Parameter Estimation for Heuristic-based Internet Traffic Classification

Michael Finsterbusch, Jean-Alexander Müller, Chris Richter
Dept. of Communication and Computer Science
Hochschule für Telekommunikation Leipzig (University of Applied Sciences)
Leipzig, Germany
{finsterbusch|jeanm|richter}@hft-leipzig.de

***Abstract*—Accurate traffic classification is necessary for many administrative networking tasks like security monitoring, providing Quality of Service and network design or planning. We apply 18 machine learning algorithms to classify network traffic based on six classes of statistical parameters. In contrast to other studies, we use a per-packet approach instead of a per-flow approach to make it possible to use the classification results for real-time network interception. In this paper we illustrate the accuracy of the algorithms with different parameter combinations with the goal to reduce the amount of necessary parameters needed for high accuracy traffic classification. Our results indicate that some parameter combinations can be used to classify a large number of protocols. We identified algorithms with good and worse classification accuracy and algorithms which need much time for classification, so that they cannot be used for real-time classification.**

***Keywords*-flow classification, Internet traffic, traffic identification.**

I. Introduction

Network traffic classification or particularly application classification and identification is the process of identifying the type of application (or the protocol) that generates a particular network flow. There is a growing need for traffic classification. Many tasks that are necessary for network operation and management as well as for business models based on providing network access can benefit from traffic classification. Traffic classification can be used for QoS (quality of service) mapping, traffic shaping, access control, content control/filtering, intrusion detection and prevention, trend analysis, monitoring, lawful interception, content optimization, billing and metering, load balancing, traffic engineering, network planning, etc.

In general, there are four kinds of traffic classification methods. The oldest and most common method is the *port based* approach. This uses the well-known-port numbers of the TCP/UDP protocols assigned by the IANA. Many client-server applications or protocols use asymmetric port numbers for client and server, which means that client port and server port differ. The port based method mostly refers to the server port to identify an application. But the server port can be set to any port number by the server administrator. Not every protocol own well-known port numbers or they use dynamic ports like P2P (peer-to-peer) protocols. By using tunnels, this method fails too. Therefore, we cannot trust in this method.

Another method used is *protocol decoding*. It is based on stateful reconstruction of sessions and application information from packet content. It identifies protocols by their characteristic protocol headers (magic numbers, incrementing counters, session identifiers, etc.), packet sequences, etc. so it avoids needing to trust in port numbers. This method provides high accuracy but it is very expensive. Every protocol detection must be implemented manually and in-depth knowledge of the entire protocol and the network environment is necessary. Problems of this method are the amount of network protocols and keeping it up to date. Furthermore proprietary protocols must be reverse engineered and encrypted traffic is out of scope for protocol decoding. Therefore this method is often used only for dedicated popular protocols, e. g., HTTP and mail protocols like in Cisco's Network Based Application Recognition (NBAR) [1].

The third method is the *pattern* or *signature based* approach [2]. This method uses application specific signatures and searches for those in the protocol header and content to identify the application. The problem of this method is to find good pattern. Furthermore pattern matching for many patterns across all network traffic can become very expensive especially for higher data rates. Additionally, the protocol detection can take up to 100 seconds or more [2].

The fourth method is based on the *machine learning* approach. This method uses machine learning algorithms as used in data mining to identify applications by characteristic packet or flow statistics. The advantage of this approach is that the algorithms can be trained with real network traffic. If a protocol changed or a new protocol appeared, it is easy to repeat the training to update the protocol identifier. Probably it can be used to identify some encrypted protocols. A problem of this method is to find the proper parameters and effective machine learning algorithms.

The machine learning approach has been discussed in numerous papers [3], [4], [5], [6], but with focus on just one algorithm. In this paper we evaluate a wide range of candidate algorithms and parameters. Furthermore, we evaluate which of these algorithms are suitable for real-time traffic classification. In this context, real-time means that traffic management (e. g., traffic shaping) can be done

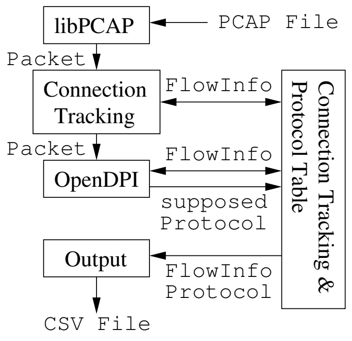

Figure 1: Block diagram of automatic flow labelling

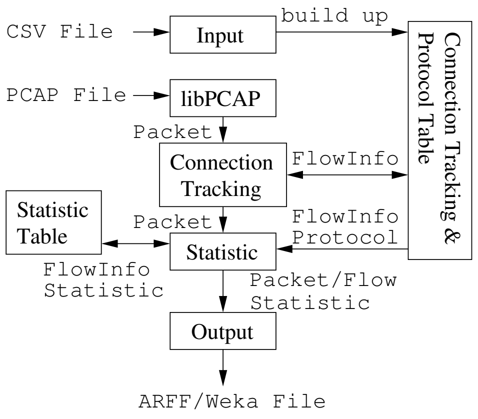

Figure 2: Block diagram of statistics computation

immediately. This approach is mostly used for non real-time or offline network traffic analysis [3], [4], [6].

The remainder of this paper is structured as follows: Section 2 contains a description of the traffic classification used as the basis for our research. The statistical parameters applied and the classifying algorithms examined can be found in the succeeding Sections 3 and 4. Section 5 describes the experimental setup of our research and Section 6 covers the parameter reduction. Finally, Section 7 provides a conclusion and the direction for our future work.

II. Traffic classification Tools

To test and train the different machine learning algorithms (see Section IV) we used different network traffic traces in PCAP (packet capture library) format [7]. To extract the necessary information from these capture files, we developed a tool to generate adequate input data to the algorithms. We had to develop our own tool because available tools like GTVS (Ground Truth Verification System) [8] do not fulfil our constraints for the automatic traffic labelling and we used many different statistical parameters (see Section III), which were not included in GTVS.

The processing and labelling of the PCAP file is done in two consecutive steps: First, the automatic protocol detection and labelling and second, computation of statistics. The result of the first step is a text file in CSV (comma separated values) format. This CSV file is used for manual post-processing to check for correctly labelled flows or to label unknown — not automatically detected traffic — manually. To detect and label the flows automatically we use OpenDPI [9], which is a library that performs protocol detection by stateful reconstruction of session and application information from packet content. Other tools, e. g., GTVS [8] use heuristics to classify and label the traffic. But we decided not to use this tool because it is based on port numbers, which are not aware of, e. g., tunnels etc. (see Section I).

The structure and function of the automatic traffic labelling is shown in the block diagram on Fig. 1. The libPCAP-API [7] is used to read single packets from a PCAP file. A connection tracking is needed to work on a per-flow basis. The connection tracking contains a TCP state machine. As a result the connection tracking provides only those packets belonging to a correct TCP flow that was already seen from beginning of the flow (3-way-hand-shake). Many protocols may only be detected if the first packets of the flow — which in many protocols contain a handshake — are present (e. g., SSL/TLS). Therefore the connection tracking is very important. To deal with the stateless nature of UDP we used timeouts to split the UDP traffic for the same 5-tuple into different flows. The necessary flow information (e. g., current state machine state) is saved to the connection tracking table with the 5-tuple (source/destination IP address, source/destination port, the IP protocol field value) as key.

The second step in PCAP processing is to compute the statistical parameters. Fig. 2 shows the block diagram of the flow based statistic generation. The CSV-File from first step is used to build up a flow table that contains the detected protocol, the flow start timestamp and the 5-tuple as key. The start timestamp is prerequisite to differentiate flows with the same 5-tuple. The computation of the statistical parameters is done for every packet of a flow. A detailed description of the parameters and their computation is in Section III.

The output of the statistics processing is in ARFF (Attribute-Relation File Format)/WEKA file format [10], see Section IV. The output contains separate training and testing data. Furthermore there are two kinds of statistical data: (1) per-packet data (2) per-flow data. The per-packet data can be used to determine if the classifiers/algorithms can be used for real-time classification tasks. The per-flow data can be used to determine the quality of the classifiers/algorithms applied to offline data or non real-time data like NetFlow (NetFlow is an industry standard protocol for traffic monitoring by collecting IP traffic information) data used for monitoring or forensics.

Traffic classification on a per-packet basis makes it possible to intercept the network traffic for management and traffic engineering in real-time. On a per-flow basis the flow has finished when it is classified. So, the classification can only be used for monitoring or statistics.

III. Statistical Parameters

The basis for traffic classification using heuristic algorithms are objects which can be classified by the algorithms used. For this approach, we are using traffic flows as the foundation for our classification. We define a traffic flow as one or more related IP-packets between two connected hosts. Each flow is characterized by a 5-tuple consisting of the source and destination IP-address, the network layer protocol number and the source and destination port (referring to TCP and UDP). For TCP based flows we are using only complete TCP flows. This implies that the investigated TCP flows have a 3-Way-Handshake for the connection establishment and a connection termination, for example with the 4-Way-Handshake or the reset flag (RST). To collect those complete TCP flows we used a TCP state machine we implemented in the tool described in Section II. Incomplete or fragmented TCP flow traces are subject of our further ongoing research.

A. Flow parameters

Besides the 5-tuple as the unique qualifier, each flow is described by additional parameters. The most parameters are differentiated for the whole flow, for upstream and for downstream. The reason for this differentiation can be found in the particular differences in upstream and downstream behaviour of various protocols, e. g., HTTP. Our selection of flow characterizing parameters is shown in Table I.

According to [11] we describe each flow with three modes:

- **idle**: A flow is idle when no packets were sent between the two hosts for more than two seconds.
- **bulk**: The bulk mode is defined as the time when there are more than three packets being successfully delivered in the same direction without any packet with data in the other direction.
- **interactive**: When there are packets sent in both directions, the flow is in the interactive mode.

Because the interactive mode is also defined as the time when the flow is not in the idle or in the bulk mode — which means that the interactive mode correlates with the other two modes, we decided to gather only the duration and quota for idle and bulk mode. Furthermore we provide information about the interarrival time, whole flow duration, packet count and the payload size.

B. Parameter computation

As shown in [11], there are more possible flow characterizing parameters than we present in Table I. But, a lot of these parameters are not correlated with the used protocol. Instead, they are only influenced by the transport network, e. g., the Internet. For example, the total number of duplicated SACK packets ([11]: parameter no. 39 and no. 40) only indicates packet loss in the network and is a result of TCP congestion control. These network influences are independent from the protocol characteristics.

Also, often the transport layer (UDP and TCP) port numbers are used as flow characteristics [4], [5], [11], [12]. Because of the use of network address translation (NAT) or by intentional administrative changes of server port numbers, these characteristics are not reliable for traffic classification. Other parameters, like the median of bytes on wire, are depending on the used network stack implementation. The number of bytes on wire, e. g., Ethernet, are influenced by the different possible options in the IP and TCP header or the use of IPv4 and IPv6. Thus, there will be a packet with the same transport layer payload with a different number of bytes on the wire and this is not dependent on the classifying protocol.

The target of our research is the usage of heuristic algorithms for real-time traffic classification. Therefore, we cannot compute the flow characterizing parameters at the end of the flow. Instead, we compute all parameters after each packet of the flow is received. Hence, we compute moving averages, like the median interarrival time or the average byte count of the network layer payload. Required aggregate values are also computed as moving values for each received packet of the flow.

IV. Algortihms

To classify the network traffic we are using different machine learning algorithms, which are also called classifiers. All the classifiers we used are off the shelf machine learning algorithms and we treat them as black-box classifiers. For this work we used the WEKA software suite [10]. The WEKA suite is written in Java language and contains a collection of machine learning algorithms for data mining tasks. WEKA provides a uniform interface for all classifiers, which made it easy to automate training and testing of different classifiers.

A. Train and Test suite

The records, generated by the tool described in Section II and used as training and testing data for the classifiers, are provided as ARFF files. An ARFF file is an ASCII text file that contains (1) a header Section, which describes the used parameters/attributes and (2) the data Section, which contains the records. Each record is represented by one line containing all parameters in a comma separated list. The training record contains all statistical parameters and the associated protocol. Therefore, we can use the supervised learning approach for the machine learning algorithms. The test records contain only the statistical parameters.

If a classifier is trained by WEKA it generates a classifier model that can be saved to a file. In a test case this model can be used by the classifier to make a prediction of the

Table I: Used flow and packet parameters and their description

No.	Class	Short Name	Description
1	1	packet_cnt	packet count whole flow
2	1	packet_cnt_us	packet count upstream
3	1	packet_cnt_ds	packet count downstream
4	2	intarv_time_med	median interarrival time whole flow
5	2	intarv_time_max	maximum interarrival time whole flow
6	2	intarv_time_min	minimum interarrival time whole flow
7	2	intarv_time_med_us	median interarrival time upstream
8	2	intarv_time_max_us	maximum interarrival time upstream
9	2	intarv_time_min_us	minimum interarrival time upstream
10	2	intarv_time_med_ds	median interarrival time downstream
11	2	intarv_time_max_ds	maximum interarrival time downstream
12	2	intarv_time_min_ds	minimum interarrival time downstream
13	3	bytes_payload_l4_med	median byte count of L4 payload whole flow
14	3	bytes_payload_l4_max	maximum byte count of L4 payload whole flow
15	3	bytes_payload_l4_min	minimum byte count of L4 payload whole flow
16	3	bytes_payload_range	(maximum - minium) byte of L4 payload whole flow
17	3	bytes_payload_l4_med_us	median byte count of L4 payload upstream
18	3	bytes_payload_l4_max_us	maximum byte count of L4 payload upstream
19	3	bytes_payload_l4_min_us	minimum byte count of L4 payload upstream
20	3	bytes_payload_range_us	(maximum - minium) byte of L4 payload upstream
21	3	bytes_payload_l4_med_ds	median byte count of L4 payload downstream
22	3	bytes_payload_l4_max_ds	maximum byte count of L4 payload downstream
23	3	bytes_payload_l4_min_ds	minimum byte count of L4 payload downstream
24	3	bytes_payload_range_ds	(maximum - minium) byte of L4 payload downstream
25	4	duration_flow	whole connection duration
26	4	duration_flow_us	connections duration on upstream
27	4	duration_flow_ds	connections duration on downstream
28	5	changes_bulktrans_mode	number of transitions between bulk- and transfermode
29	5	duration_bulkmode	time spent in bulk transfer mode
30	5	duration_bulkmode_us	time spent in bulk transfer mode for upstream
31	5	duration_bulkmode_ds	time spent in bulk transfer mode for downstream
32	5	quota_bulkmode	percentage of quota of bulk transfer mode (per mille)
33	5	quota_bulkmode_us	percentage of quota of bulk transfer mode (per mille) on upstream
34	5	quota_bulkmode_ds	percentage of quota of bulk transfer mode (per mille) on downstream
35	6	time_idle_sum	time spent in idle mode for whole flow
36	6	time_idle_sum_us	time spent in idle mode for upstream
37	6	time_idle_sum_ds	time spent in idle mode for downstream
38	6	quota_time_idle	percentage of quota of time spent in idle mode (per mille)
39	6	quota_time_idle_us	percentage of quota of time spent in idle mode (per mille) on upstream
40	6	quota_time_idle_ds	percentage of quota of time spent in idle mode (per mille) on downstream

protocol which was the origin of the statistical record. Due to the generation of training and testing data from one PCAP file by the tool in Section II, we can evaluate the quality of prediction by comparing the classifier results with the training records containing the protocol. This PCAP file must be a different file than the one which was used to generate the classifier model. To automate training and testing of different classifiers we wrote a test suite on top of WEKA. Fig. 3 shows a block diagram of the test suite. The test suite provides parallel WEKA instances to speed up training and tests. Furthermore, the test suite can measure the memory and time consumption the classifiers need for training and testing.

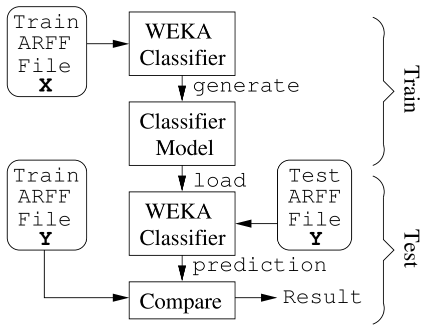

Figure 3: Block diagram of the test suite

B. Preselection of Algorithms

The WEKA software suite contains more than 100 classifiers. So, it was necessary to choose the best candidates of these classifiers to reduce the amount of time to train, test and evaluate the classifier results.

In a first preselection of the classifiers, we used a training-set that was also used as test-set. This means a good classifier should correctly predict a preponderant part of data. We dismissed classifiers with accuracy below 90%. The second criterion for preselection was the time needed for training and testing. The third criterion was the memory consumption of the classifiers. If a classifier needed more than 2 GByte of RAM for this test, it was also dismissed. The preselected classifiers are shown in the left row of Table II.

We used the classifiers as black-box with the default parameters as proposed by WEKA.

V. Experimental Setup

For training and testing we used the test suite described in Section IV. The used data and the results of our experiments are shown below in this section.

A. Used Data

During this study we have used data collected from different sources. We manually generated network traffic and captured network traffic from the campus network. Additionally we used trace files from [13], [14].

All captured network traffic was automatically classified with the tool described in Section II. The whole traffic was verified by hand. False or not classified traffic was completed by hand. Then, we split this data into two parts and generated the training and testing data in ARFF-format. Table III contains the composition of network packets for training and testing data.

Many of the investigated protocols use cryptographic encryption to protect their data, making it difficult to classify these protocols. Bittorrent, eDonkey, IMAP, POP3 and SMTP use encryption. IMAP, POP3 and SMTP can cryptographically protect their communication before they start their session or they can request encryption during the session (STARTTLS). In the second case the establishment of the session is not encrypted. Our data contains both kinds of IMAP, POP3 and SMTP traffic. The other protocols do not use encryption. Also, the HTTP traffic contains only unencrypted traffic; it does not contain HTTPS (TLS) traffic.

The results of the test run for all classifiers with the computed classifier-models and the particular test data are deterministic.

B. Results

Table II contains the results of this study. It shows if it is possible for an algorithm with any parameter combination to detect a protocol with an accuracy greater than or equal to 90%.

Additionally, Table II contains the time needed for training and testing. These times are measured by using the HPROF tool for heap and CPU profiling, which is part of the Java Virtual Machine. We measured the CPU usage time for every algorithm and used the fastest algorithm (OneR) as reference to scale the timing results. As we can see, in Table II the amount of time spent for training is much higher than for testing. But, this is not a problem in general. Training is done only once but testing is done permanently for protocol classification. All the algorithms have very similar CPU time consumption, but four algorithms (BayesNet, NaiveBayes, NaiveBayesUpdateble and ND) have a notedly higher CPU time consumption. This could make these four algorithms unusable for real-time traffic classification. Furthermore, the algorithms NaiveBayes and NaiveBayesUpdateble provide nearly the same results, which can be seen in Table II as well as in Fig. 4.

It can be seen from Table II that not all used algorithms are suitable for protocol classification with the selected statistical parameters. Also, it shows that the observed protocols have very different behaviour, so that some could be detected with high accuracy (Bittorrent, RTP, Flash) while other protocols (eDonkey, IMAP) are hard to detect.

Table III: Used training and testing data (number of packets)

Protocol	Training	Testing	Encryption
Bittorrent	7772	10461	yes
eDonkey	19961	48420	yes
Flash	21373	21212	no
HTTP	4743	1236	no
IMAP	2797	1025	yes
Oscar	315	257	no
POP3	673	3044	most
RTP	24404	57428	no
SIP	414	760	no
SMTP	560	774	yes

VI. Parameter Reduction

As described in Section III and listed in Table I, we used 40 parameters, which can be divided into six different parameter classes. Table II shows on which protocols our 18 preselected classifiers have a classification accuracy with greater than or equal to 90%. However, Table II does not show which parameter combinations were suitable for the best classification results.

One important target of our research is the reduction of the used parameters — first, to eliminate unnecessary or disruptive parameters according to the classification accuracy and second, to eliminate parameters and thus reduce the complexity of the classification for optimization according to real-time classification. Therefore, we tested all classifiers with all possible combinations of the parameter classes from Table I. Hence, we tested our 18 classifiers with our training PCAP trace and all 63 combinations of the parameter classes — the 64th combination (all classes removed) was omitted. The results of these tests are evaluated below in this section.

The evaluation of the test results are shown in different histograms. Each parameter combination is expressed as a decimal number in these histograms (see Fig. 4, Fig. 5, Fig. 6 and Fig. 7). These decimal numbers are the sum of the values of each parameter class as shown in Table IV. For example, the combination of the payload size (4) and bulk (16) parameters is the decimal number $20 = 4 + 16$.

Table II: Protocol classification with greater than or equal to 90% accuracy and time factor

WEKA Classifier Name	Bittorrent	eDonkey	Flash	HTTP	IMAP	Oscar	POP3	RTP	SIP	SMTP	Time Factor	
											Train	***Test***
AttributeSelectedClassifier	✓		✓			✓	✓	✓	✓	✓	27.8	2.8
Bagging	✓		✓			✓	✓	✓		✓	127.4	1.4
BayesNet	✓		✓			✓	✓	✓	✓	✓	40.0	42.5
DataNearBalancedND	✓		✓	✓	✓	✓	✓	✓	✓	✓	92.0	5.2
DecisionTable	✓		✓				✓	✓	✓		388.4	1.9
FilteredClassifier	✓	✓	✓			✓		✓	✓		21.9	2.1
J48	✓	✓	✓	✓		✓	✓	✓	✓	✓	37.7	2.1
J48graft	✓	✓	✓	✓		✓	✓	✓	✓	✓	49.6	3.0
NaiveBayes	✓		✓	✓			✓	✓	✓	✓	54.1	118.9
NaiveBayesUpdateable	✓		✓	✓			✓	✓	✓	✓	54.7	119.4
nestedDichotomies.ND	✓		✓	✓		✓	✓	✓	✓	✓	137.2	42.7
OneR	✓	✓	✓					✓	✓		5.0	1.0
PART	✓		✓	✓		✓	✓	✓	✓	✓	60.2	2.2
RandomCommittee	✓	✓	✓	✓		✓	✓	✓	✓	✓	42.3	2.3
RandomForest	✓		✓	✓		✓	✓	✓	✓	✓	35.0	2.3
RandomSubSpace	✓	✓	✓	✓		✓	✓	✓	✓	✓	71.6	1.9
RandomTree	✓		✓	✓	✓	✓	✓	✓	✓	✓	5.3	1.1
REPTree	✓		✓			✓	✓	✓		✓	13.7	1.3

Table IV: Binary coding of parameter combinations

Binary	Decimal	Parameter Class
2^0	1	packet count
2^1	2	interarrival
2^2	4	payload size
2^3	8	duration
2^4	16	bulk
2^5	32	idle

A. *Classifier Based Parameter Reduction*

The first basic approach to reduce the parameters is to evaluate all classifiers according to their classification accuracy with the different classifier combinations. Therefore we first filtered all parameter combinations for each classifier which have a classification accuracy greater than or equal to 90% on each of the protocols of Table III. Afterwards, we counted for each classifier, which of these parameter combinations were present on at least five protocols. The results of this evaluation are shown in the histograms of Fig. 4. For each of the 18 preselected classifiers one histogram is shown.

As you can see in the histograms of Fig. 4 nearly two-thirds of all possible parameter combinations — 39 out of 63 — have a classification accuracy of greater than or equal to 90% on at least five protocols with the used classifiers. Thus, one third of all parameter combinations is not reaching this limit and can already be dismissed for the further evaluation. Furthermore, on 32 of these 39 parameter combinations the parameter class payload size is included, which means that all possible parameter combinations with the parameter class payload size is present on at least one classifier. The other five parameter classes are only present in combinations with other parameter classes and not on their own like payload size (4).

Comparing the histograms in Fig. 4 with the results of Table II illustrates the low accuracy of the classifiers DecisionTable, OneR and filteredClassifier. In the histograms there are only fewer or no parameter combinations which have a classification accuracy greater than or equal to 90% on at least five protocols. Furthermore the best classifiers with the most classification accuracy greater than or equal to 90% in Table II are DataNearBalancedND, J48, J48graft, RandomCommittee, RandomSubSpace and RandomTree with nine out of ten protocols. But, the histograms in Fig. 4 show no parameter combination on these classifiers which can be used on all of the nine protocols. With the classifiers DataNearBalancedND, J48 and J48graft it is only possible to classify at the most seven protocols with one parameter combination to get a classification accuracy greater than or equal to 90%. For example, for DataNearBalancedND it is the parameter combination 36 (payload size and idle) and for J48 the two parameter combinations 12 (payload size and duration) and 44 (payload size, duration, idle). The classifier J48graft has six possible parameter combinations which can be used to classify seven protocols with a classification accuracy greater than or equal to 90%.

Like the classifiers RandomCommittee, RandomSubSpace and RandomTree, the classifiers PART and RandomForest have parameter combinations which can be used to classify eight different protocols with a classification accuracy greater than or equal to 90%. For all of these classifiers, the eight protocols which can be classified with one parameter combination having a classification accuracy greater than or equal to 90% are: Bittorrent, Flash, HTTP, Oscar, POP3,

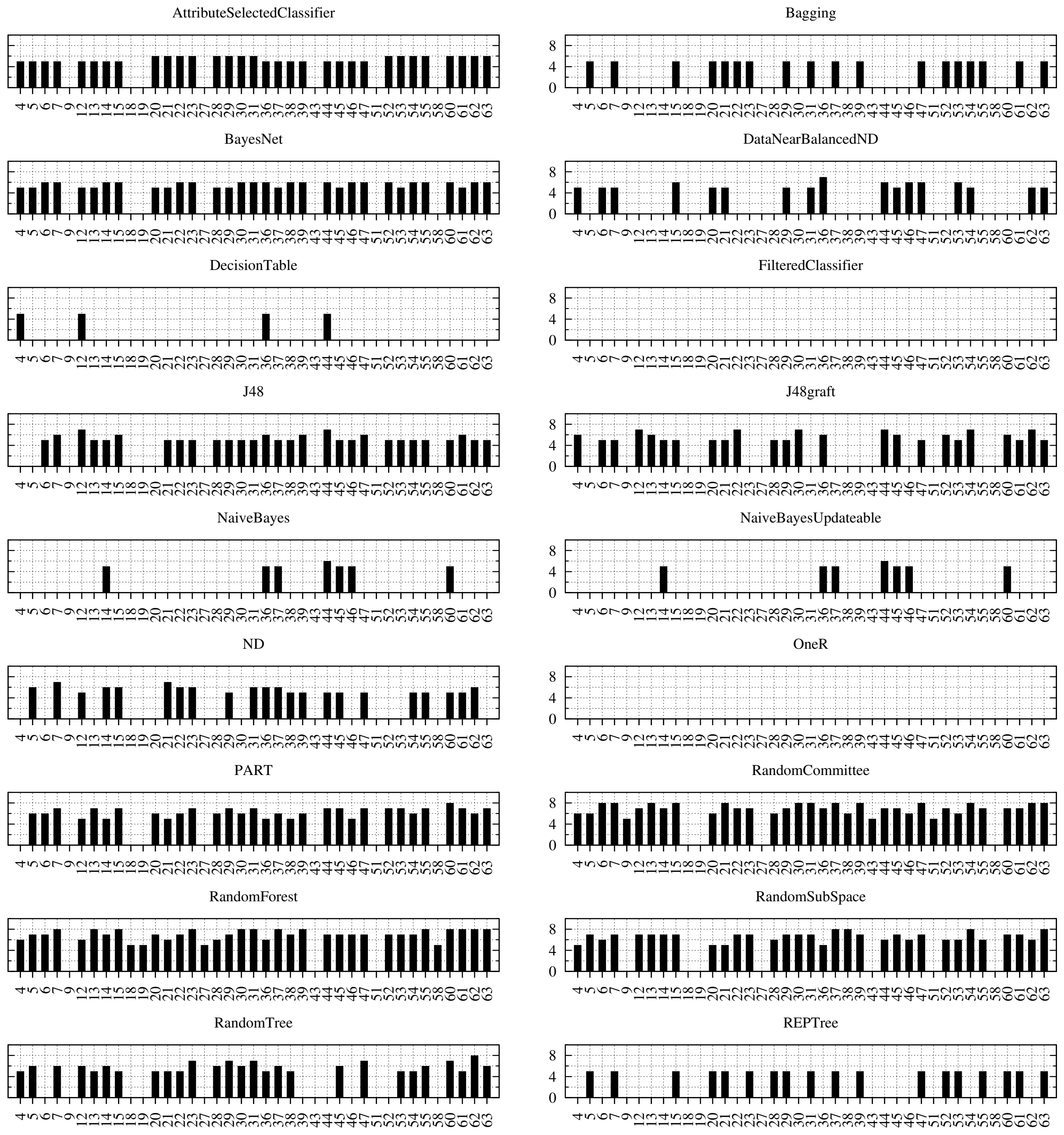

Figure 4: Distribution of parameter combinations for classifiers
(x-axis: parameter combinations, y-axis: number of occurrences)

RTP, SIP and SMTP. In fact, all of the possible parameter combinations contain the parameter class payload size. In addition, the most parameter combinations consist of at least three parameter classes. The only exception is the parameter combination 6 (interarrival and payload size) for the RandomCommittee classifier.

Fig. 5 shows a histogram with the number of classifiers on which each of the 39 parameter combinations has a classification accuracy of greater than or equal to 90% at least on five protocols. There are 16 parameter combinations which facilitate a wide spectrum — at least two-thirds of all classifiers (12 out of 18) — on the protocol classification: 7, 14, 15, 21, 29, 31, 36, 37, 44, 45, 47, 53, 54, 60, 61, 63. Also, these 16 parameter combinations contain the parameter class payload size and the most parameter combinations consist of at least three parameter classes — only exception is the parameter combination 36 (payload size and idle). In addition, the parameter combinations which are only present

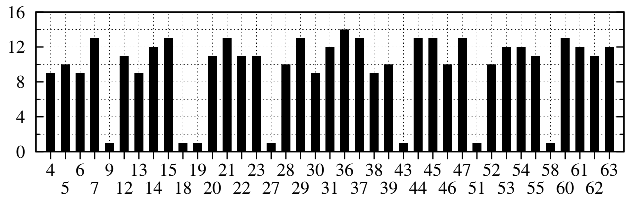

Figure 5: Number of parameter combinations for classifiers (x-axis: parameter combinations, y-axis: number of classifiers)

on one classifier (9, 18, 19, 27, 43, 51, 58) do not contain the parameter class payload size.

Finally, the results of these tests show that combinations of many or all parameter classes do not imply higher accuracy on the classification of the investigated protocols. But, for a good classification accuracy according to the classification of as many protocols as possible with one parameter combination, mostly combinations of three or more parameter classes are required.

B. Protocol Based Parameter Reduction

In contrast to the evaluation of all classifiers according to their classification accuracy with the different parameter combinations, the second approach to reduce the parameters for the classification is to evaluate the classification accuracy of the different parameter combinations depending on the network protocols. As in the first basic approach, we started with filtering all parameter combinations which have a classification accuracy greater than or equal to 90%. In the next step we counted for each protocol, which of these parameter combinations were present on at least nine different classifiers. The results of this evaluation are shown in the histograms of Fig. 6. For each of the ten protocols from Table III one histogram is shown.

The histograms of Fig. 6 include all 63 parameter combinations, but the protocol classes interarrival, duration, bulk and idle are mostly present in combination with other parameter classes and not on their own.

Table II shows some protocols (Bittorrent, Flash, POP3, RTP, SIP) which have a good classification accuracy with the most classifiers. The histograms of these protocols in Fig. 6 indicate this fact with a lot of possible parameter combinations having a classification accuracy of greater than or equal to 90% on at least nine classifiers. Furthermore, there are parameter combinations of the protocols Bittorrent, Flash and RTP with a classification accuracy of greater than or equal to 90% on all 18 classifiers. For Bittorrent these are 16 (bulk) and 48 (bulk and idle), for Flash these are 29 (packet count, payload size, duration, bulk) and 53 (packet count, payload size, bulk, idle) and for RTP these are 1 (packet count), 32 (idle) and 33 (packet count, idle).

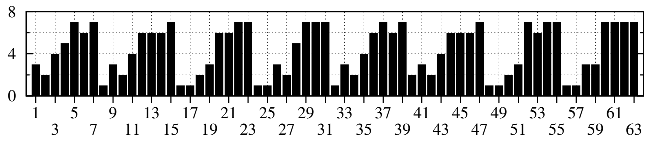

Figure 7: Number of parameter combinations for protocols (x-axis: parameter combinations, y-axis: number of protocols)

Fig. 7 shows a histogram with the number of protocols on which each of the parameter combinations has a classification accuracy of greater than or equal to 90% on at least nine classifiers. There are 18 parameter combinations which facilitate a wide spectrum — at least two-thirds of all protocols (7 out of 10) — on the protocol classification: 5, 7, 15, 22, 23 ,29, 30, 31, 37, 39, 47, 52, 54, 55, 60, 61, 62, 63. Also, these 18 parameter combinations contain the parameter class payload size and the most parameter combinations consist of at least three parameter classes — only exception is the parameter combination 5 (packet count and payload size). In addition, the parameter combinations which are only present on one protocol (8, 16, 17, 24, 25, 32, 48, 49, 56, 57) do not contain the parameter class payload size.

Additionally, the results of these tests show that combinations of many or all parameter classes do not imply higher accuracy on the classification of the investigated protocols. But, for a good classification accuracy according to the classification of a protocol with as many classifiers as possible with one parameter combination, mostly combinations of three or more parameter classes are required.

VII. Conclusion and Further Work

In this paper, we have determined how accurately 18 machine learning algorithms can be used to classify network traffic with 63 combinations of six statistical parameter classes consisting of 40 parameters. In contrast to many other studies, we used a per-packet approach instead of the per-flow approach.

As a result, we can say that the parameter class payload size has a significant influence on the classification accuracy. The four algorithms BayesNet, NaiveBayes, NaiveBayesUpdateble and ND have a time factor for testing, which is between 40 and 120 times greater than the fastest algorithm OneR. Thus these algorithms appear to be unsuitable for real-time traffic classification on high data rate links. The other algorithms have a time factor with nearly the same dimension. The algorithms DesicionTable and OneR reach only a low accuracy with the used parameters. The DataNearBalancedND, J48, J48graft, RandomCommittee, RandomSubSpace, RandomTree algorithms classified the most network protocols with an accuracy of 90% or more. The algorithms PART, RandomCommittee, RandomForest,

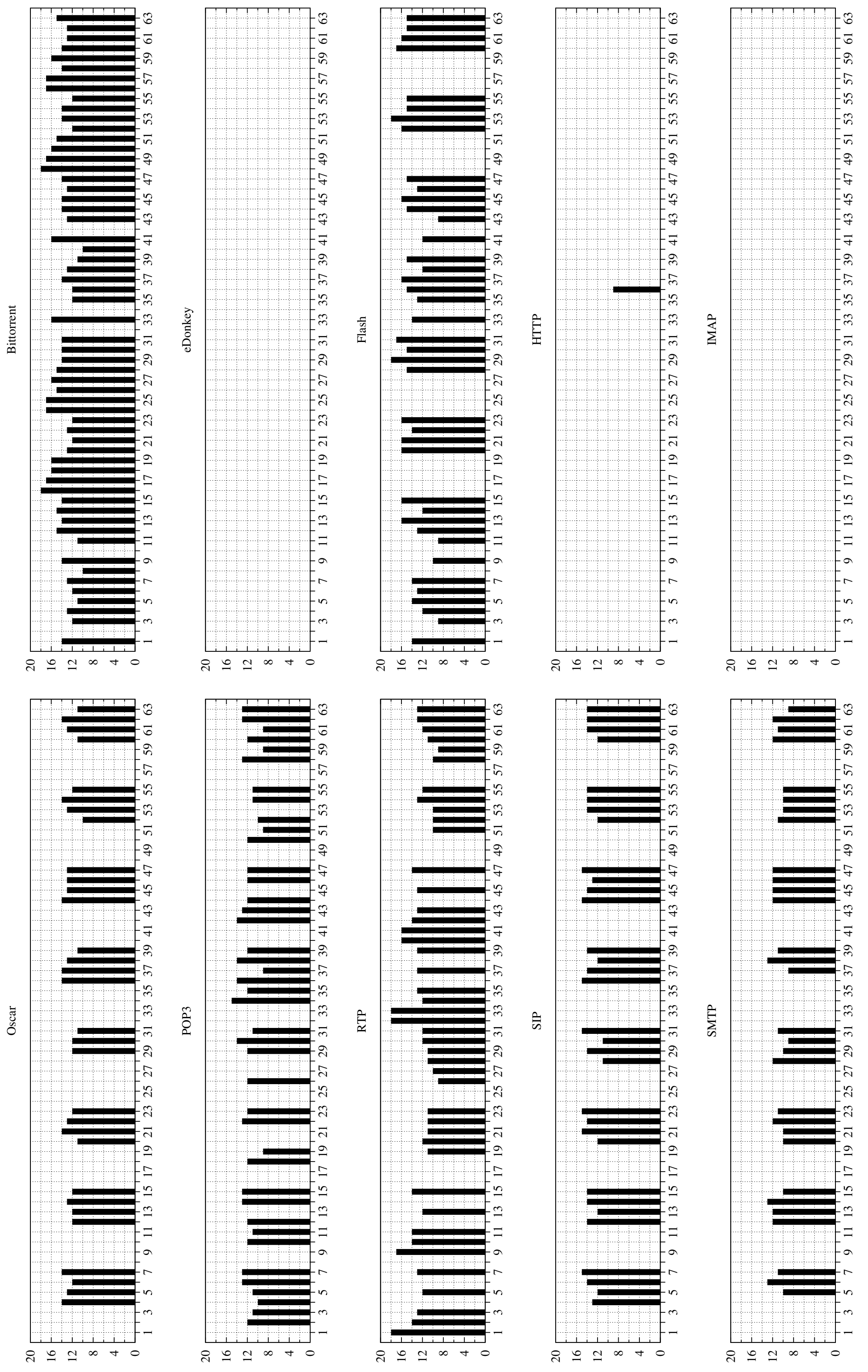

Figure 6: Distribution of parameter combinations for protocols
(x-axis: parameter combinations, y-axis: number of occurrences)

RandomSubSpace and RandomTree could classify eight of ten protocols with one parameter combination with an accuracy of 90% or more. Also the results show also that many parameters do not automatically cause a better classification. Furthermore, parameter reduction can improve the performance of per-packet classification which is necessary for the real-time traffic analysis of "fast" networks. A classification accuracy of 90% and above is a good result in comparison to other papers [3], [4], [5], [6], [15] where the classification accuracy often falls below 90%.

As a main result, we can say that the parameter combination needed for good classification results is depending on the machine learning algorithm used and the protocols to classify. There is no parameter combination, which can be used on all classifiers to get a high classification accuracy on all protocols. This implies that the results of the other studies [3], [4], [5], [6] can only be used with the same algorithm used in the particular paper.

Further Work

Other studies (e. g., [11]) showed that classification accuracy is decreasing by tests on network traffic of different locations. So we have to evaluate our results with other traces. To enhance the accuracy of traffic classification we have to find other parameters. All common statistical parameters were determined in this paper. New parameters should represent more characteristic properties of network protocols and their payload.

Another way to enhance the accuracy could be to split the protocols into those using TCP and those using UDP. Furthermore the parameter classes we are using consist of different parameters that represent information about upload, download and the whole flow. We will determine the benefit of this distinction, because possibly some of them can be dismissed.

Another field of research is the real-time classification of network traffic with machine learning algorithms.

Furthermore, we want to investigate if it is possible with machine learning algorithms to train the algorithms for different protocol classes like e-mail, bulk data transfer, P2P, interactive, gaming or multimedia to be able to classify unknown protocols which belong to a protocol class.

Finally, it would be interesting to study the best suitable algorithms in detail — instead of using them as a black box model — in order to improve them for better classification, to find out why they are more suitable than the others, and to find systematics which explains why there are some algorithms which are more suitable than others.

REFERENCES

[1] "Network Based Application Recognition (NBAR)," Cisco®, Oct. 2011, http://www.cisco.com/en/US/products/ps6616/products_ios_protocol_group_home.html.

[2] "Application Layer Packet Classifier for Linux," Oct. 2011, http://l7-filter.sourceforge.net.

[3] L. Bernaille, R. Teixeira, I. Akodkenou, A. Soule, and K. Salamatian, "Traffic classification on the fly," vol. 36. New York, NY, USA: ACM, April 2006, pp. 23–26. [Online]. Available: http://doi.acm.org/10.1145/1129582.1129589

[4] H. Jiang, A. W. Moore, Z. Ge, S. Jin, and J. Wang, "Lightweight application classification for network management," in *Proceedings of the 2007 SIGCOMM workshop on Internet network management*, ser. INM '07. New York, NY, USA: ACM, 2007, pp. 299–304. [Online]. Available: http://doi.acm.org/10.1145/1321753.1321771

[5] A. W. Moore and D. Zuev, "Internet traffic classification using bayesian analysis techniques," in *Proceedings of the 2005 ACM SIGMETRICS international conference on Measurement and modeling of computer systems*, ser. SIGMETRICS '05. New York, NY, USA: ACM, 2005, pp. 50–60. [Online]. Available: http://doi.acm.org/10.1145/1064212.1064220

[6] S. Zander, T. Nguyen, and G. Armitage, "Automated traffic classification and application identification using machine learning," in *Local Computer Networks, 2005. 30th Anniversary. The IEEE Conference on*, Sydney, NSW, nov. 2005, pp. 250 –257.

[7] "TCPDUMP & LibPCAP," http://www.tcpdump.org.

[8] M. Canini, W. Li, and A. W. Moore, "GTVS: boosting the collection of application traffic ground truth," University of Cambridge, Computer Laboratory, Tech. Rep. UCAM-CL-TR-748, Apr. 2009. [Online]. Available: http://www.cl.cam.ac.uk/techreports/UCAM-CL-TR-748.pdf

[9] "OpenDPI.org," http://www.opendpi.org.

[10] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The WEKA Data Mining Software: An Update," *SIGKDD Explorations*, vol. 11, no. 1, 2009.

[11] A. W. Moore, M. L. Crogan, and D. Zuev, "Discriminators for use in flow-based classification," Queen Mary University of London, Tech. Rep., 2005.

[12] T. T. Nguyen and G. Armitage, "A survey of techniques for internet traffic classification using machine learning," *IEEE In Communications Surveys & Tutorials*, vol. 10, no. 4, pp. 56–76, 2008.

[13] "OpenPacket.org," https://openpacket.org.

[14] "Wiki site for the Wireshark network protocol analyzer," http://wiki.wireshark.org/SampleCaptures.

[15] P. Piskac and J. Novotny, "Using of time characteristics in data flow for traffic classification," in *Proceedings of the 5th international conference on Autonomous infrastructure, management, and security: managing the dynamics of networks and services*, ser. AIMS'11. Berlin, Heidelberg: Springer-Verlag, 2011, pp. 173–176. [Online]. Available: http://dl.acm.org/citation.cfm?id=2022216.2022243