < abs(A5(1)-max(A5))) then
 8     state=4;
 9 else
10     state=2;
11 for i=2:nx, do
12     frek_min=abs(abs(A1(i))-0);
13     frek_max=abs(abs(A1(i))-medF);
14     p=polyfit([1:phase_length*3],ina(i-phase_length:i+phase_length),1);
15     if (state==4) AND (frek_max<frek_min) then
16         state=3 /stalled outside, started to move/;
17     else if (state==2) AND (frek_max<frek_min) then
18         state=1;
19     else if (state==3) AND (frek_min<frek_max) then
20         state=4 /stalled and started/;
21     else if (state==1) AND (frek_min<frek_max) then
22         state=2;
23     if (state==1) AND (A5(i)<A5(i-1) AND p<0.5) then
24         state=3 /it moved out to the desert/;
25     else if (state==3) AND (A5(i)>A5(i-1) AND p>0.5) then
26         state=1;
27     out(i)=state;
```

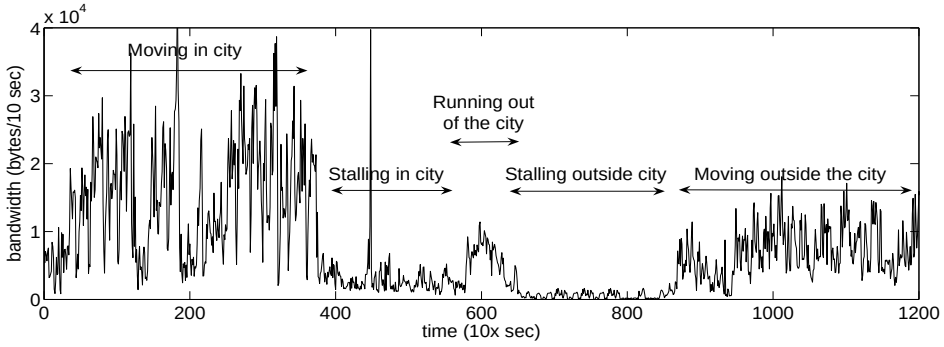


Figure 3.1: World of Warcraft measurement bandwidth (bytes/10 sec)

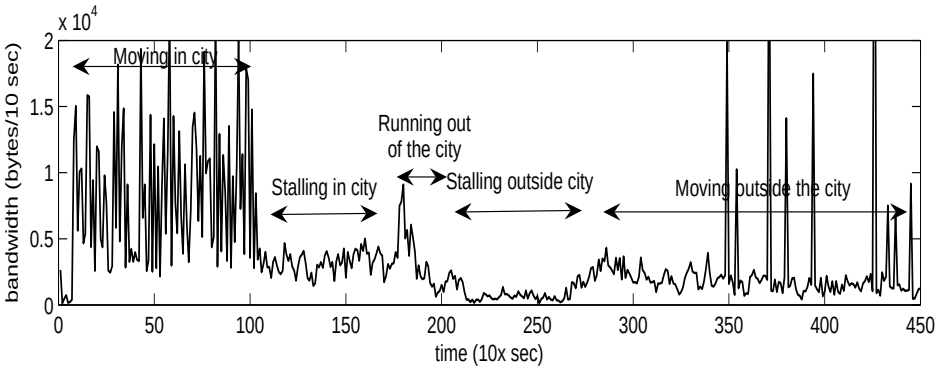


Figure 3.2: Silkroad measurement bandwidth (bytes/10 sec)

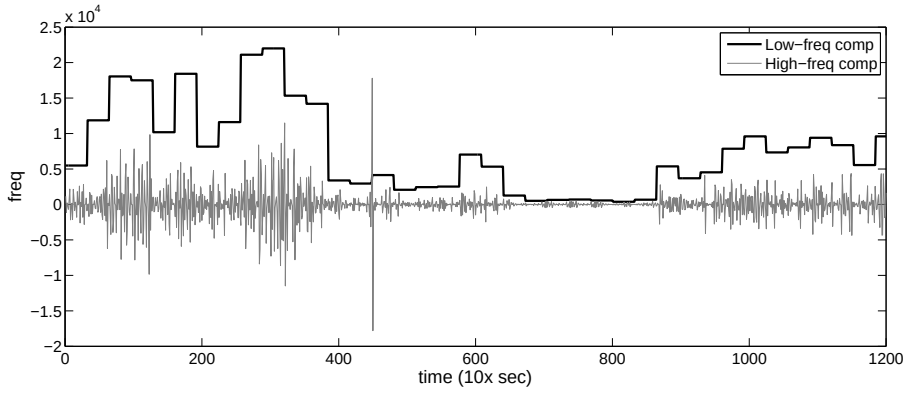


Figure 3.3: Analysis of World of Warcraft traffic and the effect of different environments and character actions

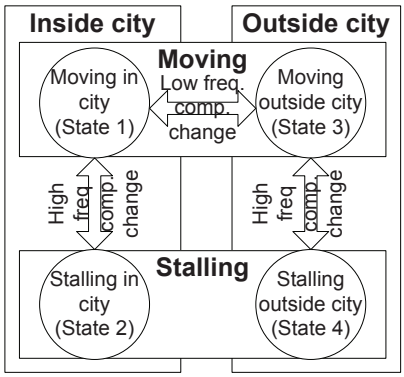


Figure 3.4: States and defined state transmissions in the proposed method

Difference	0	1	2	3
Manual logging	74%	14%	8%	4%
Video processing	68%	19%	10%	3%

Table 3.1: The difference of the MMORPG state recognition algorithm and the ingame video processing algorithm

The algorithm was constructed to adapt to the analyzed traffic, thus, no fix thresholds regarding the typical bandwidth have to be set. The advantage of this feature is that any MMORPG game traffic can be analyzed without previous parameter tuning.

It is important to note that the method is independent of the packet directions. We analyzed the server traffic in this section, but due to the per packet TCP ACKs, the client traffic is correlated to the server traffic. Thus client traffic also shows the presented traffic dynamics, though on a lower level. The advantage of server derived traffic is that its characteristics are more determined: the server is programmed to update player state in a designed manner that is less influenced by random network effects. The client-server traffic considering together can also fit into the presented algorithms.

3.1.3 Validation

We collected and examined 9 hours of MMORPG traffic (World of Warcraft [38] and Silkroad Online [54]) for the validation phase. The validation of the proposed method has been performed in several steps. In the **first step** we created several active measurements in a controlled environment by capturing the generated network traffic of the game at the client side and manually log the time of the different states of the player. After the measurement, the proposed method was applied on the network traces, and the states which were indicated by the proposed method and the logged states were compared. A state is the estimated action of the character in a given environment in a 10 sec period. The definition of the states and their values are showed in Figure 3.4. In Table 3.1, the 'manual logging' titled row shows that in this phase of the validation 74% of the states were exactly recognized and 14% of the states were missed with a nearby state. Nearby state means that the state can be reached within one state transition.

Second, we focused on Silkroad Online and World of Warcraft to check the accuracy of the method thoroughly and in bulk measurement in an automated way.

The following different methods can be collected to create game play logs:

- Capture network traffic, and parse the protocol e.g., [115]. Our aim was to analyze wide-spread MMORPGs. In the case of MMORPGs, the wide-spread ones

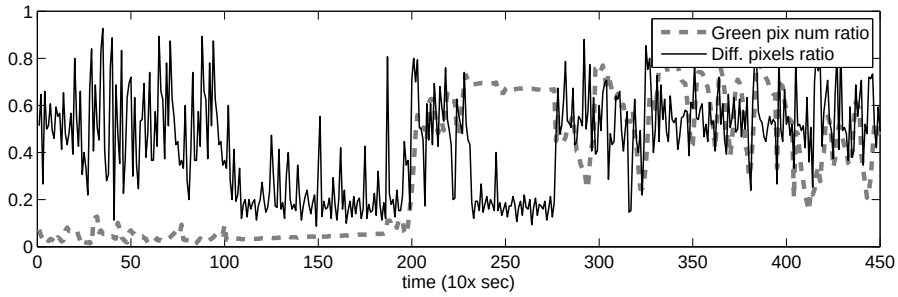


Figure 3.5: Analysis of a Silkroad ingame video

are all commercial games meaning that the protocols are proprietary and practically reverse engineering and complete reimplementation of the game server would be required to catch the ingame events.

- Use ingame scripting languages e.g., [107]. There is no wide-spread ingame scripting language which is supported by most of the MMORPGs. This is not a general applicable method to analyze several MMORPGs with the same technique.

- Use key and mouse movement loggers. The advantage of this method is its completely game independence. On the other hand it is difficult to track the ingame events with this technique as the key and mouse events can refer to other commands than character movement. Therefore a complex parsing mechanism would be required to find out the player actions. Further it is unfeasible to extract the ingame environment from the key and mouse logs.

- We decided to capture ingame video. The ingame video provides straightforward information about the character movement and the surrounding environment of the player. This technique is game independent and the required code to extract the necessary information is simple. One drawback of this method is that neither space nor processor efficient as the live measurement generates huge video files that have to be processed later.

Several measurements were taken in Silkroad Online and World of Warcraft by capturing the network traffic and the ingame video during game play. The recorded video files were processed by a heuristic algorithm to create automatically player action and game environment logs to replace the manual work for bulk measurements. Our ingame video analysis code and an example ingame video can be found at [41].

The heuristic algorithm grabs the characteristics of the ingame screens of both the environment and the moving activity. As an example outside the city, the characters

march through grassy fields and forests where majority of the pixels has green color or a gradient of green. Our prototype implementation counts the number of greenish pixels of every frame of the recorded ingame video file. (We are aware of that the implementation is not completely general in its current status, as e.g., a desert area where the environment is not greenish would mislead the algorithm. With a more complex color processing mechanism, our idea would be not restricted to specific gaming environments.) The ratio of the greenish pixels can be seen in the function of time in Figure 3.5. We used the HSL color space (hue, saturation, lightness) to check if a pixel is greenish. HSL color space makes it possible to test whether the color falls into a given hue range, but its saturation and lightness can take different values which is due to e.g., the day/night visual effect in the games. Figure 3.5 shows that as the character started inside the city with running around for 15 min, the green pixels ratio—the green and non-green pixels ratio on a given screenshot—is low and remains low for that period even if small fluctuations occurred. As the character stops, the green pixel ratio remains relatively constant. After the 30^{th} minute the player leaves the city and there is a significant rise of the green pixel ratio as in enters to the nature. The varying number of trees, grassy fields, roads, monsters, etc. results in fluctuations in the green pixel ratio, but the mean value remains high.

The player activity – whether player stalls or runs – is tracked by the comparison of the pixels around the legs of the player in two neighboring frames. The ratio of the differing pixels is low in case of stalling and high in case of moving. Figure 3.5 shows that in the first 15 minutes the character was running around, thus this ratio remained high. As the player stopped the ratio dropped and the mean of this value remained low during stalling. The above two metrics make it possible to track the activity of players and the game environments by creating simple limits. It is evident from the results that as the metrics can temporary fluctuate, it is advisable to smooth them e.g., by calculating their sliding mean value.

In Table 3.1 'video processing' titled row shows that 68% of the states were identified exactly and 19% of the states were missed by a nearby state. The validation showed that the algorithm misses mainly the city/outskirts transitions. It is clear that the term city is not perfectly proper as the buildings in the city and walls are just visual effects, thus the high bandwidth is in connection with the densely populated areas not the exact location if it is inside the city or not. It would be also possible to analyze the client generated traffic to aid in state recognition. This can be a task of further work.

In the **third phase** of the validation process, we created several active measurement traces of the following MMORPG games beside World of Warcraft [38] and Silkroad Online [54]: Guild Wars [51], Eve Online [47] and Star Wars Galaxies [49] to check how generally applicable the method is. We found that the method performs

well in cases where the player population is high and the character actions generate considerable traffic. This states well for World of Warcraft, Silkroad Online, Star Wars Galaxies and Guild Wars as well. In the case of Guild Wars one can ask whether the method is affected by the game rule that leaving a city results in that only the members of the guild and the non-player-characters remains around the player. We measured that the traffic characteristics does not alter considerably. Our algorithm is inaccurate in the case of Eve Online and the explanation for this needs further investigation. We observed low traffic rate in our traces which makes it difficult to differentiate between the states.

3.1.4 Application

The method can provide network simulators with player behavior parameters. The LRD property of the gaming traffic was revealed and it was also showed that the popular explanation for the presence of Long-Range Dependence (LRD) by heavy-tailed player behavior periods [68] cannot be held. Barabási [60] showed that the human activity patterns display bursty dynamics with interevent times following a heavy-tailed distribution. Crovella [74] showed that self-similar property of network traffic is caused by user 'think time'. In this section we show that human behavior can result in a non-heavy-tailed but strong-correlated process causing the same LRD effect.

We measured the network traffic of an operational third generation (3G) mobile broadband network. The measurement point was a middle-point interface, containing the subscribers outgoing and incoming traffic. Current 3G networks are fully capable of supporting broadband applications e.g., P2P, VoIP, gaming, streaming, etc. having the measured average RTT in [52], [59] far below 1250 msec which was measured to provide acceptable user experience in MMORPGs [83]. These applications exist in 3G mobile environment as it was shown in [C2]. We preprocessed the measurement with filtering out the MMORPG traffic from the total traffic with the method introduced in [C1]. Then we analyzed the filtered traffic with the methods presented in the previous sections. The examined traffic is a three-day-long passive measurement, which contained about 200 World of Warcraft flows and 100 other MMORPG flows.

In Figure 3.6 we can see the cumulative distribution function of the duration of filtered MMORPG traffic. It can be seen that though the average game length is about 90 minutes, we can find even 5-hour-long games. It is interesting to compare the session durations to e.g., [107] where authors found that 50% of the population remains online for 10 minutes or less. Their hypothesis was that players login to see if other friends or guild members are online and then leaving if not. Probably our measured player population was more determined to play and stay ingame.

Table 3.2 shows the ratio of time spent in different states in MMORPGs in the

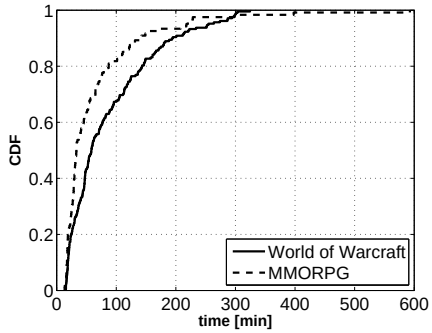


Figure 3.6: Distribution of the duration of filtered MMORPGs

	WoW		MMORPG	
	< 1h (%)	> 1h (%)	< 1h (%)	> 1h (%)
City run	23	20	19	19
City stall	35	42	29	35
Out run	30	27	25	30
Out stall	12	11	27	16

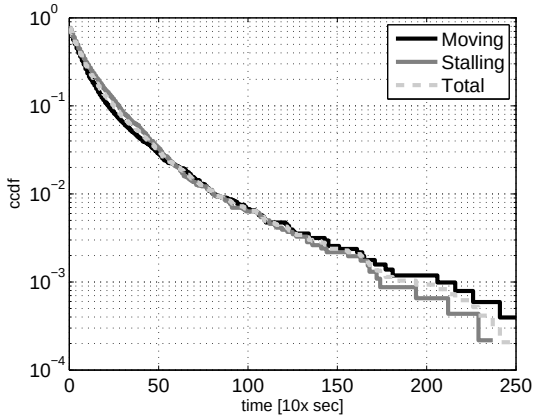
Table 3.2: The ratio of total time spent in a specific state by the players

case of the flows less than one-hour-long and more than one-hour-long. With this differentiation it is possible to check if longer MMORPG sessions are the consequence of longer 'away from computer' status of the players. We found that the state ratio do not significantly altered which means that these players take some rest during game play but remain active in most of the time. A possible explanation of the high ratio of stalling state can be explained by the socialization activity of players according to [50]. [50] says that a major part of the players like to use MMORPGs to chat and catch up with friends and not only play.

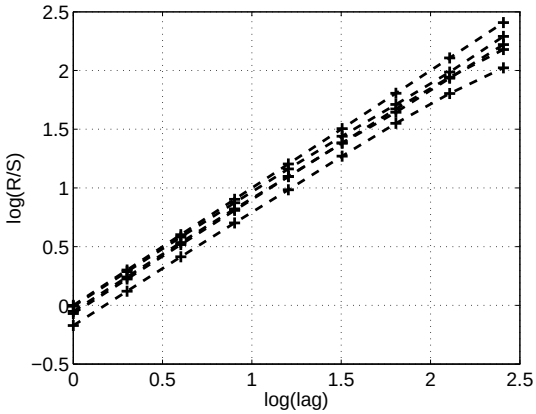
We intended to compare the observed player behavior characteristics in the gaming environment with **real world data**. Chen et al. [68] suggested that gaming traffic has Long-Range Dependence (LRD) and possibly self-similar property. They constructed an on-off model for the traffic and their hypothesis was that it may fit due to the impact of human behavior with the changes of active and idle states of the players. We argue that to unfold the origin of LRD property of the traffic is a difficult and often impossible task [74]. LRD property can be the result of a superposition of heavy-tailed on-off periods [122], TCP propagation property [118], the competition of TCP sources but it may also be caused as an artifact by non-stationarity.

Figure 3.7(a) shows the sojourn time[2] ccdf of the moving and stalling periods in Wow traffic. It can be seen that the ccdf resembles to an exponential distribution,

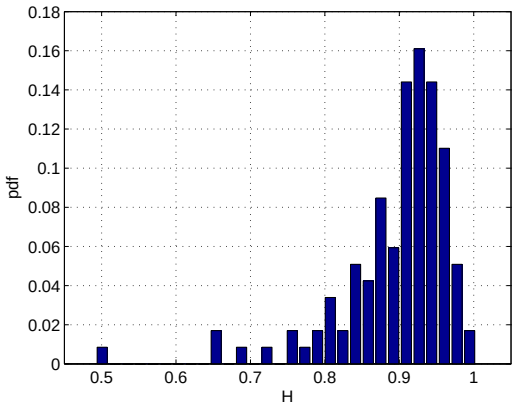
[2]The amount of time the player remains in the same state



(a) Sojourn time ccdf in moving or stalling periods in Wow traffic



(b) R/S plots of the player states in several Wow sessions



(c) Estimated H-parameter pdf of the player states in the Wow traffic

Figure 3.7: Parameters of World of Warcraft sessions

meaning that the distribution is not heavy-tailed. This excludes the explanation that the LRD property is the result of the heavy-tailed distributions of the movement state durations.

On the other hand, we checked the autocorrelation function of the time series of the states and it was found that the decrease of the correlation vs the lag is slow resulting in strong autocorrelation. Cross-checking this statement, we reconstructed the analysis performed in [68] by examining the time-series of the states in the function of time with the R/S plot [63] method. This time-series is such abstraction of the original traffic in which every effect – e.g., transport layer characteristics: TCP properties, packet sending rate, packet size – is eliminated and let us focus on the

state change events of the player. Chen et al. in [68] performed similar analysis except that they defined a player active if the player sends packets above a threshold and idle if the packet sending rate of the player is below a threshold. This property was found to occur due to the environment changes in our experiments. In our analysis we used our more sophisticated state definition. Figure 3.7(b) shows examples of the R/S plot of several WoW time-series constructed by the analysis of the filtered WoW traffic with our proposed method. The R/S plots also support the high correlation property of the time-series of the player states. The estimated H-parameter distribution for all the filtered and analyzed WoW traffic can be seen in Figure 3.7(c) with an average of 0.89.

3.2 Battle event detection in RTS games [C5]

Current wide-spread Massively Multiplayer Online Games (MMOGs) are implemented mainly as heavily centralized client-server architecture. However, this architecture has several limitations like poor scalability. To overcome such drawbacks an intensive research has been initiated to focus on more decentralized or even peer-to-peer support systems.

In case of MMOGs a single game server cannot handle all game events efficiently, requiring the world to be divided into several smaller parts which are served by a cluster of game servers [125]. In case of Massively Multiplayer Online Role Playing Games (MMORPGs) it is a straightforward approach as the player can only be at one place at a time. Player movement across territory boundaries is handled by the handover of the player instance between the different gaming servers.

Providing Real Time Strategy (RTS) game service to a high number of simultaneous players (MMORTS) causes the problem of server load balancing becoming even more complex. The player can manage hundreds of units spread across the virtual world. It is inefficient to transmit all the information around every single unit of the player. Cecin et al. [66] propose to obtain up-to-date data from the server only in those cases when the player is actually concentrating on the specific territory. The ingame events on the surrounding area of the left-alone units are simulated. The authors argued that the economy, military build-up and combat related tasks are following each other therefore, information about the duration of these tasks is vital to make efficient network traffic simulations possible. In [66] Cecin et al. argued that there had not been such published work yet thus, they applied guessed parameters based on their own gaming experiences. The motivation of our work is to find efficient methods to extract this necessary information for MMORTS player and network simulation. Therefore, we believe that our results contribute to establishing a solid base for the architecture design of upcoming MMORTS games.

This chapter focuses on the issue to establish the possibility to answer the above questions. We need a method to collect the necessary statistics from data sources containing bulk information. In some papers authors use game server logs (Lee et al. [97], Feng et al. [81]) as an input for further analysis. In RTS games any player can be the candidate to host the game server thus logs should be collected from plenty of end users which makes this approach less feasible. Another possible approach is to monitor the network traffic. As there is a number of different games e.g., a deep packet inspection method is not generally applicable as it would be valid for only one specific game. A statistical method would be desirable as it is general and works for even new upcoming games as well. It would make it feasible to analyze vast number of gaming sessions in non-intrusive measurements obtained at high network aggregation level.

In current literature there is no such method which can collect the necessary statistics thus we propose a method which builds on heuristics to deduce the battle events from passive measurements. We observed the internal structure of the network traffic when the forces are collected and sent into battle and also when the battle is over. This observation has been examined and validated with several methods and found that even if a heuristic method based on statistical properties of the traffic has several fallbacks it can be applied on the recognition of battles with high hit ratio.

3.2.1 Related work

In [71] and [70], Claypool showed that RTS games traffic rate is influenced by the number of players. They argued that there are exact level shifts in the bandwidth in case of various numbers of players. The analysis of the effects of player behavior on the traffic characteristics of RTS games are currently missing. In this section we show that there are other effects than player number which can be discovered in the network traffic of RTS games. (For further related work in multiplayer games see Section 3.1.1.)

3.2.2 Impact of war on traffic characteristics

Current client-server gaming environments works by clients sending information about the actions of its own player. The server assembles and distributes this information to make each client determine the state of the gaming environment. Each machine with identical set of information could process the data on its own and the environment would appear exactly the same for every player. In case of First Person Shooter (FPS) games or MMORPGs the number of commands a player can issue is similar during the whole game (apart from that in case of improving the capabilities of the character

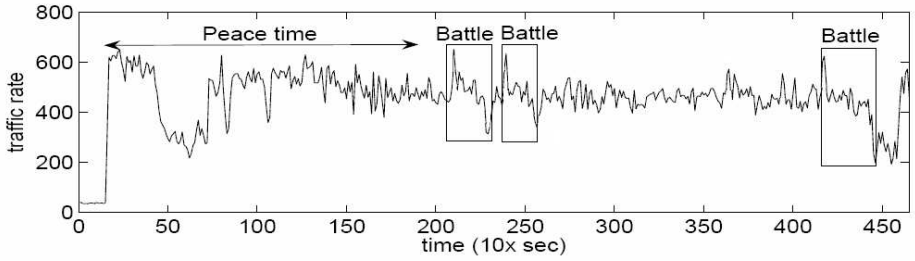


Figure 3.8: Cossacks measurement traffic intensity (packets/10 sec)

the player has a wider variety of choices).

In this section we will show that in case of RTS games, the number of commands the network has to transmit is in parallel with the number of units the user can manage. On the other hand, the number of commandable units is in connection with the state of the player's economy. A good example for how these factors influence the network traffic rate can be examined in the case of battles. Cossacks [6] was chosen for traffic analysis because it operates with grand armies containing several hundred units. We created a packet capture and the ingame video was recorded in parallel. Later the actions during game play and the packet rate were correlated. The battles can take several minutes long and during this time, the user manages the unit formations, attacks or withdraws and during the battle the player will lose most of the units. It was expected to see these effects in game traffic.

Figure 3.8 shows that on 10 sec timescales, mainly the gamer actions and the game world status (the players built-up economy, army) defines the structure of the traffic rate function. During peace time the players can not attack each other, thus they only build economy. However, the initial phase of the game has several falls in the traffic, which is the result of the lack of resources of the players during game play and can not do anything but wait for the finish of the construction of a building. During the game play progression the players establish a stable economy and the network traffic becomes smooth.

The battles result in typical changes in the gaming traffic. In the initial phase of the battle, the armies are collected and the units are marched to the front line which generates a local peak in the traffic rate. During the battle, as the units die, the traffic falls parallel. The biggest traffic rate fall occurs in the case of the final battle when the army of one of the players is fully destroyed. After the battle the winner player occupies the loser's city with its surviving army. This implies low-rate traffic. The game soon ends and the collected statistics of the game play are sent, resulting in a higher traffic rate again. Comparing the reconstruction phase of the game when no battles but rebuilding of the economy takes place, there are no falls or peaks in game traffic.

3.2.3 Detection of war

Summarizing our findings in the previous section, the battles in the traffic rate of Cossacks induce an increase in the beginning of the battle due to the collection of available forces. The end of the battle induces a fall in traffic rate due to the mass losses in commandable units. This raises the question how general the problem is. The battle in an RTS game means a local peak in the traffic rate, then a decreasing trend ending with local minima.

Algorithm 5: RTS battle recognition algorithm

```
   Input: rate, time-window: N
   Output: out
 1 nx = length(rate); m = N/2;
 2 xv= (1: 1: length(rate))';
 3 allmean=mean(rate); for i=1:nx do
 4     if ((i − m) >= 1) & ((i + m) <= nx) then
 5         p = polyfit(xv(i-m:i+m),rate(i-m:i+m),1);
 6         f = polyval(p,xv(i-m:i+m));
 7         varianceV(i)=var(rate(i-m:i+m)-f);
 8         meanV(i)=mean(rate(i-m:i+m));
 9         out(i) = p(1);
10 varianceAvg = slidingavg(varianceV, N) /sliding average calculation/;
11 meanAvg = slidingavg(meanV-allmean, N);
12 outAvg = slidingavg(out, N);
13 outvar = slidingvar(out+min(out), N) /sliding variance calculation/;
14 varmin = maxima(-varianceAvg) /select local minimas/;
15 meanmax = maxima(meanAvg) /select local maximas/;
16 outmin = maxima(-outAvg) /select local minimas/;
17 for i=1:length(outmin) do
18     finds1 = find(abs((outmin(i) - varmin) < N));
19     finds1size = size(finds1);
20     if finds1size(1) > 0 & outvar(i) > 0.02 then
21         finds2 = find((abs(outmin(i)-meanmax) < N));
22         finds2size = size(finds2);
23         if finds2size(1) > 0 then
24             newout(outmin(i)) = 100 /mark battle/;
```

To grab these specific characteristics we calculated a linear regression in a sliding window for the traffic rate of the server (see Algorithm 5 for the Matlab [39] skeleton code of the algorithm in details). The indicators of the battle are the following properties:

- *Is there a slope?* – Local minima in the gradient (p_1): We searched the coefficients of a polynomial $p(x)$ of first degree that fits the data, $p(x(i))$ to $y(i)$, in the least squares sense. $x(i), i = n..n + k$ stands for the examined time interval where n is the start of the examined time interval, and k is the length of the sliding window; $y(i)$ is the traffic rate in the i^{th} time. The result $\vec{p}$ of

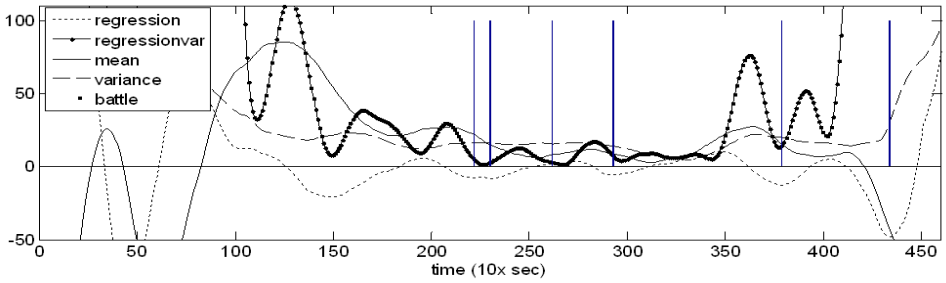


Figure 3.9: Analysis of Cossacks traffic and the effect of battles during gameplay /battles are indicated with the vertical poles/

$p(x) = p_1 x + p_2$ is a row vector of two-unit-long containing the polynomial coefficients in descending powers.

- *Is there an increase in the rate?* – Local maxima in the mean rate of the traffic in the sliding window
- *Does the linear fit well?* – Local minima in the variance of the difference between the fitted polynomial and original time series: $\min\{|y(i) - p(x)|\}, i = n..n + k$
- *Is there a unusual jump in the rate?* – The standardized variance of the gradient is above a limit: $\tilde{\sigma}\{p_1^i x\} > c, i = n..n + k$

In Figure 3.9 the analysis of the traffic of Figure 3.8 can be seen. It can be noted that at about 2300 sec there are two local minimas of the gradient function: local minima in the variance of the difference between the fitted polynomial and original time series and local maxima of the mean packet rate. The gradient variance has local maxima at about 2100 s. The effects do not occur in the same time and the sliding window shifts them away from each other and this has to be tolerated. We checked the constellation of all the positive occurrences of the properties in the length of the sliding window range, which was chosen to 30 time-unit (30x10sec).

3.2.4 Validation of war detection

During the validation we focused on the validation of the accuracy of the algorithm. With the validation such special cases can be revealed when the proposed algorithm makes mistakes. With the validation it is also possible to reveal how generally applicable is the proposed algorithm. The validation is difficult due to several reasons: the games can not be modified and extended to be capable of different logging activities and a mass number and wide variety of logged gaming activity from different players

is difficult to obtain. In order to overcome these difficulties the validation of the proposed method has been performed in several steps.

In the ***first step*** we created several active measurements in a controlled environment by capturing the generated network traffic of the game and manually log the time of the battles. After the measurement the proposed method was applied on the network traces. The battles which were indicated by the proposed method and the logged battle events were compared. Overlapping periods of battle periods in the validation and the results of the algorithm are regarded as true positives. In the validation phase 15 hours of RTS gaming traffic were collected and examined. In this phase of the validation 33 out of 36 battles (92%) were detected and 3 (8%) of the indicated battles were false positive. As this phase of the validation showed promising results we extended the validation process to be able to validate with more samples.

In the ***second step*** of the validation process we introduced an automatic method to quantify the size of the battles and to recognize battles automatically not only from the network traffic but from the game itself. It became clear that during gameplay it is not straightforward how a battle is characterized, in other words which events should be considered as battles: there are several cases when the units fire at each other, when players send scouts and they are caught, but these events are not huge battles just small conflicts. Our goal was to quantify the size of battles and this was done in the following way: the program structure of the Cossacks was examined and we managed to locate the list of sound effects of the game. All but unit dying effects were removed from the game to get rid of the e.g., unit control and management sounds. During play, beside the network traffic capture, the ingame sounds were recorded, which only contained the unit loss sounds due to our modification. The unit loss sounds are characteristic feature of a battle, furthermore, the power of the mixed sounds – as there can be more unit loss parallel – and how frequently these sounds occurred shows the size of the battle. The considerable size of the battle can be tracked by a simple limit on the power of the audio. As an example in Figure 3.10 the recorded audio and the calculated power levels of the same Cossacks gameplay of Figure 3.8 can be seen.

We created several measurements with Cossacks, but to examine how generally the method is applicable we extended the validation for different games. We used Spring [34] to validate thoroughly the proposed method in bulk measurements in which case the traffic is generated by different players. The extraction of the sound effect files was also possible in Spring thus the record of the sound effects was possible. To obtain bulk measurements we used [32] which is a collection of replay files of the game. Replays are automatically recorded for every game in Spring and the players are able to upload the interesting games to [32]. During the watching of the replay the original game is practically replayed with its sounds and as it is possible to host

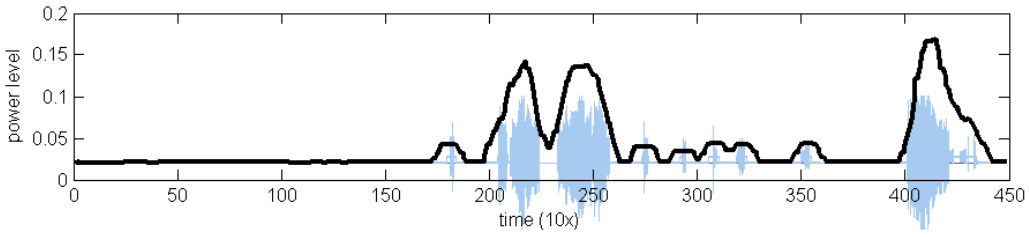


Figure 3.10: The recorded audio and the calculated power levels of the same Cossacks gameplay of Figure 3.8

the replay for others to join to see, thus the network traffic is also replayed. In this phase of the validation 33 out of 38 battles (87%) were detected and 4 (11%) indicated battles were false positives.

What we learned. It is clear from the validation that our proposed algorithm makes mistakes sometimes, thus such events as e.g., the loss of the scouts is hardly ever detected. It was experienced that the reason for the mistakes is that a significant number of units has to be commanded and lost during a battle to make the method capable of detecting the battle event. In Figure 3.11 the cdf of the power level of the audio of all the battles in the RTS measurements comparing to the power level of the audio of recognized battles can be seen. As the cdf plot of the power level of the recognized battles are under the cdf plot of the power level of all the battles occurred in the game it is clear that the proposed algorithm can mainly identify the big battles.

The other aspect which would cause the algorithm to make mistakes is such a play style which neglects the management of the army. Our method builds upon the significant effects of the micromanagement in the game. The lack of the micromanagement activity would result in smooth game traffic. The high detection ratio showed us that the existence of big battles and intensive micromanagement is a realistic precondition in the RTS games.

In the ***third step*** of the validation process it would be important to examine how the proposed method is applicable in general. We collected some traces from [15] and also created active measurements from several other games from the RTS genre. Cossacks [6], TA Spring [34], Warcraft 3 [15], Command & Conquer Generals [15], C&C 3 [3], Age of Mythology [15], Starcraft [15] were also examined. The bigger the armies in a given game, the more confident the method is in finding the battles. The main battles were found in all of the enumerated games by the method, but certainly not all smaller scaled confrontations. One exception is Starcraft, where the proposed method does not work as it has unique game traffic characteristics among the RTS games in the sense that its traffic is similar to an FPS game, thus the battles and

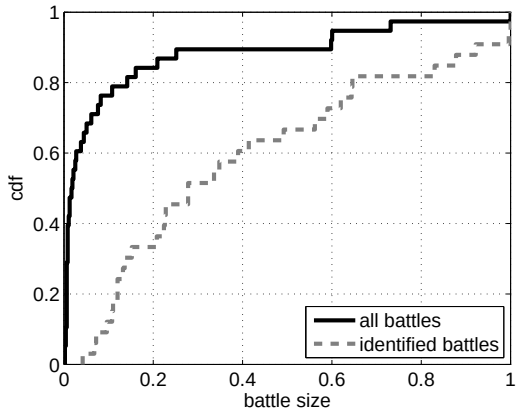


Figure 3.11: Distribution of the power level of the audio of all the battles comparing to the recognized ones

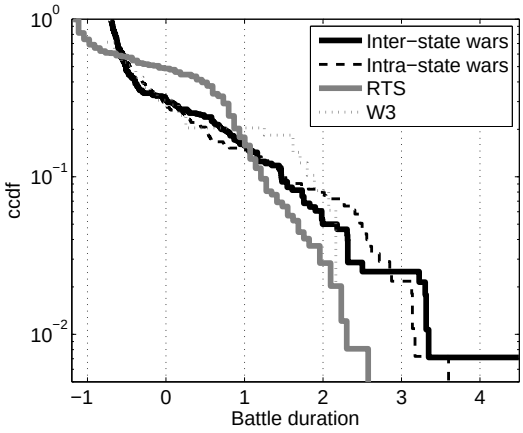


Figure 3.12: Normalized CCDF of battle durations

environment do not affect the rate. We are aware that in spite of the fact that the method works in most cases, this examination could be made more complete which can be the be the subject of further work.

3.2.5 Application – Comparison with real wars

A straightforward idea is to use parameters for player behavior simulations obtained from real world. We found that a possible candidate of an important simulation parameter is the battle length of the gaming world. We used this parameter and compared it to real world data.

We filtered out the gaming traffic from the total traffic of an operational broadband network with the method introduced in [C1] and analyzed the filtered traffic with the methods presented in the previous sections. The examined traffic is a three-day-long passive measurement, which contained about 20 Warcraft 3 flows and 30 other RTS flows (e.g., [6], [3], etc.) containing 296 battles total. In this section we discuss one of the examinable parameters as an example. The questions which can be raised and answered from the revealed data is not restricted to these parameters of course.

In Figure 3.12 the distribution of battle lengths in RTS games can be seen. The short duration battles dominate the datasets for the Warcraft 3 games. In other RTS games the distribution is more scattered along the different time lengths. We compared the characteristics of the observed RTS games to real world data. We obtained the data about the war activities of the real world from the Correlates of War project [108]. The homepage of the Correlates of War project [5] seeks to

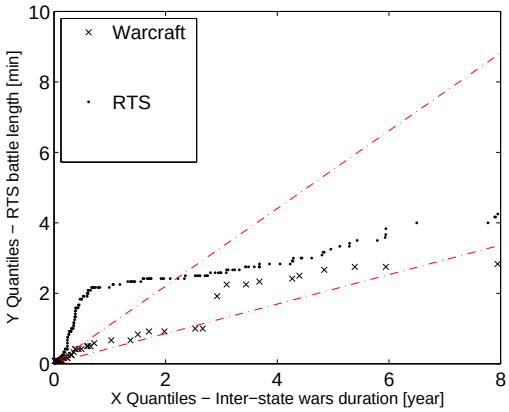


Figure 3.13: The qq-plot of real world battle durations and in-game battle durations

facilitate the collection, dissemination, and use of accurate and reliable quantitative data in international relations. We used the inter-state and intra-state dataset, which identifies interstate and intrastate wars and their participants between 1816 and 1997. In Figure 3.12 it can be seen that the short duration battles dominate the datasets both in the real and gaming world.

The ccdfs are depicted in lin-log scale and it can be seen that the tail of the distributions are exponential both in the gaming and real world. To compare the battle durations of the gaming and real world environment, the qq-plot of the two data series was depicted in Figure 3.13. There is a break point at 0.7 year of the real world and 2.25 minute of the gaming environment. If the data series are divided by these limits and replotted the two qq-plots it could be seen that the distributions are similar to each other in these intervals as they fall onto a linear.

Potential reasons. The potential reason for the short gaming session durations (see Figure 3.12) can be the difference of experience between the average or weaker than the average players. A weaker player is usually not able to withstand for long the attacks of a more experienced player. Comparing the tail of the duration of the real world battles and RTS sessions it is clear that renewal resources models the real world battles better than Warcraft 3 non-renewal resources as there is no such cut-off in real world cdf as in the case of Warcraft 3.

The exponential tail of the distributions in Figure 3.12 can be interpreted in the following way: when two armies fight against each other, when they both reach a 20% loss, both retreat and continue at a later time. A longer tailed distribution can occur in such a case that during the battle when one of the armies feel a disadvantageous situation it instantaneously retreats with as small loss as possible, so there is not significant loss in the armies thus the chance of a new battle remains the same as it

was before the battle.

The break point of the qq-plot in Figure 3.13 supports the idea that people tend to take part in only short duration battles as the long battles are expensive and risky. Other reason for the preferred short duration battles both in gaming and real world environment can be due to the difficulties in controlling the armies in the long run after the battle started and the armies smashed together. To overcome this, the armies are regularly retreated to reorganize the scattered soldiers both in the real world and gaming world environment.

3.3 Summary

In the first part of this chapter, the traffic of the MMORPGs is examined with special focus on the effects of player behavior at a macroscopic level on the gaming traffic. By understanding how the gaming traffic is influenced by the activity of the players and the changes in the gaming environment, detection methods based on time series analysis are introduced to grab specific events and states during the game play.

The proposed methods were applied on traffic traces obtained from live operational broadband networks in order to analyze the player behavior in real network situation. Both the user actions and the environment around the player were recognized in the MMORPG network traffic.

The collected data in the gaming environment was compared to data obtained from the real world environment. We revealed the LRD property of the gaming traffic and we also showed that the popular LRD explanation by heavy-tailed periods cannot be held. We also demonstrated that the player behavior affects the traffic characteristics at a macroscopic level.

Complex game server architecture is required by the growing number of Massively Multiplayer Online Real Time Strategy (MMORTS) games to make the enormous number of state information transmission feasible. The architecture design is supported by network traffic simulations based on accurate player behavior characterization.

The major contribution of the second part of this chapter is to make these simulations possible by extracting the necessary parameters with a method identifying battles from passive measurements. The proposed method is based on the analysis of game traffic characteristics. The method grabs the increase in bandwidth in the gathering of forces phase of the battle and the fallback in bandwidth due to the mass loss of units during the battle. The proposed method was validated in several steps and proved to be a general method to work well with the majority of games in the RTS game class.

The suggested method was also applied on traffic traces obtained from live operational broadband networks in order to analyze the player behavior in real network situation. It was found that the distribution of battle durations shows similarities in the gaming and real world environment. We demonstrated that the gaming environment models the real world well but it is not a complete copy of that. This means that parameters obtained from real world can be used to some extent but the most exact results can be obtained by the investigation of the gaming environment.

Another significant contribution of this chapter is in connection with traffic modeling and traffic profiling work. We showed that the player interaction can so strongly influence the traffic characteristics of a game that it can not be regarded as an approximately fix rate UDP stream as it is used in previous works but more detailed game genre specific traffic models are needed.

Chapter 4

Dynamic signature for Deep Packet Inspection [J2]

Current DPI methods search for fix byte signatures in the packet payloads. On the other hand, there are several protocol fields, e.g., especially in connection with gaming traffic, the player movement segments are always changing values – dynamic signatures. There can be found other dynamic signature types e.g., timestamps or sequence numbers in any general protocols.

To the best of our knowledge studies focusing on the structure of the packets of the game protocols do not exist yet. We examine how the effects of human behavior can be tracked in network traffic at packet level. Player behavior and movement related packets encode the ingame location of the players. We found that this packet level information shows similar statistical properties that characterizes human motion models.

The challenge was to specify the length of the fields, the direction of the fields and to differentiate them from a simple increasing sequence. There can be use cases when other methods can not deal with the content e.g., packets with Protocol Header Encryption, than a movement field. In DPI methods dealing with static fields, there is no recognizable fix signature in it.

4.1 Related work

In this chapter the protocols are analyzed at packet content level. As far as we know studies focusing on the structure of the packets of the game protocols do not exist yet. Tan et al. [115] and Liang et al. [99] extracted the structure of the gaming

protocol from the source code of the game [115] or manually with several empirical experiments [99]. In [107] Pittman et al. used a scripting language [40] which is supported by World of Warcraft to automate ingame activities and extracted the positions of every character they encountered during game.

The most related papers in literature focus on automatic signature generation with fix protocol signature subsequences. In [84] Haffner et al. explore automatically extracting application signatures from IP traffic payload content. In particular, they apply three statistical machine learning algorithms to automatically identify signatures for a range of applications. In a similar manner in [93], Kim et al. introduce a system which ranks packet content according to its prevalence, and only generates signatures as needed to cover its pool of suspicious flows. Consequently, it is designed to minimize the number of signatures it generates. Their offline evaluation on real network traces reveals that their system can be tuned to generate signature sets that perform well on worm detection. It is interesting to note that automatic signature generation can be extended for even polymorphic worms which are able to alter their form on the network to deceive pattern recognition engines. Such a method is presented in [98]. Ma et al. [100] present three classification techniques for capturing statistical and structural aspects of messages exchanged in a protocol: product distributions of byte offsets, Markov models of byte transitions, and common substring graphs of message strings. They compare the performance of these classifiers using real-world traffic traces from three networks in two use settings, and demonstrate that the classifiers can successfully group protocols without a priori knowledge. They analyzed common plain-text protocols like e.g., SMTP, DNS, SSL. Park et al. [105] used Longest Common Subsequence (LCS) for the signature extraction step. The LCS is extracted from sample flows to be the signature of the given application. The algorithm compares two samples to get the longest common subsequence between them, and then compares it with other samples iteratively to refine it. They managed to produce signatures for several P2P protocols as well e.g., LimeWire, BitTorrent, Fileguri. Our work focuses on the creation of an algorithm which can automatically grab the dynamically changing fields of the protocols. The requirements were to test player behavior in unknown protocols in an efficient way which resulted in a simplified character movement model comparing to player movement models in e.g., [115] for FPSs and [99] for MMORPGs.

Considering current DPI methods specific structures of the composition of fix byte signatures and 'do not care' segments are searched in the packet payloads (see Section for details). There are several cases when there is a variable in the packets at a fix position. Our idea was to focus on the 'do not care' segments and analyze them. This dynamic signature can be recognized by statistical methods.

The dynamic signature proved to be especially useful applying on game traffic. Wide-spread MMORPGs e.g., World of Warcraft [38] uses protocol header encryption

Figure 4.1: Dynamic signature in a packet

which makes it difficult to recognize them by current DPI methods. The dynamic signature analysis makes it possible to grab other features of the gaming protocol and recognize them even if it is unfeasible to construct a well-defined signature for the protocol headers. Games which transmit location information in coordinates were well recognizable by the proposed method. The payload segment which encodes the location of the players in the gaming environment shows such statistical properties that characterize human motion models. Our presumption was that such variables which statistically fit to motion models but actually containing not motion related information are rare. It is important to note that some games are evolving basic encryption or obfuscation of payloads. Payload obfuscation or a payload compression may modify the statistical nature of different fields in the payload. This is not the scope of our current work.

4.2 Definition of 'Dynamic signature'

We define the following three signature types: In a packet (see Figure 2.4 as an example) there can be the usual fix signatures – flags (F) –, 'do not care' segments (?) and dynamic signatures on every byte position. We define the following field types as a dynamic signature:

- *Strong correlation* (C): There is a strong temporal correlation in the specific byte segment among the sequence of the packets. The increments of the packet values are random.

- *Walk* (W): There is a strong correlation in the specific byte segment among the sequence of the packets. Further, the values in the sequence of packets are random. These two properties make the time series similar to real world human motion models, thus it is considered as walking. In our method this field is considered as a fractional Brownian motion process.

- *Sequence number* (S): These fields are used in protocols typically for e.g., sequence numbers or timestamps.

- *Fix* (F): These fields are used in protocols as fixed headers or flags.

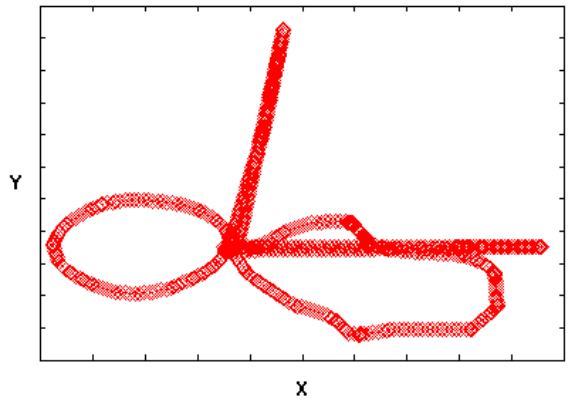


Figure 4.2: The coordinate values in the active measurement depicted in 3D and projected to XY plane

In the case of every dynamic signature the length of representation is important. In the example packet in Figure 2.4 the value in the 'walk' field is represented on four-byte-long.

An MMORPG example. We created several experiments with active measurements in World of Warcraft [38] to examine its traffic. We measured the traffic of World of Warcraft and created an active measurement in such a way that the player actions were planned before the measurements to let us synchronize the network traffic with user actions. The analysis showed the high occurrences of 34-byte-long packets suggesting the candidate packets responsible for character movement information. We performed several experiments to find out how the packet is partitioned and which fields of the packets refer to a specific data. In one of the active measurements we moved with the character from a well-defined starting point, for 1 min to north (according to the ingame compass), 1 min to south, 1 min to west and 1 min to east. In another active measurement we run around in a big circle and finally we checked if we take a bigger wandering and return to the staring point whether the coordinates are in agreement with our expectations.

Figure 4.2 shows the reconstructed original path of the player. One derivation of this experiment is that by starting and ending at the same coordinates with the character, we can be sure that the coordinates are non relative, thus not only the movement vectors are transferred to the server but they are absolute coordinates. The other gained information is the relation of the packet sending time with the moving coordinates and with the ingame timestamps.

The assumed partition of the packet fields and their function:

- *1-6 byte*: This is the protocol header encrypted part of the packet. This part contains the movement packet identification flag, which would ensure us that this is a movement packet.

- *7-10 byte*: These are some kind of flags, which rarely change, and the values they take are either 0 or 1.

- *11-14 byte*: This part seems to be an ingame sequence number or timestamp as it constantly increases.

- *15-18 byte, 19-22 byte, 23-26 byte*: One of the 3D coordinates used by the game.

- *27-34 byte*: We guess that this field is a character facing value.

4.3 Discussion on the 'Walk' signature and the player movement model

It was not straightforward that real world human walk models e.g., [96] could be applied to player behavior in gaming environment. The control environment (joystick, keyboard, mouse) can introduce constraint on the player behavior. The goal was to construct such a model which a) can be efficiently tested, b) grab the characteristics of coordinate changes of human motion, c) use few model parameters and d) not game specific. Complex multidimensional models which describe human behavior even more accurately are not suitable for this task.

Based on the main dependence characteristics of the character movement process our presumption was that fractional Brownian motion (fBm) can describe the changes of character movements. The intuition behind the fBm model was our expectation that the character position coordinates exhibit high positive correlations. Therefore a player keeps going in the same direction with high probability, thus a simple random walk (e.g., Brownian motion) cannot describe its behavior.

We have chosen two independent fBm models, where one fBm would be fitted for the X and another would be fitted to the Y coordinates. The modeling of the Z coordinate was neglected due to the fact that the terrain serves as a constraint in this case.

We think that our 'walk' model applies for a broad range of games because our fBm based model captures generally both the randomness and correlations of the players. Our model has a great potential for generalizations because tuning the parameter H (see Section 4.3.1) we can set different correlations.

4.3.1 Basic modeling definitions

A continuous time process W_t is called *fractional Brownian motion* (fBm) with parameter H, $0 < H < 1$, if W_t is Gaussian and self-similar. The increment process of fBm is called *fractional Gaussian noise* (fGn) [121]. The fGn exhibits positive correlations if $H > 0.5$ and we want to capture the important positively correlated character movement property by this process. This property means that if there is an increasing pattern in the previous 'steps', then it is likely that the current step will be increasing as well.

In order to validate our model assumption we have carried out a time series scaling analysis. We have used several tests but in this section we show only the R/S test [121] and wavelet analysis [58] results. The R/S test can be applied in the following way: given an empirical time series of length N ($X_k : k = 1, ..., N$), the whole series is subdivided into K non-overlapping blocks. Now, the rescaled adjusted range $R(t_i, d)/S(t_i, d)$ can be computed for a number of values d, where $t_i = \lfloor N/K \rfloor (i-1)+1$ are the starting points of the blocks which satisfy $(t_i - 1) + d \leq N$.

$$R(t_i, d) = max\{0, W(t_i, 1), ..., W(t_i, d)\} - min\{0, W(t_i, 1), ..., W(t_i, d)\}$$

where $W(t_i, k) = \sum_{j=1}^{k} X_{t_i+j-1} - k(\frac{1}{d}\sum_{j=1}^{d} X_{t_i+j-1})$, $k = 1, ..., d$. $S^2(t_i, d)$ denotes the sample variance of $X_{t_i}, ..., X_{t_i+d-1}$. For each value of d one obtains a number of R/S samples, which decreases from K for larger values of d, i.e., $d_{l+1} = md_l$ with $m > 1$, starting with d_0 of about 10. Plotting $\log R(t_i, d)/S(t_i, d)$ vs $\log d$ results in the R/S plot. Finally, a least squares line is fitted to the points of R/S plot, where both the R/S samples of the smallest and largest values of d are omitted. The slope of the regression line is an estimate for H.

Scaling properties of traffic can also be efficiently investigated by multifractal analysis via wavelet-based methods [58]. The discrete wavelet transform represents a data series X of size n at a scaling level j by a set of wavelet coefficients $d_X(j, k), k = 1, 2, ..., n_j$, where $n_j = 2^{-j}n$. Define the q^{th} order Logscale Diagram (q-LD) by the log-linear graph of the estimated q^{th} moment $\mu_j(q) = 1/n_j \sum_{k=1}^{n_j} |d_X(j, k)|^q$ against the octave j. Linearity of the LDs at different moment order q suggests the scaling property of the series, i.e. $log_2\mu_j(q) = j\alpha(q)+c_2(q)$ where $\alpha(q)$ is the scaling exponent and $c_2(q)$ is a constant. In our test results we plot $y_j = log_2\mu_j(q)$ for $q = 2$ which is called the second-order logscale diagram (LD).

4.3.2 Preprocessing of data

The filtered gaming traffic of an operational broadband network (see Section 3.2.5 for details) was analyzed with the methods presented in the previous sections (Section 4.3.1). An important assumption in statistical analysis and modeling is the

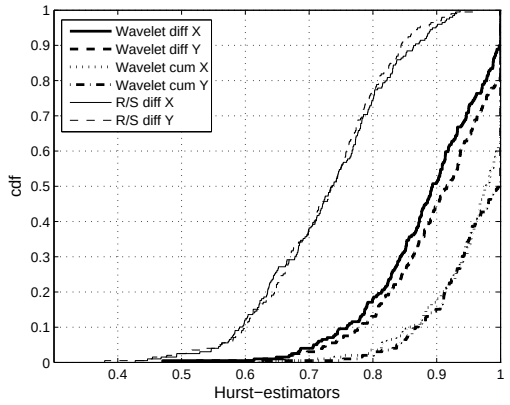


Figure 4.3: The cdf of the Hurst estimators for the different traces

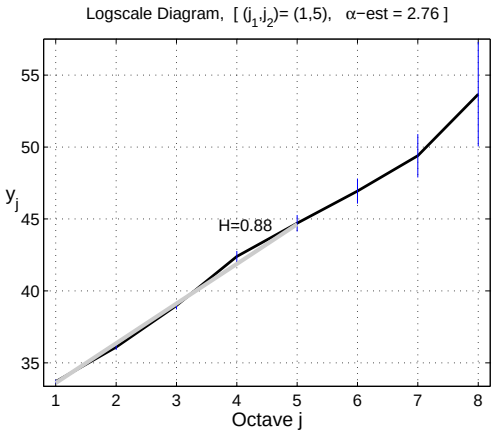


Figure 4.4: An example logscale diagram of the time series of X coordinates

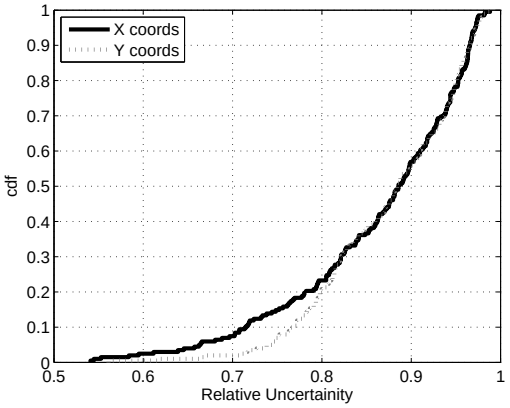


Figure 4.5: The cdf of the estimated Relative Uncertainty of the difference of the moving packets

stationarity. The character movement coordinates contain jumps which are the characteristics of non-stationarity processes. With the inspection of the movement packets of the World of Warcraft flows we found that there are several jumps in the coordinate values which suggest that the character changed its location from one place to another with very high pace. This can happen in case of the character used teleports. To create time series holding the stationarity property for analysis, we constructed first the time series of increments from the original (cumulant) time series of the coordinates. For this increment time series we applied a filter and removed the outliers

from the time series. Therefore we created a pre-processed stationary increment time series from our original data.

4.3.3 Hurst parameter estimation – Validation of the player movement model

First we applied an R/S test [121] for the increments. The test resulted in that the average estimated H parameter is 0.73 for both the X and Y coordinates calculated from 200 independent time series. The cumulative density function of the H parameter is depicted in Figure 4.3. As the linear fitting to the R/S plot is difficult to tune and need a lot manual refinement, we decided to apply the wavelet based method [58] because it is more robust than the R/S estimator. During the examination of the time series of the increments, the wavelet method estimates H by the $H = \frac{\alpha+1}{2}$ formula. The results for the different time series can be seen in Figure 4.3 where the average H-parameter is 0.95 for X coordinate and 0.96 for the Y coordinate time series.

An additional validation step has been performed by applying the wavelet based H-estimators on the original cumulant time series, which were constructed by summarizing the pre-processed increments. These pre-processed increments were obtained by the above mentioned filtering process in order to keep the stationarity property of the time series. The formula for the estimation of H changes to $H = \frac{\alpha-1}{2}$ during the examination of the cumulant data. The average H estimation resulted in 0.89 for the X coordinate and 0.9 for the Y coordinate time series. (The logscale diagram is depicted in Figure 4.4). It can be clearly seen from the figure that the cumulative character movement process is consistent with statistical self-similar process and supports the fBm model assumption.

Therefore our presumption that the character movements can be well modeled with two independent fractional Brownian motion for the X and Y coordinates has been validated by the statistical test results of real data. An appropriate H parameter is the average of all the estimation results from R/S, increment logscale and cumulant logscale analysis over 200 samples. It resulted in $H = 0.9$.

4.3.4 Filtering out non-random processes

An important thing to note in connection with the estimated Hurst parameter is that, such high values are typical of non-random processes e.g., in the case of a process which has fix increments. To significantly differentiate this case from the one where randomly distributed values occur in the increments, the randomness of the values in the timeseries is examined. In the case of a moving packet, if the

difference of two neighboring coordinates were examined it is likely that the size of the character step would occur most of the cases. Therefore a sliding window has been defined and the difference of the coordinates of the first and last element of the sliding window is calculated. These difference values construct a time series, which has been grouped into bins to make it possible to properly estimate the *Relative Uncertainty* with the method which was used in [124], [95]. Suppose we randomly observe X for m times, which induces an empirical probability distribution on X, $p(x_i) = \frac{m_i}{m}, x_i \in X$, where m_i is the frequency or number of times we observe X taking the value x_i. Empirical entropy of X can be calculated by $H(X) = -\sum_{x_i \in X} p(x_i) log p(x_i)$. Let N_X denote the number of discrete values X can take. The maximum entropy is $H_{max}(X) = \log \min\{N_X, m\}$. *Relative Uncertainty* can be defined as $RU(X) = \frac{H(X)}{H_{max}(X)}$. With this normalization, the value of Relative Uncertainty is in the $[0, 1]$ range. The higher values mean less predictable behavior, the lower values mean more deterministic behavior. In particular $RU = 0$ means that X has only one possible value, while $RU = 1$ means that X is uniformly distributed.

In Figure 4.5 it can be seen that the average of the RU values is 0.87, and there is not any occurrence of the RU below 0.5 meaning that the high value of the estimated RU is a strong proof of the fact that differences of the coordinates in the moving packets are randomly distributed.

4.4 'Dynamic signature' construction algorithm

In the previous section, a model for the character position has been constructed. This model makes it possible to introduce a method in this section to identify the moving packets based on time series analysis in MMORPGs. The presented method is not restricted for this purpose. It can be applied more generally, where the same packet structure is used for transmitting the same kind of information.

4.4.1 Temporal time-series construction

The working mechanism of the constructed algorithm (Algorithm 6) is presented on a World of Warcraft traffic measurement. The preconditions during the examination of a set of possible moving packets are that it is neither known if they are really moving packets nor the fields where the different values take place. During the examination of the values it is critical to know on how many bytes the values are represented. The presented algorithm steps through all byte positions and creates time series of all possible byte lengths (Algorithm 6 Steps 3,4).

In Figure 4.6 an example can be seen describing how the timeseries are constructed from the raw packet data for the analysis (Algorithm 6 Steps 5-7): when the number

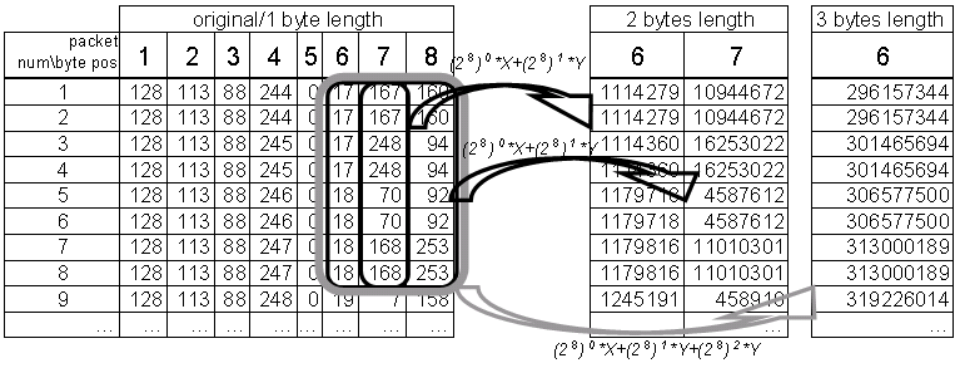


Figure 4.6: An example describing how the timeseries are constructed from the raw packet data

is considered to be represented in 2 bytes, regarding one row, the values in the 2 bytes representation output in the 6^{th} column can be calculated by the $(2^8)^0 * A + (2^8)^1 * B$ formulae where A is the value in the 6^{th} column and B is the value from the 7^{th} column. The result of our model (see Section 4.3) was that we can consider the coordinate values as the result of a fractional Brownian motion with H close to 1. Therefore there is a strong correlation between the values of the same fields in the time series. The same test should be performed during the search for this structure in an unknown traffic. We knew that the original fBm in the examined World of Warcraft traffic was represented on four bytes. We examined how the test results altered if the original $fBm \sim \sum_{n=0}^{i} (2^8)^n X_n$, where i is the length of byte representation of the value, is decomposed to the X_n components.

4.4.2 H-parameter examination – Application of the player movement model

The algorithm searches for fBm-process candidates (Algorithm 6 Step 9). We made several simulations regarding the Hurst-parameter test results of the fBm process decomposing it into different byte-representation ranges. Figure 4.7 shows the result of such simulation. The original timeseries is an fBm process with $H = 0.8$ parameter. It can be seen that the variance of the fBm process determines the range where the fBm properties can be observed after the decomposition. As the original process was tested on its original representation range, the H-parameter testing method finds the same H-parameter independently from the variance. The tested H-parameter of the $(2^8)^0$ component falls below 0.5 in the case of $\sigma > 30$, while the $(2^8)^1$ component tested H-parameter falls below 0.5 approximately in the case of $\sigma > 10^4$.

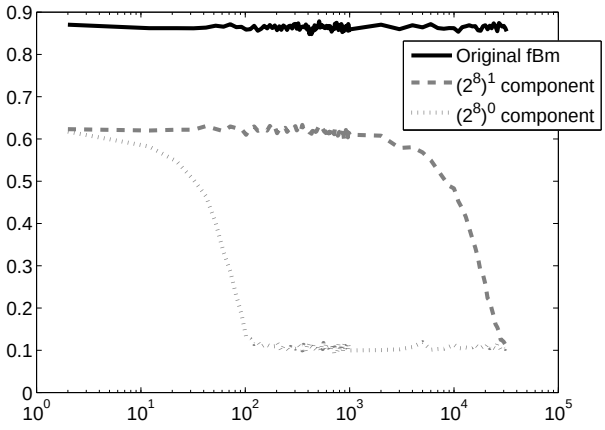


Figure 4.7: The H-test results of an fBm process in the function of variance

Another important thing to note is that the H-value of the original process is always higher than the decomposed one. This means that even if the variance of the tested fBm process is so low that it falls in the range of a smaller component, considering all components the test results in a better fit for the original process meaning a higher H-value.

If we examine the H-test results on the different byte positions with different length of representation of the World of Warcraft movement packets (see Figure 4.8) it can be seen that considering the values as one-byte long numbers there is no proper H-parameter ($0.5 < H < 1$). Considering the values at least two-bytes long, the proper H-parameter appears at the 16, 20, 25 positions, meaning that the 16-17, 20-21, 25-26 bytes are proper candidates of walking fields. Checking the H-parameter in the next row, meaning that considering the representation longer in 3 bytes we can see the same H-parameter for the 16-18, 20-22 positions but a non-proper value for the 25-27 bytes. This means that considering them longer does not deteriorate their fBm property. World of Warcraft represents the coordinate values in the little-endian form, thus the least significant byte is in the left most position. If we take into consideration that the coordinates are represented in four-bytes-long, we can conjecture that the least significant byte varies the most, and the others far less, going toward the most significant value. Considering the byte representation longer in our case means that:

1. If the variance of the process falls into a more proper component, the H-value will fit better to the original process

2. If the new byte value does not alter the H-parameter that means a constant level-shift in the considered time-series

	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31	32
1	-1.41	-1.40	-1.22	-1.39	-1.60	-1.54	-0.87	0.00	0.00	0.00	-1.09	0.11	0.23	0.00	-1.09	0.05	0.09	0.00	-1.12	-0.17	0.06	0.00	-1.06	-1.00	-0.03	0.03	-1.14	-1.13	-0.52	-0.40	-0.11	-0.17
2	-1.40	-1.22	-1.39	-1.60	-1.54	-0.87	-0.87	0.00	0.00	-1.09	0.11	0.31	0.23	-1.09	0.05	0.86	0.09	-1.12	-0.17	0.90	0.06	-1.06	-1.00	-0.03	0.96	-1.14	-1.13	-0.52	-0.39	-0.11	-0.18	0.00
3	-1.22	-1.39	-1.60	-1.54	-0.87	-0.87	-0.87	0.00	-1.09	0.11	0.31	0.31	-1.09	0.05	0.87	0.86	-1.12	-0.17	0.91	0.90	-1.06	-1.00	-0.03	0.97	-1.14	-1.13	-0.52	-0.39	-0.11	-0.18	0.00	0.00
4	-1.39	-1.60	-1.54	-0.87	-0.87	-0.87	-0.87	-1.09	0.11	0.31	0.31	-1.09	0.05	0.87	0.87	-1.12	-0.17	0.91	0.91	-1.06	-1.00	-0.03	0.97	-1.14	-1.13	-0.52	-0.39	-0.11	-0.18	0.00	0.00	0.00
5	-1.60	-1.54	-0.87	-0.87	-0.87	-0.87	-1.09	0.11	0.31	0.31	-1.09	0.05	0.87	0.87	-1.12	-0.17	0.91	0.91	-1.06	-1.00	-0.03	0.97	-1.14	-1.13	-0.52	-0.39	-0.11	-0.18	0.00	0.00	0.00	0.00
6	-1.54	-0.87	-0.87	-0.87	-0.87	-1.09	0.11	0.31	0.31	-1.09	0.05	0.87	0.87	-1.12	-0.17	0.91	0.91	-1.06	-1.00	-0.03	0.97	-1.14	-1.13	-0.52	-0.39	-0.11	-0.18	0.00	0.00	0.00	0.00	0.00
7	-0.87	-0.87	-0.87	-0.87	-1.09	0.11	0.31	0.31	-1.09	0.05	0.87	0.87	-1.12	-0.17	0.91	0.91	-1.06	-1.00	-0.03	0.97	-1.14	-1.13	-0.52	-0.39	-0.11	-0.18	0.00	0.00	0.00	0.00	0.00	0.00
8	-0.87	-0.87	-0.87	-1.09	0.11	0.31	0.31	-1.09	0.05	0.87	0.87	-1.12	-0.17	0.91	0.91	-1.06	-1.00	-0.03	0.97	-1.14	-1.13	-0.52	-0.39	-0.11	-0.18	0.00	0.00	0.00	0.00	0.00	0.00	0.00

Figure 4.8: The H-test results on the different byte positions with different length of representation of the World of Warcraft movement packets

3. The most-significant value is not an fBm process any more thus the H-parameter will drop.

In the 16-18, 20-22 positions the case 2. occurs, while in the 25-27 byte positions, the case 3. Moving one column left, checking what happens if the longer byte-representation is considered from a former position, we can see the increase of the H-parameter falling into the case 3. Considering them longer (15-4), the case 2. occurs, but for even longer byte representations case 3. occurs for (15-5), (15-6). When the most-significant bytes of the considered time series is again a real fBm process – which is the next walking field of the World of Warcraft packet – then the H-value becomes proper again. These considerations are implemented in the proposed algorithm in Algorithm 6 Steps 22-30.

4.4.3 Analysis of randomness

High H-parameter can be obtained by testing the timeseries with fix increments which are part of the gaming protocol message like sequence numbers or timestamps. The H-parameter testing has to be extended with the examination of the randomness of the candidate byte positions to differentiate a fBm-process and a sequence number. To do this, the Relative Uncertainty (for details see Section 4.3) of the differences of the coordinates in a sliding window is calculated, and if the value is less than 0.8 then it is a sequence number with fix increments otherwise if RU is high it is considered as a moving packet (Algorithm 6 Steps 13-17). The sliding window is necessary as neighboring values can be fix in case of walking behavior – size of the step –, but the coordinate distance differences in a sliding window show randomness due to the movement in several dimensions. Regarding the size of the sliding window, the start and the end values of the window should be big enough to avoid such a situation that a character moving into the same direction creates fix increments among the coordinates and the algorithm reaches a false negative decision. This consideration results in a sliding window size bigger than 2 packets covering a few minutes period in the movement of player e.g., about 120 packets which means 1 minute movement in the same direction. In World of Warcraft, in case of moving in the same direction

the client sends a packet in every 500 msec. In case of changes of direction the client sends packets in 50-100 msec intervals with the current orientation of the player. Note that the gaming environment is full of objects and monsters which the players try to avoid thus moving into the same direction in minute time scales is unlikely.

To differentiate the sequence numbers from the fix values in a specific position, the original time series is grouped into bins – the number of bins is selected in the function of the examined range of values – and the relative uncertainty is calculated for the binned values. If the RU is low then the examined values can be regarded as fix values (Algorithm 6 Steps 18-20). The size of the bins should be a trade-off between the processing speed and the accuracy of the estimation. In my specific implementation I fixed the number of bins to 100 resulting in varying value ranges covered by the bins.

Finally, the still empty fields are filled with the longest non-walking type fields from the state and H-parameter test tables (Algorithm 6 Steps 31-36).

4.5 Application and validation – Signatures of some popular applications

The method was validated with a common-practice applied in other papers [84], [93] as well by analyzing open protocols, the RTP and the Gnutella protocol, which have several dynamic fields in the protocol.

Several active traces of different applications have been examined by the introduced algorithm. The list of results can be seen in Table 4.1. The meaning of the columns of Table 4.1 are the following:

1. *Application* – The first column refers to the examined application
2. *Size* – The second column refers to the total length of the signature
3. *Signature* – The third column shows the constructed dynamic signature
4. *Endianess* – The fourth column shows whether the value is represented in **B**ig or **L**ittle endianess
5. *Ratio* – The fifth column shows the ratio of this packet comparing to the total number of packet in our traces of the given application

First, several games from the FPS, RTS and MMORPG genres have been examined. World of Warcraft [38] was the first target of our inspection and beside the 34-byte-long movement packet which was thoroughly examined in Section 4.2,4.3,4.4,

Algorithm 6: Dynamic signature construction

```
   Input: M[i][b]:=packets for examination, i: the i^th packet, b: the byte position in the packet
   Output: Out_state = ∅ /Output field type/, Out_length = ∅ /Output field length/
 1 H_o = ∅ /Hurst-test results/;
 2 S = ∅ /Guess for the field/;
 3 for w = 1, w < 8, w++ do
 4     for k = 0, k < b, k++ do
 5         X = zeros(size(M[:]));
 6         T = M[:][k : k + w − 1];
 7         for q = 1, q < w, q++ do
 8             X = X + T(:, q) ∗ ((2^8)^(q−1));
 9         H_o[w][k] = waveletestimation(diff(X));
10         S_temp := 0;
11         if 0.5 < H_o[w][k] < 1 then
12             S_temp := 1 /fBm candidate H/;
13         RU_diff,window = calc_RU(diff_window(X)) (Section 4.3);
14         if S_temp := 1 AND RU_diff,window > 0.8 then
15             S_temp := 4 /walking/;
16         else if RU_diff,window < 0.5 then
17             S_temp := 2 /sequence number/;
18             RU_diff = calc_RU(create_bins(X, (2^8)^w/100));
19             if RU_diff < 0.2 then
20                 S_temp := 1 /fix/;
21         S[w][k] := S_temp;
22 for w = 1, w < 8, w++ do
23     for k = 0, k < b, k++ do
24         if S[w][k] == 4 then
25             if S[w − 1][k + 1] < 4 OR (S[w − 1][k + 1] == 4 AND S[w − 1][k + 1] < S[w][k]) then
26                 Out_state[k] := 4;
27                 Out_length[k] := w;
28                 for j = k + 1, j < k + 1 + w, j++ do
29                     Out_state[j] = 0;
30                     Out_length[j] = 0;
31 for k = 0, k < b, k++ do
32     for w = 8, w > 0, w−− do
33         if 0 < S[w][k] < 4 then
34             if !conflict(Out_state, Out_length) then
35                 Out_state[k] := S[w][k];
36                 Out_length[k] := S[w][k];
```

another packet of 71-byte-length showed correlation structures. Our active measurements showed that this packet is in connection with the visit of mission-giving non-player characters in the gaming world. When the player approaches to such a character and clicks on it, the mission which can be given by that character is sent by the server. The correlation is due to the closeness of these mission givers around a specific location e.g., in a city, and as the players visit them one by one, it induces a correlation in the transmitted coordinates. Other MMORPGS were also examined

Application	Size	Signature	Endianess	Ratio
Age of Mythology [1]	10	?{1} CCCCCCC{7} SS{2}	B	19%
Command & Conquer 3 [3]	12	FFFFCCCCCCC{11} ?{1}	L	25%
Command & Conquer 3	16	???W{4} CCCWW{5} CC{2} ?CCC?{5}	L	18%
C&C Generals [4]	23	WW{2} ?C{2} SSSSSSS{7} FCCCCC{6} CC-CCC{5} ?{1}	L	19%
C&C Generals	31	WW{2} ?C{2} WCCCC{5} FFCCC{5} ??CCW{5} FC{2} C{1} ?{1} CCC{3} CCCC{4} ?{1}	L	16%
Cossacks - Back to War [6]	32	CCCCC{5} FFCC{4} FC{2} FC{2} F{1} ?{1} CW{2} CC{2} ?CCC??CC?CC??{13}	L	4%
Cossacks - Back to War	18	W{1} CCWWWWWWWC{10} SSS{3} CC{2} ??{2}	L	93%
Gnutella [11]	23	?{1} SSSS{4} CCCFFFFF{8} ?{1} F{1} ????????{8}	B	80%
MSN Messenger [17]	varies	??{2} SS{2} CCCW{4} CCCCCCC{7} ?{1} C{1} ???{3}	L	100%
Silkroad [54]	6	CCCC{4} ??{2}	L	5%
Silkroad	20	?{1} CC{2} ????W???C{9} C{1} ?{1} CC{2} CC{2} ??{2}	L	4%
Silkroad	30	??????W{7} WC{2} ??????W{7} C{1} ??????{6} C{1} ??C{3} C{1} ??{2}	L	11%
UT2003 [46]	20	W{1} ??{2} C{1} ???{3} WC{2} ???CC{5} CC{2} C{1} ???{3}	L	17%
UT2003	22	WWWWC{5} ?W???W?WW??C?{13} CCC{3} ?{1}	L	18%
World ot Warcraft [38]	34	??{2} C{1} ?{1} CCCWWW{6} WWWW{4} WWWW{4} WWWW{4} WWWW{4} WWWWCCC{7} ?{1}	B	25%
World ot Warcraft	71	????{4} CCCCC{5} WCCC{4} ??? {3} C{1} ????W?{6} C{1} CC{2} CCCFFFFFFF{10} ??{2} C{1} CC??{4} CFFFF{5} CC?{3} CCW{3} ???{3} CC{2} C?{2} CW{2} ???{3} CCC{3} ?{2}	B	4%

Table 4.1: Application signature patterns /C-strong correlation, F-flag, S-sequence number, W-walk, ?-not specified/

by the algorithm, and Silkroad [54] was found to contain a fBm structure. It is interesting to note that the values of the byte positions which were found to be fBm-like do not show such characteristics as an absolute position value which was found in World of Warcraft. We conjecture that these packets contain the movement vectors not the actual positions.

Games from the First Person Shooter (FPS) genre have been also examined by the introduced algorithm. FPS games are typical of transmitting movement vectors rather than absolute positions. A good example was found during the examination of the packets of UT2003 [46] (traces are available at [42]).

Games from the Real Time Strategy (RTS) genre have been also examined by the introduced algorithm. Age of Mythology [1], C&C Generals [4] and Cossacks [6] contain packets containing timestamps or sequence numbers which are responsible

for maintaining the synchronization among the gaming clients. Beside this there are other type of messages in the packets which seem to be in direct connection with the gaming environment itself: probably the number of active units in the gaming environment as it shows an increase overall the game session.

Our method can examine non-gaming applications as well. The traffic of a Gnutella [11] client has been examined and it was found that there are similar packets to the gaming protocols in terms of having the same size and same packet structure. Two UDP packets of Gnutella were found to contain the correlation characteristics in the time series of the packets.

Other applications generating varying packet sizes are also examined. MSN Messenger [17] VoIP traffic can be analyzed by selecting one direction of the flow, than the longest packet of the flow is selected and the smaller packets byte values are filled with uniformly distributed random values. In this way the dynamic signature search can be executed on the packets having varying size. MSN Messenger works with RTP messages thus it is also a good candidate to validate our algorithm. The algorithm finds the first two bytes of the RTP message where several flags take place. The 3-4 byte is the sequence number in the RTP message and it is found to be fBm-like according to the algorithm. The next 4 byte is also found to be fBm-candidate: the 5-8 byte is the place of the timestamp. The next 7 bytes are considered as coherent: the 9-11 byte is the place of the synchronization source identifier, which is a fix values on its own, but 3 additional bytes are added to this by the algorithm. Examining the payload of the MSN RTP packets, the first byte of the RTP payload seems to be fix, and the second and third byte varies in small steps. Interesting to note that there are other two byte fBm-like values at the 34-35, 36-37, 58-59, 82-83, 84-85 bytes of the MSN RTP packet. We conjecture that this property is due to the used codec.

4.6 Summary

The contribution of this chapter is the introduction of a novel model and algorithm to extend the Deep Packet Inspection traffic classification method. We focus on the analysis of variable length byte signatures. The model captures the variation of the dynamic byte segments and provides parameters for the algorithm.

The proposed algorithm considers the packets of the same flow as a time series and examines the H-parameter of the values of the different byte positions. The algorithm constructs protocol specific signatures of the examined traffic, which makes it possible to identify the protocol in a traffic measurement even in those cases when the signature in a protocol is not a fix value. As a proof-of-concept several proprietary gaming traffic and known traffic types have been examined with the introduced algorithm and the existence of the fBm-like structure is proved to exist in several cases.

Chapter 5

Wire speed traffic classification with parallel architecture [J3]

Certain applications, most notably belonging to the class of peer-to-peer (P2P) applications, are difficult to identify by signature search either because there are no appropriate signatures, or the signatures are difficult to match. Connection pattern-based methods, e.g. [91, 16] provide a light-weight, albeit heuristic solution to this problem.

In theory signature search and connection pattern-based methods could be combined to increase classification accuracy: finding signatures need Deep Packet Inspection (DPI), and if no signatures are found, the connection pattern method can be used. Or vice versa, if DPI marks the traffic of a host on a specific port as P2P, further connections to this host-port pair are very likely to be also P2P even if DPI has no results for them [113].

A serious downside of this combined solution is that it is not light-weight any more and raises considerable challenges to implementation. Therefore, extending a simple flow collector system with traffic classification capabilities needs more computing power than what CPUs can offer. Today there is a trend to use commodity hardware e.g., [33, 23] as it has low price/performance ratio, short development time and it is easy to get familiar with. The requirement for a dedicated hardware would slow down the wide-spreading of a particular method.

Not long ago, Graphical Processing Units (GPUs) have also become well programmable computing elements [8]. Furthermore, the current paradigm shift in processor technology towards many core architectures[1] has already occurred in graphical

[1]The terms many-core and massively multi-core are used to describe multi-core architectures with an especially high number of cores (tens or hundreds). In case of many-core processors the number

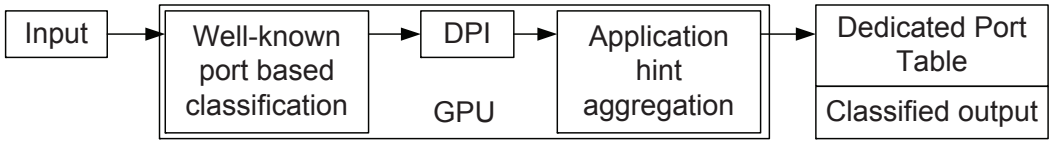


Figure 5.1: Steps of traffic classification with GPU

processing units a few years ago. Thus highly parallel algorithms can be experimented there which will be later well suit for general processor architectures as well when they become many core architectures. This chapter explores the conceptual and architectural differences between GPU and CPU based traffic classification.

This chapter addresses certain, key algorithmic challenges of such a combined solution, namely high-speed signature search and connection pattern-based classification. Beyond describing these tasks, we show how to implement them on GPU and present measurement results on the efficiency of the proposed system.

The contributions of the chapter are the following (see also Figure 5.1):

- We propose a memory optimized technique for signature matching. It involves a modification of the Zobrist hashing algorithm [129] to encode the application signatures. We store the dictionaries in the cache of the GPU and apply the same encoding to check which application a packet may belong to. Due to the compactness of the hashing algorithm the data sets of the proposed DPI method reside in the fast cached memory of the GPU providing high efficiency, in contrast to [19] where the signatures do not fit to the cache. The details of the method are presented in Section 5.1.2.

- We propose a fast-speed aggregation and update method to support connection pattern-based identification. We aggregate all the information necessary for the classification of flows into one data structure providing the best-available hint all the time. A Dedicated Port Table (DPT) stores the number of times a specific host-port pair was identified as the source or the destination of a specific application. In order to avoid a per-packet update of DPT, we propose to aggregate this information in the GPU by an extension to the parallel prefix scan [85]. This method provides massive parallelization and high efficiency with GPU. The details of the method are presented in Section 5.2.

Finally, Section 5.1.7 and 5.2.4 presents evaluation of the GPU based algorithms. We show that we can possibly reach beyond 6 Gbps traffic classification rate using commodity hardware.

---

of cores is so large that traditional multi-processor techniques are no longer efficient.

5.1 Deep Packet Inspection with Graphical Processing Unit

In order to perform DPI a common solution is to express patterns as sets of regular expressions and convert them into finite state automate (FSAs), mathematical models that represent (potentially infinite) sets of strings. The behavior of FSAs is easy to emulate on computing devices to perform the actual matching procedure. Regular expressions are supported by the vast majority of pattern-matching tools and represent the de-facto standard in the field. Two different kinds of finite-state automate are known from the literature, deterministic (DFA) or non-deterministic (NFA). While automate theory proves them equivalent in terms of expressiveness, their practical properties are different: NFA traversal includes, by definition, non-deterministic choices that are hard to emulate on deterministic devices such as all current processors; on the other hand DFAs, while fast to execute, can be less space-efficient, requiring very large amounts of memory to store certain peculiar patterns that are rather common in practice (the so-called state space explosion) [61]. In general, software-based NFA implementations suffer from a higher per-byte traversal cost when compared with DFAs: intuitively, this is because multiple NFA states can be active at any given step, while only a single one must be considered when processing DFAs. So far the focus of software research has mainly been on DFAs as they provide a relatively easy way to achieve fast processing speeds and high throughputs; there have been many efforts aimed at solving the inherent downsides of this model and avoid the aforementioned memory explosion. At the same time, NFAs have often been relegated to the design of hardware devices (e.g. FPGAs) where it is possible to easily emulate their behavior, and for use where high throughput is not the primary concern (e.g. many general purpose pattern-matching libraries). [18]

In our work we applied a different approach to achieve high packet processing speed by introducing restrictions on the grammar the method is capable of searching for. In order to achieve high processing speed with a GPU-based solution, the memory access overhead should be minimized. Our goal was to store the signatures on a minimized memory footprint to achieve this. State machines are processor-efficient methods for signature matching but not space-efficient [19]. Use of hashes results in data compression, but it is difficult to perform wild-card character matching. E.g., typical solutions store the same signature with all the possible wildcard values in the hash [76]. This results in the boom of the hash table and in false positive hits as well.

The fast cached memory of the GPU is small, thus to make it possible to store a high number of signatures compression is desirable. The compression has to support wildcards.

5.1.1 Related work

According to [65], the most resource consuming task is signature matching in traffic classification systems. String matching can be accelerated with ASICs, FPGAs [77, 119] and previous generations of videocards [87, 89, 109] (as texture operations), but they are difficult to modify and extend with new signatures and functions, which would be essential for traffic classification systems.

In [19], the deterministic finite automate (DFA) and the extended finite automate (XFA) based signature matching was analyzed. The authors found that the G80 GPU implementation was 9x faster than the Pentium4 CPU implementation. They emphasized the problem that data structure of the automate (in the order of MBytes) does not fit in the cache of current GPU architectures which would be essential for optimal operation. The packet processing speed of this solution was about 80,000 packets/sec (with FTP, SMTP and HTTP signatures).

In [116], the signature matching of an Intrusion Detection System (IDS) was done with GPU providing system throughput of 2.3 Gbps with synthetic traces and 600 Mbps with real traffic. In [117], Vasiliadis et al. further developed the work of [116] by reimplementing the DPI engine to the new CUDA architecture [8]. The overall system throughput increased by 30% on real traffic compared to [116]. [123] further refined the problem of load balancing the signature matching procedure of a large dictionary for multiprocessors. They proposed pattern set partition and input text partition together to fully utilize GPUs during the pattern matching.

Note that in [19, 116, 117, 123] the main tasks were to implement IDS on the GPU. This requires (a) the recognition of a huge set of protocol signatures with their numerous exploitation signatures, (b) find the signature anywhere in the byte stream, and (c) false positive hit is not allowed. I.e., the proposed methods for traffic classification task should not be directly compared to the above methods. As it can be seen later, our methods outperform them in raw packet processing speed, but on a more focused problem set of traffic classification (see Section 5.1.7 and 5.2.5 for details).

5.1.2 Signature Matching in GPU

In case of DPI, application specific bitstrings are searched in the packet payloads. DPI is a well-parallelizable task considering either the analysis of several packets in parallel or several signatures on the same packet. Moreover, traffic classification has a more focused requirement set comparing to an IDS functionality:

- The 'only' goal is to identify the applications, so less signatures is needed focusing on handshake messages typically found in the first few packets of the

(a) Input application signature dictionary (S)

App#1	a?b
App#2	aaa
App#3	?a?
App#4	??a

(b) Bitmask of applications signatures (B)

101
111
010
001

(c) Alphabet-position dictionary (α)

	0	1	2
a	1100	1011	1000
b	1010	1110	1001

(d) Encoded signature database (E)

0101
1111
1011
1000

Table 5.1: Input data of the DPI method

flows. For example, it is enough to identify an SMTP protocol from the first `EHLO` message and there is no need to parse any of its later messages.

- In most cases the signatures can be found in the first few bytes, thus the method can focus only on them.

In order to achieve high processing speed with a GPU-based solution, the memory access overhead should be minimized. Our goal was to store the signatures on a minimized memory footprint to achieve this. State machines are processor-efficient methods for signature matching but not space-efficient [19]. Use of hashes results in data compression, but it is difficult to perform wild-card character matching. E.g., typical solutions store the same signature with all the possible wildcard values in the hash [76]. This results in the significant increase of the hash table size and in false positive hits as well. In the following paragraphs we propose an encoding method, which eliminates the above problems. I.e., the input and output data of the DPI process occupy only the fast access memory of the GPU and the method provides wildcard support.

5.1.3 Zobrist hashing

Our proposal applies the idea of Zobrist hashing [129]. This technique was developed for creating hash codes in board games and storing the states of the players. Each piece has a unique identifier corresponding to its actual position on the board. The hash code of the actual state is calculated by XORing the unique position identifier of each piece. Instead of storing the whole board in each step of the game, only the hash values should be stored. I.e., the hash itself stores the data and not only a pointer to an external data field.

5.1.4 Input data for string matching

The proposed string matching technique uses the following input data (see Table 5.1 for illustration).

- The application signature dictionary is denoted by S (see Table 5.15.0(a)). One element of this dictionary, S_i^j denotes the j^{th} character of the i^{th} application.

- The list of bitmasks of non-wildcard characters in the application signatures is denoted by B (see Table 5.15.0(b)). One element of this list, B_i^j denotes whether the j^{th} character of the i^{th} application is wildcard or not.

- The alphabet-position dictionary is denoted by α and represented as a matrix (see Table 5.15.0(c)). The rows of the matrix represent the characters of the alphabet. The columns represent the positions of the characters in the signatures. One element of this matrix, $\alpha^j_{S_i^j}$ denotes the character code of the j^{th} character of the i^{th} application. These codes are represented as random numbers with a bigger range than the size of the dictionary in order to minimize the collision of encoded signatures.

- The encoded signature database calculated from the above input data and denoted by E (see Table 5.15.0(d)).

The encoded signature database of a particular application is calculated by XORing together the codes of its characters if their corresponding bitmask is 1. In general it can be written as $E_i = \bigoplus_{j=0}^{|S_i|-1}(\alpha^j_{S_i^j} \bullet B_i^j)$. For illustration, let us see a specific example based in Table 5.1: e.g., `a?b` is encoded into: $1100 \oplus 1001 = 0101$.

5.1.5 The proposed string matching method

The proposed string matching method is as follows. Each thread of the GPU deals with the content of one packet. This choice is a direct consequence of the Single Instruction Multiple Data (SIMD) architecture. Further, the GPU can deal efficiently with thousands of lightweight threads [8]. This is a big difference to an e.g., x86 architecture where the threads are monolithic and both task switching and scheduling are complex tasks.

The content of packets is considered as character strings. Apply the same procedure on these character strings as on the application signatures: the codes of the characters are XORed together according to the corresponding bitmasks. If the encoded packet data matches with the corresponding encoded application signature, then the packet is considered to belong to the particular application. The hit by the DPI process sets the proper bit of the specific application to 1 in the Packet Hit List (see Figure 5.2 and Section 5.2 for further discussion).

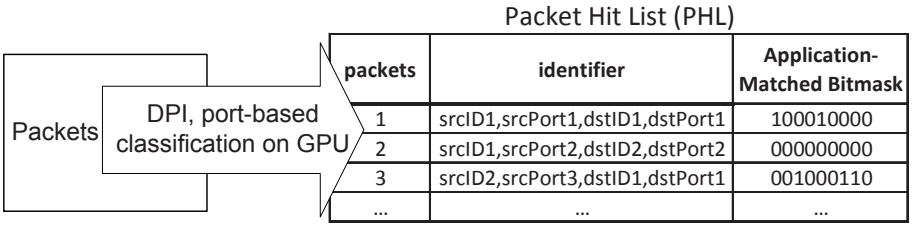


Figure 5.2: Input/output of signature matching on GPU

5.1.6 DPI data structures in GPU memory architecture

The *global memory* (uncached) space is filled with the array of network packets. During the initialization of each thread, the corresponding array of the packet bytes are copied from the global memory to the registers or to the *shared memory* (cached) of the thread. Thus global memory access does not reduce the speed of arithmetic calculations with the same data.

The *constant memory* space is cached so a read from constant memory costs one memory read from device memory only on a cache miss, otherwise it just costs one read from the constant cache. The pre-calculated input data structures are loaded into the constant memory space (the alphabet-position dictionary, the bitmasks and the encoded signatures). The allocable constant memory size is 64 Kbytes for the whole kernel in the test configuration (see Appendix). If we consider our example implementation where the signature database consists of 4 byte long values, then about 10 thousands of signatures could fit into the constant memory.

5.1.7 Validation – Evaluation of the GPU based DPI method

Compression vs. accuracy. The collision probability of the encoded signature database (i.e., misclassification of packets, false positive hit) can be analyzed as a function of the length of the signatures and their representations in the alphabet-position dictionary.

The size of the alphabet-position dictionary is $a*p$, where a is the possible number of characters and p is the possible number of positions (i.e., length of the signatures). To represent the signatures completely collision free, each character of the alphabet should be represented with $\log_2 a$ bit, and the character coding should be rotated according to the position. I.e., the size of one element of the dictionary should be $m \geq p \log_2 a$. In our case $a = 256$, i.e., $m \geq 8p$. In practice, however, much smaller m can provide negligible collision probability. Our experiments show in case of $m = 16$ (2 bytes) the collision probability falls below 10^{-4} independently of p if there is no

m	10	11	12	13	14	15	16
$p = 16$	0.2527	0.0867	0.0250	0.0069	0.0017	0.0004	0.0001

Table 5.2: The collision probability of the compression

more signatures than unique signature representations (i.e., here $2^m = 65536$). For an illustration, see Table 5.2 showing the collision probability in case of 1000 signatures of 16 characters for $m = 10 \dots 16$ (note that $2^m > 1000$ indicates $m \geq 10$), which is a feasible practical setup [19].

Performance. The proper initialization setup of the GPU kernel regarding the number of threads processed parallel by a multicore unit on the GPU has a high impact on the overall performance [8]. We created several measurements with different setups where the parallel processed packet batch size was varied from 16k to 500k packets. In case of CPU, the batch size would be the number of packets processed one-by-one without the interruption of any other operation, while in the case of GPU it means the number of parallel threads.

In the performance tests, the following implementations of the DPI task were analyzed:

- `regexp`: A perl-based implementation of the DPI method with standard regular expressions.
- `CPU`: The CPU-based version of the proposed DPI method.
- `GPU`: The GPU-based version of the proposed DPI method without shared memory caching of the packet headers and payloads.
- `GPU_shared`: The GPU-based version of the proposed DPI method with shared memory caching of the packet headers and payloads.

In our tests we replayed recent traces captured in live broadband mobile networks during busy hour traffic. The current traffic mixture of broadband mobile networks is similar to those presented in [36, 13]. The applied signature database contained an extended version of [2] consisting of approx. 300 signatures in total. The average signature length is 16 bytes characterizing 35 different application protocols grouped into 16 application types. Table 5.3 summarizes the performance tests focusing on the average packet processing speed. Note that the processing time is linearly increasing with the increase of input signature list. The slowest solution is `regexp`. Over this, `CPU` provides 5 times faster solution. `GPU` provides additional 2 times speed increase. Additional magnitude of speed can be gained by `GPU_shared` with exploitation of cached shared memory. In this latter case, the global memory is processed in small well-cacheable data blocks.

	regexp	CPU	GPU	GPU_shared
Average [thousand packets]	14	72	138	1844
Relative variance *f(batch size)* [%]	1.1%	3.1%	25%	0.2%
Relative variance *f(payload)* [%]	49%	36%	37%	27%

Table 5.3: DPI performance comparison – Packets processed per second

In practice, the batch size has only effect on the GPU [8]. Since the memory access overhead in the shared memory case is 2 magnitudes less than the global memory case, thus the effect is visible only in the GPU case without the shared memory. Regarding the payload dependence, the signature matching process for a specific packet ends in case of a hit in the signature database. Depending on the position of the signature in the signature database the threads finish in different times. According to [8], the threads in the same block are scheduled to run parallel thus they wait for each other to finish. This means that the slowest thread would determine the run time of the block. Therefore the more the parallel the solution is, the less variance occurs. Summarizing the results, the GPU-based shared memory version of the proposed method introduces a performance increase of two orders of magnitudes compared to the original regular expression based method. The average packet processing speed of the GPU is approx. 1,800,000 pkts/sec. This speed can be translated into network line speed by calculating with 500 byte average packet size which results in 6.7 Gbps speed.

5.1.8 Application

The method is capable of compressing a set of regular expressions and doing wire-speed DPI.

5.2 Flow classification based on the Dedicated Port Table

Some specific traffic types are difficult to be identified simply based on DPI without considering flow groups and their history. For example according to [91], in case of small P2P flows trying to establish connection to non-existent peers, partially encrypted P2P traffic or Skype. In this section we propose how to combine efficiently the DPI and port based classification methods for flow classification on GPU.

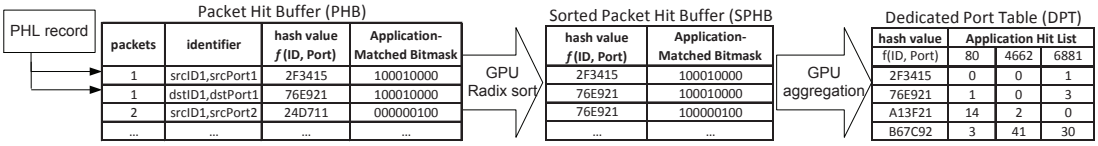


Figure 5.3: Dedicated port search

The **purpose** of the DPT is to store the application hints of host-port pairs based on their communication history and provide the traffic classification engine with the best available hint all the time. The DPT is represented as a list of records containing the number of hits per application for each host-port pairs. The DPT is created and updated as follows.

5.2.1 Starting from per-packet application hits

The starting point of the process is the Packet Hit List (PHL). Whether a packet comes from an application or not is a yes/no question, thus the PHL is represented as a bitmask. In order to deduce per host per port information, each record of PHL is split into source and destination identifiers (hash(srcID, srcPort), hash(dstID, dstPort)), for illustration see Figure 5.3.

The number of stored application types and the size of the application-matched bitmasks are the results of a trade-off due to the memory addressing of the GPU. The GPU can handle 2 or 4-byte long data most efficiently [8], thus 16 or 32 possible application categories can be distinguished in these cases. I.e., if P2P is chosen as one category, its 'variants' like BitTorrent and Gnutella cannot be differentiated later. But our experiences support that 16/32 application types are practically enough to cover the main types such as P2P file sharing, web browsing, streaming, VoIP, etc. If needed, this part of the algorithm can be extended to support more application types by switching to longer non-hardware supported data length representation. However, this solution results in performance degradation. An alternative solution is to switch between signature lists. Once the packets filled in the global memory are evaluated based on the first list, then these packets can be evaluated based on a second, third, etc. lists.

The theoretical speed of the proposed DPI engine with current hardware is approx. 2 million packets/sec (see 5.1.7 for the performance test) that is 20k flow presuming 100 packets/flow in average [36]. This means that a 2-bytes-long hash representation will not fully occupy its array on the GPU which is stored only for one DPI iteration per batch. Since the duration of the flow is approx. 60 sec [36] there are 60 times more flows to be stored in the DPT (hosted in the main memory). Therefore

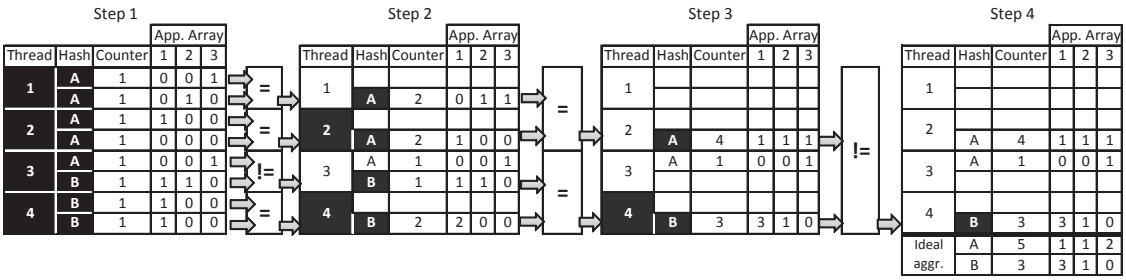


Figure 5.4: The aggregation of packet information with parallel reduction

the hash representation on the CPU side can be increased to 3 bytes which results in a 256 times larger hash array occupying in the order of 100 MBytes memory.

In order to avoid the overhead of frequent GPU initializations, we propose to store the PHLs in a Packet Hit Buffer (PHB) in the operative memory until the aggregation phase.

5.2.2 Aggregating the per-packet application hits

Before updating the DPT, we propose to aggregate the PHB based on parallel reduction. The idea of parallel reduction is to slice a big problem to smaller tasks, distribute it to processing units and execute the computation of the partial tasks parallel. The algorithm proceeds to recursively reduce the problem size.

The reduction of the PHB is more efficient if its input is sorted. The sorting ensures that the same [srcID; srcPort] and [dstID; dstPort] pairs come after each other. This way the shared memory blocks of the GPU will contain application information about packets that likely belong to the same [host; port] pair. Therefore the chance of effective aggregation per block is high. We have applied parallel radix sort [85] to create a Sorted PHB (SPHB) based on the hash value of the packets.

The SPHB is aggregated with parallel reduction into the Aggregated Packet Hit Buffer (APHB) similarly to the parallel prefix sum [85]. Extending [85], we propose to use conditional aggregation while sweeping through the data structure. The idea is to handle the SPHB as a balanced binary tree and sweep it from the leaves to compute the aggregated sums.

The steps of the aggregation can be followed in Figure 5.4 and Algorithm 7. The input of the algorithm is the SPHB extended with a counter in each row. This counter is initialized to 1 and it represents the number of successfully aggregated packets (i.e., rows). In the first step in Figure 5.4, each thread compares the hash keys of the two rows they are responsible for. If the keys are equal then the aggregation occurs by summing the aggregation counters and the proper application hit array elements. The

results are stored at every second row. In the following step, every second thread does actual work, comparing and adding data in the recently updated rows, and storing them in every fourth array position. This is repeated until there is no more row left untouched. The advantage of parallel reduction with such a data structure is that there are no concurrent memory reads or writes. The threads have to be synchronized after each step to ensure memory content consistency.

Operations are performed in place in the shared memory, thus no additional memory allocations are needed. If the binary tree has n leaves and $\log(n)$ levels, then only $O(n)$ operations are needed by the algorithm (it performs $(n-1)$ adds).

Algorithm 7: Packet Aggregation

```
   Input: SPHB
   Output: APHB
1  for d = 0, d < log_2 n − 1, d++ (for the size of n see Section 5.2.4) do
2      for k = 0, n − 1 > k, k+ = 2^(d+1) parallel do
3          if x[k + 2^d − 1].hash == x[k + 2^(d+1) − 1].hash then
4              x[k + 2^(d+1) − 1].counter+ = x[k + 2^d − 1].counter;
5              for i = 0, i < size(bitmask), i++ do
6                  x[k + 2^(d+1) − 1].app[i]+ = x[k + 2^d − 1].app[i];
```

5.2.3 Sequential update of the DHT

The APHB is moved back to the operative memory and the DPT is updated sequentially by adding the corresponding applications hit values to the rows of the DPT. The update is started from the last row of each block of the APHB since these rows have the highest aggregation level. The counter of these rows shows how many rows should be jumped upwards to get to the next useful row to be included in the DPT. The speed of updating the DPT is also increased by the fact that the SPHB remains sorted after the parallel reduction. This results in good spatial locality of the hash values in the DPT, which means high hit rate in the L2-cache of the CPU during update [104]. Also note that when querying the DPT during traffic classification, the best matching application for a [host; port] pair can be obtained in one step, if the application hit list in the DPT is a heap, where the number of hits for different application types is in decreasing order.

A flow is **classified** by querying the DPT for the most probable application type corresponding to the [srcID; srcPort] pair and the [dstID; dstPort] pair, and comparing their results. In case of equivalence the application type is unambiguous. In case of difference, we decide for the more probable and more specific category.

5.2.4 Validation – Evaluation of GPU based aggregation

We can measure the compression ratio of the aggregation as the size of the APHB divided by the size of the SPHB. When evaluating the compression ratio, the following issues should be considered:

- *Ideal compression*: All possible items are aggregated, i.e., every element occurs only once.
- *Parallel reduction*: Due to the binary tree based execution model, this solution cannot aggregate nodes from different branches of different roots. So this solution provides lower compression ratio than the ideal case by definition. For illustration see Figure 5.4.
- *Practical HW constraints*: The number of SPHB rows that can be aggregated in a block of the GPU is limited by the maximum number of threads per block (recommended maximum is 512) and the size of the shared memory (16 Kbytes) [8]. In our example, a row is stored on 19 bytes (2 bytes for the hash key, one for the aggregation counter and 16 for the application hit counter). Thus at most 862 rows could fit in the shared memory. But the number of leaves of the binary tree is the power of 2 and each thread examines 2 rows at the same time, so only 512 rows with 256 threads can be handled per block. I.e., if X adjacent rows could be ideally aggregated to 1 row, the GPU can aggregate it to $\lceil X/512 \rceil$ rows.

In an example from a live traffic trace, the SPHB contained 256,000 rows. By ideal aggregation, APHB could be aggregated into 4,560 rows meaning that the compression ratio is 1.78%. By using the GPU, the APHB could be aggregated to 41,095 rows meaning a compression ratio of 16%. However, the aggregation ratio can be improved by repeating the aggregation several times. The compression ratio as a function of the number of iterations is showed in Figure 5.5. It can be seen that the initial compression ratio can be approx. halved in the next few iterations resulting a total compression ratio of 8%.

The next question is the performance of the aggregation, i.e., how much time the aggregation needs. Our experiments show that the aggregation tasks of the GPU are more than 4 magnitudes faster than the DPT update done by the CPU (hash insert and update). That is, the aggregation causes no practical overhead in the overall traffic classification process.

5.2.5 Total system throughput

In this section we discuss the overall performance of the capture and classification system in case of one CPU core and one GPU. The task of the CPU is to obtain the

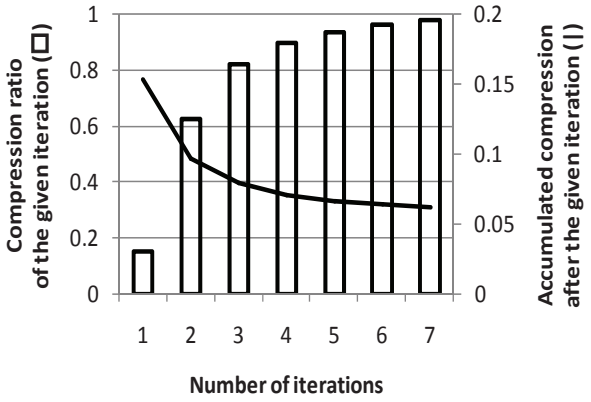


Figure 5.5: The compression ratio of the GPU based aggregation in the function of execution iteration number

data from the Network Interface Card (NIC), compress the data, feed the GPU with the data and create a classified flow log. The GPU is responsible for the classification.

According to our test (see Figure 5.6), the proposed system can classify real-time traffic at approx. 1.7 Gbps per CPU core. The components of the CPU load are as follows.

- 20% CPU load to capture the traffic: we use our own developed kernel module (similar to [24]) to avoid the built-in packet handling mechanism of the kernel. This results in a significant reduction of CPU load during the NIC read.

- 70% CPU load to create flow logs: the output of the kernel module is passed to a user level process which collects the packets into flows.

- 10% CPU load is connected to the feeding of the GPU and process the classified data.

Note that the GPU is idle in most of the time, thus it is able to serve several traffic classification threads parallel. The current bottleneck is the CPU part of the process. With a multicore environment, the achievable speed scales up to the number of processors linearly achieving an approx. 6.7 Gbps total system throughput in a 4 core environment.

5.2.6 Application

Beside the proposed use case, the method can be a building block of a wire-speed multiprocessor flow aggregation module further offloading the CPUs.

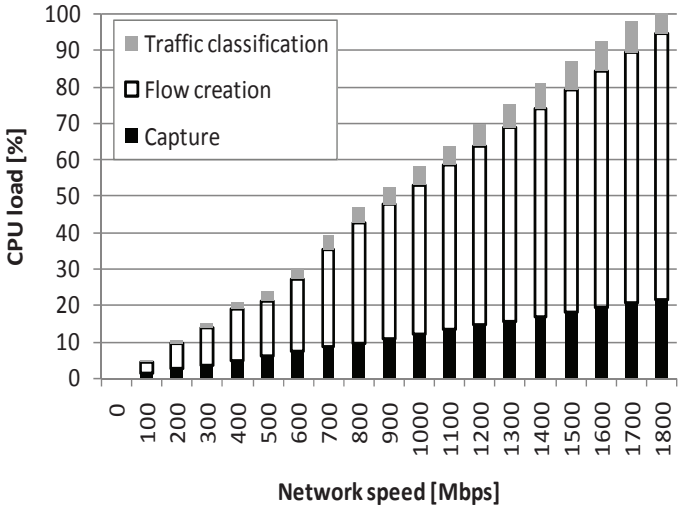


Figure 5.6: Total CPU load as a function of network speed

5.3 Measurement setup

The measurements were done on a PC with two Dual-Core Intel Xeon Processor 5130/2.00 GHz, Intel Pro/1000 NICs, Asus ENGTX260 video card. The operation system was a 32-bit Ubuntu Linux 8.04 with 2.6.26-1-686 kernel and NAPI supported driver. The applications were compiled with `gcc-4.1`. The perl is an 5.10.0 multi-threaded version. The CUDA API used the CUDA SDK 2.0 with an NVIDIA Driver for Linux with CUDA Support (177.73) and the CUDA Toolkit for Ubuntu 7.10. The NICs were bridged by `brctl`, thus they appeared as one virtual adapter towards the system.

5.4 Summary

This chapter discusses a GPU assisted solution to reach multi-Gbps speeds on commodity hardware. A modification of Zobrist hashing for signature matching problem was introduced, which uses compact signatures allowing fitting the entire dictionary into fast memory of the GPU. The achievable speed was measured over 6 Gbps for a typical traffic mix. The connection pattern analysis method utilizes a parallel-prefix-sum algorithm adapted to GPU, which is four magnitudes faster than the CPU based aggregation. It was also shown, that when the above algorithms are placed in a commodity PC, the system performance including traffic capture, flow aggregation, GPU algorithms, and GPU communication allows an overall typical speed exceeding 6 Gbps.

Chapter 6

Summary of thesis results

Accurate identification and categorization of network traffic according to application type is an important element of numerous network management activities such as flow prioritization, traffic shaping/policing, and diagnostic monitoring. Traffic classification aims at identifying the traffic mixture in the network. Several different classification approaches can be found in the literature. However, the validation of these methods is weak and ad hoc, because neither a reliable and widely accepted validation technique nor reference packet traces with well-defined content are available. In Section 2.1, a *novel validation method* is proposed for characterizing the accuracy and completeness of traffic classification algorithms. The main advantages of the new method are that it is based on realistic traffic mixtures, and it enables a highly automated and reliable validation of traffic classification. As a proof-of-concept, it is examined how state-of-the-art traffic classification methods perform for the most common application types.

A couple of different traffic classification approaches have been proposed in the literature, but none of them performs well for all different application traffic types present in the Internet. Thus, a *combined traffic classification method* that includes the advantages of different approaches is needed, in order to provide a high level of classification completeness and accuracy. The pros and cons of the classification methods are analyzed based on the experienced accuracy for different types of applications. Using the gained knowledge about the strengths and weaknesses of the existing approaches, a novel traffic classification method is proposed in Section 2.2. The novel method is based on a complex decision mechanism, in order to provide an appropriate identification mode for each different application type. As a consequence, the ratio of the unclassified traffic becomes significantly lower. Further, the reliability of the classification improves, as the various methods validate the results of each other. The novel method is tested on several network traces, and it is shown

that the proposed solution improves both the completeness and the accuracy of the traffic classification, when compared to existing methods.

Game traffic depends on two main factors, the game protocol and the gamers' behavior. Based on a few popular real-time multiplayer games this work investigates the latter factor showing how a set of typical game phases – e.g., player movement, changes in the environment – impacts traffic on different observation levels. The nature of human behavior has such a high impact on traffic characteristics that it influences the traffic both at a macroscopic – e.g., traffic rate – and at a microscopic – payload content – level.

By understanding the nature of this impact a *user behavior detection algorithm* is introduced in Section 3.1 to grab specific events and states from passive traffic measurements. The algorithms focus on the characteristics of the traffic rate, showing what information can be gathered by observing only packet header information. As an application of our method *some results*, including a detailed analysis of measurements taken from an operational broadband network, are presented.

Emerging Massively Multiplayer Online Real Time Strategy games require complex game server architecture to make the transmission of the state information of a huge number of units generated by a lot of players feasible. This architecture design is supported by network traffic simulations based on the accurate characterization of player behavior. However, these characteristics of player behavior in RTS games have not been investigated yet, thus, this is the main motivation of this work. In particular in Section 3.2, we introduce a method that can *identify the war periods* in real-time strategy game sessions based on non-intrusive measurements thus, it is possible to analyze a vast number of game plays. Some early results comparing the real and gaming worlds are also presented.

A novel model and an algorithm are introduced in Section 4 to extend the Deep Packet Inspection traffic classification method with the analysis of non-fix byte signatures, which are not considered in current methods. The model captures the variation of the dynamic byte segments and provides parameters for the algorithm. The introduced algorithm exploits the spatial and temporal correlation by examining and extracting the correlation structure of the traffic and constructing signatures based on the observed correlation. The algorithm is evaluated by examining proprietary gaming traffic and also other known non-gaming protocols.

Section 5 discusses necessary components of a *GPU-assisted traffic classification* method, which is capable of multi-Gbps speeds on commodity hardware. The majority of the traffic classification is pushed to the GPU to offload the CPU, which then may serve other processing intensive tasks, e.g., traffic capture. Two massively parallelizable algorithms are presented suitable for GPUs. The first one performs signature search using a modification of Zobrist hashing. The second algorithm supports connection pattern-based analysis and aggregation of matches using a parallel-prefix-sum

algorithm adapted to GPU. The performance tests of the proposed methods showed that traffic classification is possible up to approximately 6 Gbps with a commodity PC.

Bibliography

[1] *Age of Mythology*, http://www.microsoft.com/games/ageofmythology/.

[2] *Application specific bit strings,* *http://www.cs.ucr.edu/~tkarag/papers/strings.txt*.

[3] *Command and Conquer 3*, http://www.commandandconquer.com/default.aspx.

[4] *Command and Conquer Generals*, http://www.ea.com/official/cc/firstdecade/us/generals.jsp.

[5] *Correlates of War Project, 2008*, http://www.correlatesofwar.org/.

[6] *Cossacks*, http://www.cossacks.com/.

[7] *Cristian Estan: Fast and Compact Signature Matching*, http://pages.cs.wisc.edu/~estan/publications/xfa-tr.html.

[8] *CUDA Programming Guide 2.0*, http://developer.download.nvidia.com/compute/cuda/2.0-Beta2/docs/Programming_Guide_2.0beta2.pdf.

[9] *Ericsson Service Aware Support Node (SASN)*, http://www.ericsson.com/solutions/products/hp/SASN_pa

[10] *Erik Hjelmvik: The SPID Algorithm -Statistical Protocol IDentification*, http://spid.wiki.sourceforge.net/.

[11] *Gnutella*, http://www.gnutella.com.

[12] *IANA port numbers,* *http://www.iana.org/assignments/port-numbers*.

[13] *Ipoque: Internet Study 2008/2009*, http://www.ipoque.com/resources/internet-studies/internet-study-2008_2009.

[14] *Lightreading Insider, Targeted Analysis of the Telecom Industry: Deep Packet Inspection: 2009 Market Forecast.*

[15] *List of gaming traces*, http://perform.wpi.edu/downloads/.

[16] *M. Iliofotou, P. Pappu, M. Faloutsos, M. Mitzenmacher, S. Singh, G. Varghese: Network Traffic Analysis using Traffic Dispersion Graphs (TDGs): Techniques and Hardware Implementation, Technical Report, 2007*, http://www.cs.ucr.edu/~marios/Papers/UCR-CS-2007-05001.pdf.

[17] *MSN Messenger*, http://join.msn.com/messenger/overview2000.

[18] *N. Cascarano, P. Rolando, F. Risso, R. Sisto: iNFAnt: NFA Pattern Matching on GPGPU Devices*, netgroup.polito.it/Members/niccolo_cascarano/pub/cuda-02-2010.pdf.

[19] *N. Goyal, J. Ormont, R. Smith, K. Sankaralingam, C. Estan: Signature Matching in Network Processing using SIMD/GPU architectures*, http://www.cs.wisc.edu/techreports/2008/TR1628.pdf.

[20] *NetEqualizer Bandwidth Shaper*, http://www.netequalizer.com/.

[21] *PacketLogic*, http://www.proceranetworks.com/products/traffic-shaping.html.

[22] *PacketShaper*, http://www.packeteer.com/products/packetshaper/.

[23] *Penguin Computing Delivers University Of Delaware's Fastest Supercomputer to Global Computing Laboratory , 2009*, http://www.penguincomputing.com/press/press_releases/Delaware.

[24] *PF_RING, snapshot date: 2008.07*, http://www.ntop.org/PF_RING.html.

[25] *Ports for Internet Services,*
http://www.chebucto.ns.ca/~rakerman/port-table.html.

[26] *R. Kohavi, R. Quinlan: Decision Tree Discovery*, http://robotics.stanford.edu/users/ronnyk/treesHB.ps/.

[27] *Rapidminer*, http://sourceforge.net/projects/yale.

[28] *RFC 2113*, http://www.networksorcery.com/enp/rfc/rfc2113.txt.

[29] *RFC 2246*, http://www.ietf.org/rfc/rfc2246.txt.

[30] *RFC 4251*, http://www.ietf.org/rfc/rfc4251.txt.

[31] *Skype*, http://www.skype.com.

[32] *Spring Replays*, http://replays.unknown-files.net/.

[33] *Stephen Shankland: Google uncloaks once-secret server, 2009*, http://news.cnet.com/8301-1001_3-10209580-92.html.

[34] *TA Spring*, http://spring.clan-sy.com/.

[35] *The measurement created for this paper*, http://www.crysys.hu/~szabog/measurement.tar.

[36] *Traffic Measurements and Models in Multi-Service Networks (TRAMMS): Project Achievements, 2009*, http://projects.celtic-initiative.org/tramms/files/TRAMMS_leaflet_final_lq.pdf.

[37] *uTorrent*, http://www.utorrent.com.

[38] *World of Warcraft*, http://www.worldofwarcraft.com/index.xml.

[39] *Matlab*, 1983-2009, http://www.mathworks.com/.

[40] *Lua Programming Language*, 1993-2008, http://www.lua.org/.

[41] *MMORPG analysis code*, 1993-2008, http://www.crysys.hu/~szabog/MMORPG/.

[42] *Mark Claypool's Homepage*, 1995-2009, http://web.cs.wpi.edu/~claypool/.

[43] *Bro Intrusion Detection System*, 1998-2009, http://bro-ids.org/publications.html.

[44] *Snort Intrusion Detection System*, 1998-2009, http://www.snort.org/snort.

[45] *CoralReef*, 1999-2009, http://www.caida.org/tools/measurement/coralreef/description.xml.

[46] *Unreal Tournament 2003*, 2002, http://www.unrealtournament2003.com/.

[47] *Eve Online*, 2003, http://www.eve-online.com/.

[48] *Second Life*, 2003, http://secondlife.com/.

[49] *Star Wars Galaxies*, 2003, http://starwarsgalaxies.station.sony.com/.

[50] *The Daedalus Project*, 2003-2009, http://www.nickyee.com/daedalus/archives/001586.php.

[51] *Guild Wars*, 2005, http://www.guildwars.com/.

[52] *Measuring Online Game Application in GPRS and UMTS*, 2005, http://www.netlab.hut.fi/opetus/s38310/04-05/Kalvot_04-05/H%E4m%E4l%E4inen_070605.ppt.

[53] *Quake4*, 2005, http://www.idsoftware.com/games/quake/quake4/.

[54] *Silkroad Online*, 2005, http://www.silkroadonline.net/.

[55] *Allot Communications whitepaper: Digging Deeper Into Deep Packet Inspection (DPI)*, 2007, https://www.dpacket.org/articles/digging-deeper-deep-packet-inspection-dpi.

[56] *Top 5 most costly viruses of all time*, 2009, `http://www.anti-virus-software-review.toptenreviews.com/top-5-most-costly-viruses-of-all-time-pg5.html`.

[57] A. W. Moore and K. Papagiannaki, *Toward the Accurate Identification of Network Applications*, Proc. PAM (Boston, MA, USA), March 2005.

[58] P. Abry and D. Veitch, *Wavelet Analysis of Long-Range-Dependent Traffic*, IEEE Transactions on Information Theory **44** (1998), no. 1, 2–15.

[59] Andres Arjona, Cedric Westphal, Antti Ylä-Jääski, and Martin Kristensson, *Towards High Quality VoIP in 3G Networks - An Empirical Study*, AICT '08: Proceedings of the 2008 Fourth Advanced International Conference on Telecommunications (Washington, DC, USA), IEEE Computer Society, 2008, pp. 143–150.

[60] A.-L. Barabási, *The Origin of Bursts and Heavy Tails in Human Dynamics*, Nature **435** (2005), 207.

[61] M. Becchi and P. Crowley, *Efficient regular expression evaluation: theory to practice*, ANCS '08: Proceedings of the 4th ACM/IEEE Symposium on Architectures for Networking and Communications Systems (New York, NY, USA), ACM, 2008, pp. 50–59.

[62] T. Beigbeder, R. Coughlan, C. Lusher, J. Plunkett, E. Agu, and M. Claypool, *The Effects of Loss and Latency on User Performance in Unreal Tournament 2003*, NetGames '04: Proceedings of 3rd ACM SIGCOMM workshop on Network and system support for Games (New York, NY, USA), ACM Press, 2004, pp. 144–151.

[63] J. Beran, *Statistics for Long-Memory Processes*, Chapman & Hall (1994), 71–86.

[64] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, *Traffic Classification On The Fly*, SIGCOMM Comput. Commun. Rev. **36** (2006), no. 2, 23–26.

[65] J.B.D. Cabrera, J. Gosar, W. Lee, and R.K. Mehra, *On the statistical distribution of processing times in network intrusion detection*, In 43rd IEEE Conference on Decision and Control, Dec 2004, pp. 75–80.

[66] F. R. Cecin, J. V. Barbosa, and C. F. R. Geyer, *A communication optimization for conservative interactive simulators*, IEEE Communications Letters Vol. 10, No. 9, 2006, pp. 686–688.

[67] N. V. Chawla, L. O. Hall, and A. Joshi, *Wrapper-based computation and evaluation of sampling methods for imbalanced datasets*, UBDM '05: Proceedings of the 1st international workshop on Utility-based data mining (New York, NY, USA), ACM, 2005, pp. 24–33.

[68] K. Chen, P. Huang, C. Huang, and C. Lei, *Game Traffic Analysis: An MMORPG Perspective*, NOSSDAV '05 (New York, USA), 2005.

[69] K. Chen, J. Jiang, P. Huang, H. Chu, C. Lei, and W. Chen, *Identifying MMORPG Bots: A Traffic Analysis Approach*, ACM SIGCHI ACE'06 (Los Angeles, USA), Jun 2006.

[70] M. Claypool, *The Effect of Latency on User Performance in Real-time Strategy Games*, Comput. Netw. **49** (2005), no. 1, 52–70.

[71] M. Claypool, David LaPoint, and Josh Winslow, *Network Analysis of Counter-strike and Starcraft*, IEEE International Performance, Computing, and Communications Conference (IPCCC) (2003).

[72] A. Cricenti and P. Branch, *ARMA(1,1) Modeling of Quake4 Server to Client Game Traffic*, NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for Games (New York, NY, USA), ACM, 2007, pp. 70–74.

[73] M. Crotti, M. Dusi, F. Gringoli, and L. Salgarelli, *Traffic Classification through Simple Statistical Fingerprinting*, SIGCOMM Comput. Commun. Rev. **37** (2007), no. 1, 5–16.

[74] M. E. Crovella and A. Bestavros, *Self-similarity in World Wide Web Traffic: Evidence and Possible Causes*, IEEE/ACM Transactions on Networking **5** (1997), 835–846.

[75] Dorothy E. Denning, *An intrusion-detection model*, IEEE Symposium on Security and Privacy, 1986, pp. 118–133.

[76] L. Deri, *High-speed dynamic packet filtering*, vol. 15, Plenum Press, 2007, pp. 401–415.

[77] S. Dharmapurikar, P. Krishnamurthy, T. Sproull, and J. Lockwood, *Deep packet inspection using parallel bloom filters*, Hot Interconnects (2003), 44–51.

[78] J. Erman, M. Arlitt, and A. Mahanti, *Traffic Classification Using Clustering Algorithms*, Proc. MineNet '06 (New York, NY, USA), 2006.

[79] J. Erman, A. Mahanti, M. Arlitt, I. Cohen, and C. Williamson, *Offline/realtime traffic classification using semi-supervised learning*, Perform. Eval. **64** (2007), no. 9-12, 1194–1213.

[80] J. Erman, A. Mahanti, and M. F. Arlitt, *Internet traffic identification using machine learning*, GLOBECOM, 2006.

[81] W. Feng, D. Brandt, and D. Saha, *A long-term study of a popular mmorpg*, NetGames '07 (New York, NY, USA), ACM, 2007, pp. 19–24.

[82] S. Fernandes, R. Antonello, J. Moreira, and C. Kamienski, *Traffic Analysis Beyond This World: the Case of Second Life*, NOSSDAV '07 (Urbana, Illinois, USA), June 2007.

[83] T. Fritsch, H. Ritter, and J. Schiller, *The Effect of Latency and Network Limitations on MMORPGs: A Field Study of Everquest2*, NetGames '05 (New York, USA), 2005.

[84] P. Haffner, S. Sen, O. Spatscheck, and D. Wang, *Acas: automated construction of application signatures*, MineNet '05 (New York, NY, USA), 2005.

[85] M. Harris, S. Sengupta, and John D. Owens, *Parallel prefix sum (scan) with cuda*, Hubert Nguyen, editor, GPU Gems 3 (2007).

[86] T. K. Ho, *The random subspace method for constructing decision forests*, IEEE Trans. Pattern Anal. Mach. Intell. **20** (1998), no. 8, 832–844.

[87] N.-F. Huang, H.-W. Hung, S.-H. Lai, Y.-M. Chu, and W.-Y. Tsai, *A gpu-based multiple-pattern matching algorithm for network intrusion detection systems*, Mar 2008.

[88] N .Ierace, C. Urrutia, and R. Bassett, *Intrusion prevention systems*, Ubiquity **2005**, no. June, 2–2.

[89] N. Jacob and C Brodley, *Offloading ids computation to the gpu*, Dec 2006.

[90] T. Karagiannis, A. Broido, M. Faloutsos, and K. Claffy, *Transport Layer Identification of P2P Traffic*, Proc. IMC (Taormina, Sicily, Italy), October 2004.

[91] T. Karagiannis, K. Papagiannaki, and M. Faloutsos, *BLINC: Multilevel Traffic Classification in the Dark*, Proc. ACM SIGCOMM (Philadelphia, Pennsylvania, USA), August 2005.

[92] H. Kim, Kc Claffy, M. Fomenkov, D. Barman, M. Faloutsos, and K. Lee, *Internet traffic classification demystified: Myths, caveats, and the best practices*, ACM CoNEXT 2008, Dec 2008.

[93] H. Kim and B. Karp, *Autograph: Toward Automated, Distributed Worm Signature Detection*, SSYM'04 (Berkeley, CA, USA), 2004.

[94] J. Kim, J. Choi, D. Chang, T. Kwon, Y. Choi, and E. Yuk, *Traffic Characteristics of A Massively Multi-Player Online Role Playing Game*, NetGames '05 (New York, USA), 2005.

[95] A. Lakhina, M. Crovella, and C. Diot, *Mining Anomalies Using Traffic Feature Distributions*, Proc. ACM SIGCOMM (Philadelphia, Pennsylvania, USA), August 2005.

[96] K. Lee, S. Hong, S. J. Kim, I. Rhee, and S. Chong, *SLAW: A Mobility Model for Human Walks*, Infocom'09, March 2009.

[97] Y. Lee, S. Agarwal, C. Butcher, and J. Padhye, *Measurement and Estimation of Network QoS Among Peer Xbox 360 Game Players*, PAM (Mark Claypool and Steve Uhlig, eds.), Lecture Notes in Computer Science, vol. 4979, Springer, 2008, pp. 41–50.

[98] Z. Li, M. Sanghi, Y. Chen, M. Kao, and B. Chavez, *Hamsa: Fast Signature Generation for Zero-day PolymorphicWorms with Provable Attack Resilience*, SP 2006 (Washington, DC, USA), 2006.

[99] H. Liang, I. Tay, M. F. Neo, W. T. Ooi, and M. Motani, *Avatar Mobility in Networked Virtual Environments: Measurements, Analysis, and Implications*, CoRR **abs/0807.2328** (2008), informal publication.

[100] J. Ma, K. Levchenko, C. Kreibich, S. Savage, and G. M. Voelker, *Unexpected Means of Protocol Inference*, IMC '06: Proceedings of the 6th ACM SIGCOMM conference on Internet measurement (New York, NY, USA), ACM, 2006, pp. 313–326.

[101] S. Mallat, *A Theory for Multiresolution Signal Decomposition: The Wavelet Representation*, IEEE Pattern Anal. and Machine Intell. **11, no. 7** (1989), 674–693.

[102] A. McGregor, M. Hall, P. Lorier, and A. Brunskill, *Flow Clustering Using Machine Learning Techniques*, Proc. PAM (Antibes Juan-les-Pins, France), April 2004.

[103] A. W. Moore and D. Zuev, *Internet Traffic Classification Using Bayesian Analysis Techniques*, Proc. SIGMETRICS (Banff, Alberta, Canada), June 2005.

[104] A. Papadogiannakis, D. Antoniades, M. Polychronakis, and E. P. Markatos, *Improving the performance of passive network monitoring applications using locality buffering*, vol. 36, 2007.

[105] B. Park, Y. J. Won, M. Kim, and J. W. Hong, *Towards automated application signature generation for traffic identification*, NOMS, 2008, pp. 160–167.

[106] M. Perényi and S. Molnár, *Enhanced Skype Traffic Identification*, Proc. Valuetools 2007 (Nantes, France), 2007.

[107] D. Pittman and C. GauthierDickey, *A Measurement Study of Virtual Populations in Massively Multiplayer Online Games*, NetGames '07: Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for Games (New York, NY, USA), ACM, 2007, pp. 25–30.

[108] S. Reid and M. Reid, *The correlates of war data on war: An update to 1997*, Conflict Management and Peace Science, 18/1, 2000, pp. 123–144.

[109] E. Seamans and T. Alexander, *Fast virus signature matching*, GPU Gems **3** (2007), 771–783.

[110] S. Sen and J. Wang, *Analyzing peer-to-peer traffic across large networks*, Proc. Second Annual ACM Internet Measurement Workshop, November 2002.

[111] M. Sipser, *Introduction to the theory of computation*, **1997**, 31 – 47.

[112] P. Svoboda and M. Rupp, *Online gaming models for wireless networks*, Internet And Multimedia Systems And Applications (IASTED) (2005).

[113] G. Szabó, I. Szabó, and D. Orincsay, *Accurate traffic classification*, Proc. IEEE WOWMoM (Helsinki, Finnland), June 2007.

[114] F. Tan, *Improving feature selection techniques for machine learning*, Ph.D. thesis, Atlanta, GA, USA, 2007, Adviser-Bourgeois, Anu G.

[115] S. A. Tan, W. Lau, and A. Loh, *Networked Game Mobility Model for First-Person-Shooter Games*, NetGames '05: Proceedings of 4th ACM SIGCOMM workshop on Network and system support for Games (New York, NY, USA), ACM, 2005, pp. 1–9.

[116] G. Vasiliadis, S. Antonatos, M. Polychronakis, E. P. Markatos, and S. Ioannidis, *Gnort: High performance network intrusion detection using graphics processors*, RAID '08: Proceedings of the 11th international symposium on Recent Advances in Intrusion Detection (Berlin, Heidelberg), Springer-Verlag, 2008, pp. 116–134.

[117] G. Vasiliadis, M. Polychronakis, S. Antonatos, E. P. Markatos, and S. Ioannidis, *Regular expression matching on graphics hardware for intrusion detection*, RAID, 2009, pp. 265–283.

[118] A. Veres, Z. Kenesi, S. Molnár, and G. Vattay, *On The Propagation of Long-Range Dependence in The Internet*, ACM SIGCOMM (Stockholm, Sweden, USA), Aug 2000.

[119] N. Weaver, V. Paxson, and J. M. Gonzalez, *The shunt: an fpga-based accelerator for network intrusion prevention*, FPGA '07: Proceedings of the 2007 ACM/SIGDA 15th international symposium on Field programmable gate arrays (New York, NY, USA), ACM, 2007, pp. 199–206.

[120] N. Williams, S. Zander, and G. Armitage, *A preliminary performance comparison of five machine learning algorithms for practical ip traffic flow classification*, SIGCOMM Comput. Commun. Rev. **36** (2006), no. 5, 5–16.

[121] W. Willinger, M. Taqqu, and A. Erramilli, *A Bibliographical Guide to Self-similar Traffic and Performance Modeling for Modern High-speed Network*, 1996.

[122] W. Willinger, M. S. Taqqu, R. Sherman, and D. V. Wilson, *Self-similarity Through High-variability: Statistical Analysis of Ethernet LAN Traffic at The Source Level*, IEEE/ACM Transactions on Networking 5 **5** (1997), 71–86.

[123] C. Wu, J. Yin, Z. Cai, E. Zhu, and J. Chen, *A hybrid parallel signature matching model for network security applications using simd gpu*, APPT '09: Proceedings of the 8th International Symposium on Advanced Parallel Processing Technologies (Berlin, Heidelberg), Springer-Verlag, 2009, pp. 191–204.

[124] K. Xu, Z. Zhang, and S. Bhattacharyya, *Profiling Internet Backbone Traffic: Behavior Models and Applications*, Proc. ACM SIGCOMM (Philadelphia, Pennsylvania, USA), August 2005.

[125] M. Ye and L. Cheng, *System-performance modeling for massively multiplayer online role-playing games*, IBM Syst. J. **45** (2006), no. 1, 45–58.

[126] F. Yu, Z. Chen, Y. Diao, T. V. Lakshman, and R. H. Katz, *Fast and Memory-efficient Regular Expression Matching for Deep Packet Inspection*, ANCS '06: Proceedings of the 2006 ACM/IEEE symposium on Architecture for networking and communications systems (New York, NY, USA), ACM, 2006, pp. 93–102.

[127] S. Zander, T. Nguyen, and G. Armitage, *Automated Traffic Classification and Application Identification Using Machine Learning*, Proc. IEEE LCN (Sydney, Australia), November 2005.

[128] G. Zhang, G. Xie, J. Yang, Y. Min, Z. Zhou, and X. Duan, *Accurate online traffic classification with multi-phases identification methodology*, CCNC 2008, Dec 2008.

[129] A. L. Zobrist, *A hashing method with applications for game playing*, Tech. Rep. 88, Computer Sciences Department, University of Wisconsin (1969).

Publications

Journal papers

[J1] A. Callado, **G. Szabó**, B. P. Gerö, J. Kelner, S. Fernandes, D. Sadok, : Survey on Internet Traffic Identification and Classification, *IEEE Communications Surveys and Tutorials*, 2009, Vol. 11, Num. 3, pp. 37–52.

[J2] **G. Szabó**, A. Veres, S. Molnár: On The Impacts of Human Interactions in MMORPG Traffic, *Journal of Multimedia Tools and Applications*, 2009, Vol. 45, Num. 1-3, pp. 133–161.

[J3] **G. Szabó**, I. Gódor, A. Veres, Sz. Malomsoky, S. Molnár: Traffic Classification over Gbit Speed with Commodity Hardware, *IEEE Journal of Communications Software and Systems*, 2010, Vol. 5, Num. 3.

[J4] **G. Szabó**, S. Molnár: Nagyon sok szereplős online szerepjátékok skálázási tulajdonságainak vizsgálata, *Híradástechnika*, 2007, Vol. LXII., Num. 11, pp. 20–26.

[J5] **G. Szabó**, S. Molnár: On The Scaling Characteristics of MMORPG Traffic, *Híradástechnika*, 2008, Vol. LXIII., Num. 1, pp. 40–47. (Selected Papers)

Conference papers

[C1] **G. Szabó**, I. Szabó, D. Orincsay: Accurate Traffic Classification. In *Proc., IEEE Wowmom 2007*, Jun 17-21, 2007, Helsinki, Finland.

[C2] **G. Szabó**, D. Orincsay, B. P. Gerö, Sándor Györi, Tamás Borsos: Traffic Analysis of Mobile Broadband Networks In *Proc., ICST Wicon 2007*, Oct 22-23, 2007, Austin, Texas, USA.

[C3] **G. Szabó**, D. Orincsay, Sz. Malomsoky, I. Szabó: On the Validation of Traffic Classification Algorithms In *Proc., PAM 2008*, April 29-30, 2008, Cleveland, Ohio, USA.

[C4] **G. Szabó**, A. Veres, S. Molnár: Effects of User Behavior on MMORPG Traffic In *Proc., ICC 2009*, June 14-18, 2009, Dresden, Germany.

[C5] **G. Szabó**, A. Veres, S. Molnár: Towards Understanding the Evolution of Wars in Virtual and Real Worlds In *Proc., ICSNC 2009*, September 20-25, 2009, Porto, Portugal.

Other non-related publications

[J6] **G. Szabó**, B Bencsáth: DHA támadás elleni védekezés központosított szűréssel, *Híradástechnika*, 2006, Vol. LIX. Num. 2, pp. 42–49.

[J7] A. Szentgyörgyi, **G. Szabó**, B. Bencsáth: Bevezetés a botnetek világába, *Híradástechnika*, 2008, Vol. LXIII. Num. 11, pp. 10–15. (Pollák-Virág díjas).