Automatic Protocol Signature Generation Framework for Deep Packet Inspection

Géza Szabó
TrafficLab, Ericsson Research
Hungary
geza.szabo
@ericsson.com

Zoltán Turányi
TrafficLab, Ericsson Research
Hungary
zoltan.turanyi
@ericsson.com

László Toka
HSN Lab, BUTE, Hungary
Eurecom, France
toka@tmit.bme.hu

Sándor Molnár
High Speed Networks Lab.
Dept. of Telecomm. and
Mediainformatics
Budapest Univ. of Technology
and Economics
molnar@tmit.bme.hu

Alysson Santos
Networking and Telecomm.
Research Group
Universidade Federal de
Pernambuco, Recife, Brazil
alysson@gprt.ufpe.br

ABSTRACT

We present an automatic application protocol signature generating framework for Deep Packet Inspection (DPI) techniques with performance evaluation. We propose to utilize algorithms from the field of bioinformatics. We also present preprocessing methods to accelerate our system. Moreover, we developed several postprocessing techniques to refine the accuracy of the results. Finally, we propose a DPI system, based on *approximate* string matching, and find it a viable, novel alternative for the refinement of *exact* string matching algorithm's results.

Keywords

deep packet inspection, automatic protocol signature generation, motif finding

1. INTRODUCTION

In-depth understanding of the Internet traffic profile is a challenging task for researchers and a mandatory requirement for most Internet Service Providers (ISP). Deep Packet Inspection (DPI) can aid to ISPs in the profiling of networked applications. With this information ISPs may then apply different charging policies, traffic shaping and offer different quality of service guarantees to selected users or applications. Current DPI tools and techniques rely on comparing the content of the packet payload with a set of strings or regular expressions, which essentially assumed to represent a given "signature" of an application. The collection and definition of the proper signatures is a time consuming, challenging task requiring lot of manual work from protocol experts. To ease this manual work automatic protocol signature generation tools help to process the network traces of a specific application and define signature candidates.

VALUETOOLS 2011, May 16-20, Paris, France
Copyright © 2011 ICST 978-1-936968-09-1
DOI 10.4108/icst.valuetools.2011.245606

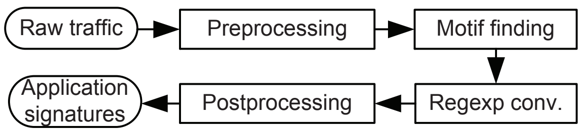

Figure 1: The proposed framework

Automatic signature generation is cumbersome due to the following requirements that it must fulfill:

- it should be automatic to as high extent as possible;
- it should process a high number of samples within a reasonable time period;
- it should provide the most descriptive (often this longest) possible signature candidates;
- it should find important signatures to represent the underlying traffic well.

In this paper we propose a framework for automatic signature generation. This framework is built on two basic algorithms: *motif finding* and *sequence alignment* inspired by related methods from bioinformatics. Moreover, we also propose *preprocessing* and *postprocessing* methods to accelerate the system and increase its accuracy. As a result we have a system with several building boxes, see Figure 1. Throughout the paper the following notation is used to denote the various processing steps. 'P' for preprocessing, 'MF' for motif finding, 'R' for Regexp conversion and 'Po' for Postprocessing. The performance factors of the system we consider are *speed* and *signature expressiveness*. Speed can be measured by the CPU time used to generate the signatures from recorded traffic traces. Signature expressiveness reflects the appropriateness of the signature set found. The goal is to find the smallest set of signatures for the biggest coverage ratio for a specific application. There is an obvious tradeoff between these two performance metrics, but we found that our proposed system manages to perform better than prior solutions in terms of both speed and expressiveness. The improvement is so significant that this approach may open new use cases in traffic classification e.g., online per-user signature generation.

The methods for motif finding and sequence alignment are algorithms widely used in bioinformatics for similar purposes and a well-established tool set and literature are available to rely on. However, the application of these algorithms for networking purposes is far from trivial due to several reasons, such as the different distribution and number of symbols. (In bioinformatics there are 4 symbols in DNA, 5 in RNA and 19 in aminoacid sequences, while in the networking case a 1-byte representation of network traffic streams induces 256 different symbols.)

Since motif finding is a complex, time consuming procedure, we introduce a preprocessing step to remove parts of the input which appear only once or a few times. Preprocessing comprises of two steps. The first step is a hash algorithm based on the Rabin-Karp fingerprinting technique to filter substrings which occur only once. The second step is a prefix tree construction algorithm to collect substrings that occur frequently. Preprocessing can reduce the running time to about 3-16% of the original processing time.

We also introduce a postprocessing step in order to increase signature expressiveness by decreasing the overlapping coverage on the flow set. It is composed of three steps. In the first step the candidate signature set is refined by removing those signatures which give false positive results by cross-checking the candidate signatures with other applications. In the second step further information is collected about the positions of signatures in specific byte streams of flows. In the third step the minimal signature set with the maximal flow hit is determined. The postprocessing results show significant improvements in signature effectiveness, i.e. the size of the resulting signature set is decreased 5 times or even more.

In addition for proposing a system for the automatic generation of regexp signatures, we also propose to use Approximate String Matching (ASM) for actual DPI as an alternative to the common DPI techniques based on regular expressions. The proposed system results in high signature expressiveness.

The main contribution of the paper is as follows:

- a general framework for automatic signature generation;
- various adaptations of the framework to achieve different performance purposes; and
- a DPI system using Approximate String Matching with motifs as signatures.

This paper is organized as follows. Section 2 overviews the related work. The elements of our framework are introduced in Section 3. Preprocessing and resulting speedup is discussed in Section 4. Our postprocessing steps to select the best performing signatures are explained and evaluated in Section 5. We discuss using Approximate String Matching for DPI in Section 6. Finally, the paper is concluded in Section 7. The methods considered, their input and output are summarized in Figure 1.

2. RELATED WORK

Three types of protocol signature generation methods can be found in the literature: a) worm signature generation e.g., [18, 8, 10, 16], b) spam rule generation [2] and c) application signature generation [12, 14, 17, 26]. Authors of [26] presented AutoSig which extracts multiple common substring sequences from sample flows as application signature. First, all possible common substrings in an application protocol are extracted and then a substring tree is constructed to generate the final signature of the application. Being one of the latest articles in this topic we used AutoSig as a reference to measure the performance of our proposed system.

Topics in bioinformatics relevant to our problem are exact string matching, global pair-wise sequence alignments, local pair-wise sequence alignments, multiple sequence alignments and sequence motif finding [5].

The adaptation of bioinformatics algorithms for network protocol analysis has recently been found extremely useful. The primary goal of bioinformatics is to increase our understanding of biological processes. One can observe a similarity in the problems of bioinformatics compared to protocol analysis. As an example, in bioinformatics the purpose is to identify genes that produce proteins, while in protocol analysis the task is to identify the location and purpose of fields in the packets. This similarity makes it possible to investigate the application of bioinformatics algorithms in protocol analysis, however, the differences in the problems make this application a challenge. As an example, in [6] the authors use bioinformatics algorithms to determine fields in protocol packets. These authors propose a global sequence alignment based on the Needleman-Wunsch algorithm [15] with encouraging results.

Takeda proposes to apply bioinformatics algorithms for network intrusion detection in [22]. The method based on two algorithms. The Smith-Waterman algorithm [20] is applied to captured network traffic to locate patterns similar to known intrusion traffic. The Needleman-Wunsch algorithm is used to measure the similarity of the result to the known intrusion patterns. Coull et al. also addresses the intrusion detection by a bioinformatics approach [11]. Their method is a variation of the Smith-Waterman algorithm and using a novel scoring scheme to construct a semi-global alignment.

Tang et al. present a bioinformatics approach to generate accurate exploit-based signatures for polymorphic worms [23]. The core of the method is multiple sequence alignment which is used to identify invariant bytes from a set of polymorphic worm samples. The proposed pairwise sequence algorithm is also an improvement of the Needleman-Wunsch algorithm. The method is powerful to accurately analyze the intrinsic

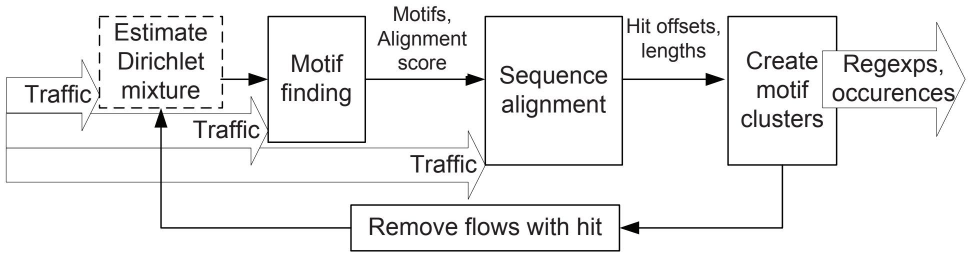

Figure 2: The regular expression construction process

similarities of worm samples.

The main difference of our approach proposed in this paper compared to the ones in related work is that we directly apply algorithms from the field of bioinformatics for motif finding. To our knowledge this is a novel approach and has not been published before. For further detailed discussion on related work and the application of bioinformatics approach to our purpose see our technical report [4].

3. REGULAR EXPRESSION CONSTRUCTION (M+R)

To construct regular expressions from the network traffic we propose a system applying motif finding and sequence alignment methods.

3.1 Proposed architecture

The input of the system is collected network traffic: either an application-aware active measurement or the capture of the traffic of an aggregating measurement point. In case the input traffic is classified according to the protocols, the generated application signatures can be associated with applications. Before feeding the input to our system, the TCP flows are reconstructed from the actual packet trace using `tcpflow` [9]. Since signatures are expected to be at the beginning of the flows only the first $10 - 100$ packets worth of data is considered from each flow.

In the following paragraph we show how we applied two bioinformatics algorithms – motif finding and multiple sequence alignment –, for the automatic application signature generation problem. A *motif* is a possibly gapped sequence of key positions, which is a re-occurring semi-deterministic sequence pattern found in multiple sequences generated by the same source. Key positions hold symbols (sequence elements) that are important for the motif's function. We used the `glam2` software package [3] developed to find motifs in biological sequences. The main innovation of `glam2` is that it allows insertions and deletions in motifs: it essentially implements a generalization of Gibbs Sampling technique[1] [24] to allow insertions and deletions in a fully general fashion. The tool takes the distribution of the symbol appearances as input. This distribution should incorporate prior knowledge of the functional similarities between symbols. To accomplish this, a Dirichlet mixture distribution is used which is a weighted sum of Dirichlet distributions [19].

[1]A special case of the Metropolis-Hastings algorithm, and thus an example of a Markov chain Monte Carlo algorithm. Sampling from probability distributions based on constructing a Markov chain that has the desired distribution as its equilibrium distribution: the state of the chain after a large number of steps is then used as a sample from the desired distribution.

The input flows are first fed to the Dirichlet mixture estimation algorithm. The output of this step is a Dirichlet mixture. Then, the input flows are fed to the motif finding algorithm as multiple sequences together with the Dirichlet mixture. The output of the motif finding algorithm is a set of motifs with alignment scores, which are the sum of the score of each appearance of the given motif (how well the motif fits the concrete character sequence it matches). In this step we consider only the motif with the best alignment score. To find the flows in which a hit occurred with the best motif we apply sequence alignment on the input flows. The output of sequence alignment is a list of flow ids, starting and ending positions of the match in the decreasing order of the matching scores. As we would like to get signatures in the form of regular expression, we collect all the appearances of the best motif in the original flows by saving the substrings in the positions indicated by the sequence alignment process. The byte values on the same positions with multiple occurrences are collected and a regular expression is created by putting an OR operator between them[2].

Applications typically have several protocol messages. In an extreme case one particular motif could describe them all, but the total score would be lower comparing to the case when the protocol messages are clustered and several motifs are defined for the message clusters. The creation of motif clusters can be done by defining the clusters based on the alignment scores. Those flows are considered together which score at least 80% of the maximum value in the list. These flows are separated from the original set of flows and the whole regular expression construction process is started over until no more flows left or only less than 10% of the original flows can be removed from the original set. To follow the steps of the signature generation algorithm, see Figure 2.

3.2 Performance evaluation

In order to evaluate the algorithm in different use cases, we investigated the following *metrics*:

- Number of motifs generated by the process;
- Flow coverage ratio: the ratio of flows covered by the yielded clusters;

[2]The same method is used in MEME-suite [7] for motif to regular expression conversion

	# of Dirichlet components	without preprocessing		with preprocessing but without common substring extraction		with full preprocessing	
		bit/symbol					
		4	8	4	8	4	8
Average number of motifs	0	3.92	3.50	1.65	0.91	1.68	0.84
	1	6.17	4.92	3.17	2.00	3.00	2.33
	10	9.17	8.67	7.06	5.37	7.61	6.85
Flow coverage ratio	0	47%	45%	24%	17%	23%	15%
	1	61%	49%	44%	34%	37%	31%
	10	83%	74%	72%	55%	63%	47%
CPU occupancy period [sec]	0	884	889	74	90	22	57
	1	972	1261	113	252	36	177
	10	1022	2609	174	594	76	423

Figure 3: The average number of motifs, the flow coverage ratio and the CPU occupancy period of 1000 Gnutella flows in the function of bit/symbol and number of Dirichlet components

- CPU time of the process.

Our aim is to find the alignment(s) with maximum score and derive a well-fitting motif. The various *parameters* on which the outcomes highly depend are[3]:

- Number of bits describing a symbol: aggregating bits may impose loss of information (e.g., in case of insertion, deletion); length of sequences and motifs decreases; number of symbols increases (varying alphabet size, e.g., 4 bits/symbol induces 16 symbols total). Modifications on applied tools are required in order to allow for general size of alphabets.
- Number of Dirichlet components: many components slow down motif finding, few components lower quality of *a priori* information. The decision to use a given number of components is somewhat arbitrary. As in any statistical model, a balance must be struck between the complexity of the model and the data available to estimate the parameters of the model. A mixture with too few components will have a limited ability to represent different contexts for the symbols. On the other hand, there may not be sufficient data to estimate precisely the parameters of the mixture if it has too many components.

For evaluation purposes we used active measurements with per packet information about the generating application [21]. The traces were divided into training and testing data sets each containing 1000 flows of the specific application. Note that motif finding contains random initialization sequences regarding e.g., the starting positions of the motif candidates. In order to filter out its effects we repeated every measurement 100 times.

3.2.1 *Effects of parameter settings on the motif finding algorithm*

[3]The investigation of other parameters, e.g., the minimum number of sequences in the alignment or the deletion and insertion preferences could be the target for future work.

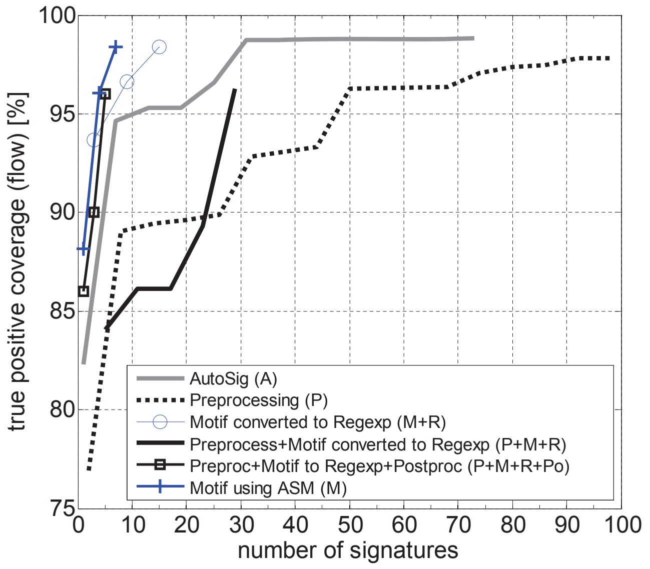

Figure 4: The true positive coverage per flow in the function of average number of signatures of the examined methods

In Figure 3 ("without preprocessing" column, "Average number of motifs" row) the average number of motifs generated for 1000 Gnutella flows shows that increasing the number of Dirichlet components, the number of motifs also increases. The consequence of this is that the flow coverage ratio and the overall CPU occupancy period also increase. In the other dimension, if the bit/symbol parameter is decreased from the intuitively set 8 bit/symbol to 4 bit/symbol, the number of found motifs also increases resulting in higher coverage ratio. It is interesting to note that the CPU occupancy decreases, probably due to the lower number of symbols. We tried to further decrease the bit/symbol parameter to 2 and 1, but in these cases the CPU occupancy increased with approximately 2-3 orders of magnitudes, therefore we considered these parameter sets practically inapplicable. The conclusion is that the motif finding algorithm is more sensitive to the length of the input sequences than the number of symbols.

3.2.2 *Performance comparison to state-of-the-art tool*

To compare the performance of the motif finding method for regular expression creation (M+R in Figure 4) with a state-of-the-art tool, we used [1], which is an implementation of AutoSig [26] with a slight speed-up. The matching algorithm for DPI is the conventional Deterministic Finite-state Automata (DFA) method.

The true positive (TP) metric is calculated as follows. The motifs are constructed per application, thus the training data of the motif finding algorithm contained only the flows of one specific application and later the found motifs are evaluated on the testing data set of that specific application which is a different set of flows than the training data. The resulting true positive ratios were averaged over the tested 11 applications (see Table 3). The false positive (FP) metric is calculated in a similar methodology but the calculation was different in the testing phase when the found motifs of

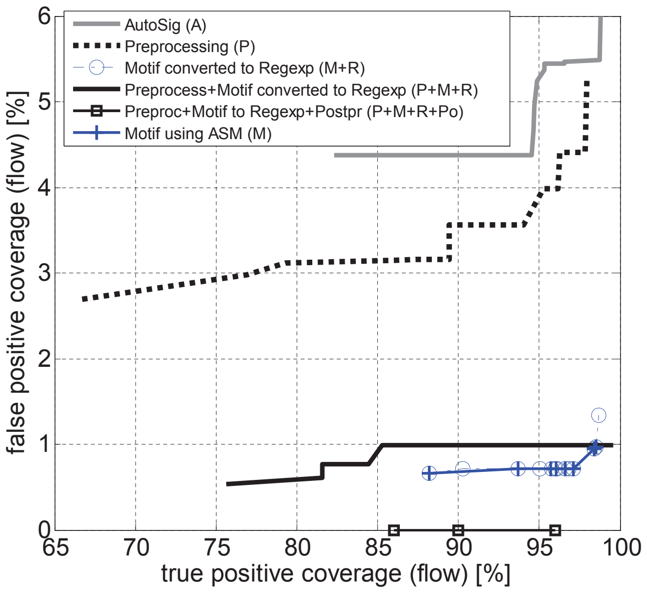

Figure 5: The false positive coverage of the examined methods in the function of true positive coverage

one specific application were tested on the testing flows of 10 other applications each.

Packet inspection complexity increases with the number of signatures to look for. Hence the compactness of the signature set describing an application (i.e., the number of signature required) is an important metric. By tuning the parameters of the methods we can generate signature sets of varying size – of course with different coverage of application traffic. On Figure 4 we have depicted the TP coverage of the signature sets generated by the various methods as a function of the signature set size. E.g., the method generated 100 signatures for a specific application and we take the first $1, 2, \ldots 100$ which gives the highest flow hit number. It can be seen that the regular expressions created by the motif generation converge faster to a total coverage than the output of the AutoSig tool. The most straightforward explanation for this is that AutoSig creates string signatures, and the expressiveness of such a grammar is limited by definition.

An other metric of signature expressiveness is the FP ratio in the function of TP ratio (M+R in Figure 5). If the FP ratio is high it means that the particular method finds mainly short signatures which can be found in other applications as well. Note that the ideal method would result in a dot at the bottom right corner of the figure with TP=100 and FP=0 values. AutoSig converges to 100% TP coverage with much higher FP coverage comparing to the M+R case. Regarding the average number of generated signatures the M+R method creates only 1/4 of the AutoSig signatures. This means a higher expressiveness of the constructed regular expressions. An other big difference is that the motif finding is 8 times faster than the AutoSig method (see Table 1).

	A	P	M+R	P+ M+R	P+M+ R+Po	M
Speed [flow/sec]	0.02	12.76	0.16	3.38	2.72	0.16
Avg. sig#	51.43	171.23	13.62	29.3	12.41	9.17

Table 1: The speed and average number of generated signatures of the methods

4. SPEEDUP WITH PREPROCESSING (P,P+M+R)

The above presented process may take hours even in a limited set of flows, thus we made efforts to speedup the process with a two step preprocessing phase. The preprocessing steps can be seen in Figure 6.

4.1 Speedup with fingerprinting

The first preprocessing phase applies a fast, memory efficient technique to significantly reduce the input size of the raw traffic by filtering substrings that occurred only once in the raw traffic. A possible way to do this is to create hashes from the content of a sliding window. The size of the hash table can be estimated and limited, so to control memory consumption. Then by flagging each hash value seen, we can roughly determine if a certain substring has been seen or not. In order to correctly detect substrings shorter than the window size (W_{len}) there has to be a separate hash table for all string lengths below W_{len}. The hash algorithm we used in this step is the Rabin-Karp fingerprinting method in a similar way as in the Earlybird paper [8].

To *compare the results of preprocessing to the raw traffic input case* we aggregated the total number of motifs, flow coverage and CPU occupancy time for all examined applications and compared them to the original case when the input of the motif finding was the raw traffic. In this way we can obtain information which extent the preprocessing phase affected the original traffic. Considering the flow coverage in the case of 4 bit/symbol and 10 Dirichlet components, it provides approx. 90% of the original coverage (see Figure 3 "with preprocessing but without common substring extraction" column, "flow coverage ratio" row). The required overall CPU time was only approx. 8-23% comparing to the original case (see Figure 3 "CPU occupancy period" row).

4.2 Speedup with prefix tree construction

The first preprocessing phase passes a substring to the second proposed step of the preprocessing phase only if it has already been seen more than once. The output of the first phase may contain longer substrings divided into shifted smaller substrings occurring multiple times in the output. Therefore we introduced a second preprocessing step to collect the same pre- and postfixes into the longest common substring. This way the input to the motif finding algorithm is further compressed.

The first step is to extract common substrings from the input streams. This is basically done running a fixed length sliding window (of length W_{len}) over the input and inserting all window content into a tree, counting the times each string has been inserted. Each node in the tree represents a substring, which is not longer than W_{len}. By summing

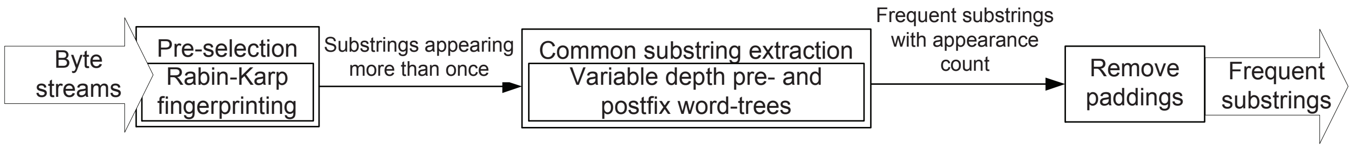
Byte streams
Pre-selection
Rabin-Karp fingerprinting
Substrings appearing more than once
Common substring extraction
Variable depth pre- and postfix word-trees
Frequent substrings with appearance count
Remove paddings
Frequent substrings

Figure 6: The preprocessing phases

the counters on the leafs of each sub-tree below a node, we can see how often the prefix represented by the node occurred in the input stream. This allows us to generate a list of substrings that occurred more than O_{min} times. When one of two substrings is a prefix of the other, we take only the longer one, except if the shorter one occurred at least by O_{min} times more than the longer one. For example, if "abcde" occurred 10 times and "abc" 30 times, we can deduct that 10 out of the 30 occurrences of "abc" were as part of "abcde". So if O_{min} is, e.g., 15, we will print "abc", too, with 20 occurrences. The resulting substrings are then checked in the reverse direction once more to eliminate those which are postfixes of another string present.

The above preprocessing construction algorithm can be run in a second pass on the input stream to detect common substrings longer than W_{len}. In this case we consider only those window contents which are preceded in the input by one of the substrings of maximum length (W_{len}) resulting from the first pass. If we find many occurrences of a such substring (always following the same W_{len}-length substring from the first pass), we can concatenate this to the substring from the first pass. This can then be repeated in multiple passes to detect even longer common substrings. The result of the whole tree operation is a list of common substrings with an occurrence count.

The bottleneck in the prefix tree construction operation is memory consumption in the first pass. Thus W_{len} has to be chosen in the function of the available memory. Many of the window contents will occur only once, yet they are all inserted into the tree. This limits the length of the window (W_{len}) and makes the whole process longer.

The output of the prefix tree is substring candidates with occurrence values. Motif finding is still needed as there are several examples in practice where e.g., the middle of a signature there is a sequence number and takes all the possible 256 values of a byte many times (over the minimum occurrence threshold). These cases can not be handled with the prefix tree. Feeding the substring candidates to the motif finding tool causes the loss of the occurrence information. Furthermore, a specific substring with high number of occurrences but with few substring variants is not found by the motif finding algorithm, thus these signatures should be added to motif clusters later. For example if "abc" occurred 100 times, "efxg", "efyg" and "efzg" occurred 10 times each, than the motif finding algorithm in this step would find with the last three, as a motif ("ef.g") can be found for them and does not consider the first one at all.

Comparing the results of preprocessing to the raw traffic input case, we found that the 4 bit/symbol with 10 Dirichlet component case provides approx. 76% of the original scores, being the closest to it (see Figure 3 "with full preprocessing" column, "flow coverage ratio" row). The required overall CPU time is only approx. 3-16% comparing to the original case, so further gain is achieved comparing to the first preprocessing phase (see Figure 3 "CPU occupancy period" row).

4.3 Remove padding

The output of the two preprocessing phases usually contains signature candidates with long padding, e.g., "00" and "ff" runs. It also frequently occurs that some optional fields are typically unused or unset in a protocol, or reserved for later usage thus resulting in long zero runs. The motif finding algorithm can not judge which zero runs are part of a signature or are only padding. We introduced a preprocessing step to remove these zero runs: the third phase of preprocessing on the signatures skips all the forthcoming zeros in case of 2 zero bytes. At the following non-zero byte, the method starts to collect a new signature, thus the original signatures are *split* by the double zero bytes. The same is performed for the "ff ff" bytes.

4.4 Performance evaluation

4.4.1 Preprocessing as a method on its own

We compared the performance of the preprocessing phase (P in Figures 4 and 5) with a state-of-the-art tool AutoSig [26]. The signatures of preprocessing converge to total coverage the slowest in the function of the number of signatures in Figure 4. Thus it needs the most signatures for certain TP coverage among all the methods. It has the highest average number of generated signatures. On the other hand, the FP coverage does not jump up to high values converging to the total TP coverage in Figure 5. It is 3 magnitudes faster than AutoSig and 2 magnitudes faster than motif finding on the raw traffic (P in Table 1).

4.4.2 Preprocessing with motif finding

Considering the preprocessing and motif finding methods together (P+M+R in Figure 4) it can be seen that it covers better with the same number of signatures than the output of the preprocessing (P) on its own but remains under AutoSig (A). Regarding the FP coverage (P+M+R in Figure 5) it has similar characteristics to M+R. Note that in this case the joint algorithm is 2 magnitudes faster than the AutoSig method (P+M+R in Table 1).

5. IMPROVING SIGNATURE EXPRESSIVENESS WITH POSTPROCESSING (P+M+R+PO)

The signature candidates yielded by motif finding are frequently occurring signatures in the given traffic. To further refine and restrict the signatures to the most valuable ones, we applied several postprocessing phases (see Figure 7).

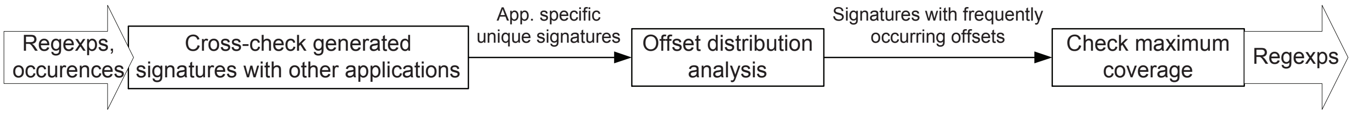
Regexps, occurences
Cross-check generated signatures with other applications
App. specific unique signatures
Offset distribution analysis
Signatures with frequently occurring offsets
Check maximum coverage
Regexps

Figure 7: The postprocessing phases

5.1 Proposed stages

The *first stage* in the post processing phase is the cross-check of the resulting signature candidates with other applications. Those signatures should be removed which can give false positive results.

The *second stage* is gathering additional information about the positions of signatures in specific byte streams of flows or packets. This step receives the signatures and the flow list as input and provides the following information per signature:

- the number of occurrences the given signature occurred at a specific offset considering all the flows
- the total number of matches of the specific signature (considering multiple times a multiple match per flow)
- the number of matches of the specific signature in different flows
- the number of different users with hits

The resulting signature set has usually overlapping coverage on the flow set, meaning that for one given flow there are several signatures which occurred. This overlap is non-optimal for the DPI engine as it has to check several signatures for the same hit ratio. In the *third stage* the goal is to select the minimal signature set which gives maximal flow, volume or user coverage. The problem is called the weighted maximum coverage problem [25] and considered to be NP-hard. A global optimum can be reached only by brute-force method comparing the coverage of every possible signature set.

The problem can be formulated in the following way. Lets consider the example flow and signature set in Table 2, where each row represents a flow, each column represents a signature and an 'X' is placed in a cell if the specific signature matches to the specific flow. The first column is an id of the flow, while the second column is the weight of the flow (W_i). It can be tuned e.g., with the byte volume that some of the flows are more important than the others. The third column is the user id of the flow generating terminal (U_i). The last column is the logical connection of the specific signatures for the specific flow. The elements of the signature set (S) are binary variables ($s_1, s_2, s_3 = \{0, 1\}$).

Several optimization problems can be formulated:

- Optimize on flow number coverage: The problem is to determine a $\sum_{i=1}^{flow\#} C_i$ is maximum while $\sum_{i=1}^{flow\#} S_i$ is minimum.
- Optimize on byte volume coverage: The problem is to determine a $\sum_{i=1}^{flow\#} C_i{*}W_i$ is maximum while $\sum_{i=1}^{flow\#} S_i$ is minimum.

flow id (i)	weight (W_i)	user id (U_i)	Sign (s_1)	s_2	s_3	Coverage (C_i)
1	10	U_1 = 192.168.1.1	X		X	$C_1 = s_1 \vee s_3$
2	15	U_2 = 192.168.1.1			X	$C_2 = s_3$
3	20	U_3 = 192.168.1.2		X		$C_3 = s_2$

Table 2: Flow coverage of signatures

- Optimize on user coverage: The problem is to determine a $\sum_{i=1}^{flow\#} C_i \wedge U_i$ and $\sum_{i=1}^{flow\#} U_i$ are maximum while $\sum_{i=1}^{flow\#} S_i$ is minimum.

Note that the obtained signatures for the different optimization cases are likely to differ. For instance regarding a P2P application, the signaling (e.g., peer search, file search) flows will be dominating the dataset, thus the optimization on flow number coverage will suggest these signatures. After a successful peer search, the data transfer flows will dominate the dataset in volume. Thus signatures referring to data transport will be suggested by the optimization on byte volume coverage. Few heavy users can dominate datasets thus both the above optimizations may suggest signatures for the specific user e.g., its user id or preferred music performers. Optimization for user coverage can overcome this issue and can provide non-user specific signatures.

We used constraint logic programming [13] to efficiently search for the optimal signature set. Constraint logic programming over finite domains makes it possible to narrow the search space as much as possible. When we run out of state space narrowing ideas at any time during the program execution, labeling of the variables can be started which is an exhaustive search over the possible values of the variables (brute-force search).

5.2 Performance evaluation

Considering the *preprocessing, motif finding and postprocessing methods together* (P+M+R+Po in Figure 4) it can be seen that it preserved the main attributes of the P+M+R line as the starting and ending TP coverage is similar to that case but the number of required signatures to achieve this is significantly lower. Regarding the FP coverage (P+M+R+Po in Figure 5) the FP coverage is 0% as a consequence of the working mechanism of the postprocessing phase. Note that postprocessing can only consider those applications which have signature candidates and traffic for cross-checking purposes. Further unprocessed applications may result in increased FP hits. A further consequence of postprocessing is the limited TP coverage. For instance such flows of an application which uses also common protocols for communication would provide signatures with FP hits thus removed from the signature set resulting later false negative hits on those flows by DPI. Regarding the overall speed of the combined methods (P+M+R+Po in Table 1), it remained about 2 magnitudes faster than the AutoSig method (P+M+R+Po in Table 1).

5.3 Example of found signatures

Table 3 shows a few examples of the resulting signatures on the tested applications. It is important to note that the number of motifs is far less than the number of different signatures found. A good example is in the case of World of Warcraft, where approx. 30 gaming servers with their ingame names, IPs and ports are collected out of the traces, but they are all covered with one motif.

Application	Regexp

BitTorrent	`BitTorrent ex`
	`1:rd2:id2`
	`node1:t8`

DirectConnect	`6/6.TTH:A`
	`ock ZLIG |`

Gnutella	`-Ultrapeer`
	`NUTELLA CON`
	`TLS@.UPC`
	`uting: 0.1`
	`DHTC.....DU`

MSN Messenger	`text value="aW..."`
	`@hotmail`
	`MIME-Version:1.0`
	`w" />  <imtext`

POP3	`+OK 0 me`
RTP	`[T|U][T|U]UUUUUUUU`
SSH	`aes128-cb`

World of Warcraft	`.\%.@.2mv/.P'T3y.}...m..`
	`rider.80.239.179.51:372`
	`Blade:80.239.179.39.372`
	`light:80.239.185.80.372`

eDonkey	`E.....].`
	`..{......p`

pplive	`!.B..!.B..!.B..!.B..!`
Spotify	`._.e..9âĂę`

Table 3: Examples of signatures on the tested applications

6. APPLICATION OF APPROXIMATE STRING MATCHING IN DPI (M)

The most accurate method to recognize protocols would be complete protocol parsing. As these techniques are very resource consuming, DPI is used which searches for characteristic byte signatures in the traffic. This technique is accepted to be the most accurate among the traffic classification techniques but it should be noted that this technique remains a heuristic. On the contrary, results are considered as a final verdict. If a match occurs, the traffic is classified to the signature of the application which generated the hit. All information related to the reliability of the hit is lost.

6.1 Proposed system

We propose to use approximate string matching (ASM) as a basis of DPI. The identification of unknown traffic is as simple as performing sequence alignments for the unknown traffic with the several motifs describing various applications. The resulting scores of the sequence alignment runs can be summed per application motifs and can be compared to each other. The underlying application of the group of motifs with the highest sum score is the most probable generating application of the unknown traffic.

One advantage of the proposed method is to make the DPI engines to be able to use such signature sets which would otherwise give false positive hits on their own. E.g., '@hotmail.com' for MSN is a good factor of the sum motif score (as MSN usernames are usually hotmail addresses), but not application specific on its own. As not necessarily every motif is specific for only one application but using the sum of the motif scores for one specific application make them reliable indicator for an application hint. It is also a straightforward advantage of the method when such motifs are the application descriptors which known to be changed deliberately e.g., the e-mail spam and other text-like characteristics protocols, such as 'VIAGRA' changes to 'V.I.A.G.R.A'. The motifs are even more robust for protocol version changes over time than regular expressions. E.g., new option fields in a protocol do not largely affect the motifs. It is also important to note that fewer motifs are enough to describe the same protocol compared to the required number of regular expressions.

6.2 Performance evaluation

Figure 4 (M) shows the case when the usual DFA is substituted with sequence alignment and motifs are used. It can be seen that it has the highest coverage in the function of the number of signatures. Regarding the FP coverage in Figure 5, it has similar characteristics to the M+R case and has the lowest among all methods. The motifs evaluated via ASM represent a theoretical maximum of the motif expressiveness. During the regular expression conversion and the DFA based evaluation in the M+R case, information lost occurs by definition.

Comparing the calculation complexity of the ASM with DFA the following can be found. The DFA has $O(n)$ complexity where n is the length of input string. The sequence alignment has $O(nm)$ complexity [5] where n is the length of the input string, m is the length of the motif. The difference is linear, thus the algorithm may be a proper candidate on e.g., post processing of such traffic which can not be identified with the common DPI techniques.

7. CONCLUSION

In this paper we present a general framework of an automatic application protocol signature generation for Deep Packet Inspection (DPI) techniques. The proposed framework utilizes algorithms from the field of bioinformatics. The framework also consists of preprocessing and postprocessing techniques to gain better performance. In the preprocessing phase we applied a Rabin-Karp fingerprinting based method to filter the once occurring substrings and a prefix tree construction method to summarize the substrings with common pre- and postfixes. We found that the motif finding system extended with the preprocessing phase can achieve high flow coverage ratio with low CPU occupancy period. We also introduced several postprocessing methods to select the best performing signatures of the candidates. We applied a cross-checking phase to filter out signatures with false positive hits for other applications, an offset distribution analysis phase and a maximum-coverage optimization phase with focus on either flow number, volume or user number.

We carried out a detailed performance analysis and systematically compared the quality of the signatures generated by the framework composed of different building boxes to a state-of-the-art tool. We found that our introduced method can result in better performance in terms of both speed and signature expressiveness. It gives about 5 times smaller signature sets in about 100 times shorter period of time than the state-of-the-art tool. Furthermore, our general framework can also be tuned by using different building boxes to optimize specifically for speed or signature expressiveness. Finally, we discussed a DPI system based on approximate string matching and our results showed that it is a viable alternative for the refinement of exact string matching algorithm outcomes.

8. REFERENCES

[1] A. F. Santos: Automatic Signature Generation, Diploma Thesis, 2009. `http://www.cin.ufpe.br/~tg/2009-1/afs5.pdf`.

[2] Eric Conrad: Detecting Spam with Genetic Regular Expressions. `http://www.sans.org/reading_room/whitepapers/email/detecting_spam_with_genetic_regular_expressions_2006`.

[3] glam2 – a motif finding tool. `http://bioinformatics.org.au/glam2/`.

[4] L. Toka, G. Szabó, Z. Turányi: Discovering Motifs in Application Flows, Tech Report, 2010. `http://www.crysys.hu/~szabog/TR2010.pdf`.

[5] Lectures in bioinformatics. `http://www.cs.otago.ac.nz/cosc348/lectures.html`.

[6] M. A. Beddoe: Network Protocol Analysis using Bioinformatics Algorithms, 2009. `www.4tphi.net/~awalters/PI/pi.pdf`.

[7] MEME-suite [2010]. `http://meme.nbcr.net/meme4_3_0/doc/examples/meme_example_output_files/meme.html`.

[8] S. Singh, C. Estan, G. Varghese, and S. Savage: The Earlybird System for the Real-time Detection of Unknown Worms, UCSD, Department of Computer Science, Technical Report CS2003-0761,. `http://www.cs.unc.edu/~jeffay/courses/nidsS05/signatures/savage-earlybird03.pdf`.

[9] tcpflow. `http://sourceforge.net/projects/tcpflow/`.

[10] H. ah Kim. Autograph: Toward Automated, Distributed Worm Signature Detection. In *In Proceedings of the 13th Usenix Security Symposium*, pages 271–286, 2004.

[11] S. Coull, J. Branch, B. Szymanski, and E. Breimer. Intrusion detection: A bioinformatics approach. In *Proceedings of the 19th Annual Computer Security Applications Conference*, pages 24–33, 2001.

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

[13] M. Hanus. Programming with constraints: An introduction by kim marriott and peter j. stuckey, mit press, 1998. *J. Funct. Program.*, 11(2):253–262, 2001.

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

[15] S. B. Needleman and C. D. Wunsch. A general method applicable to the search for similarities in the amino acid sequence of two proteins. *Journal of Molecular Biology*, 48(3):443–453, March 1970.

[16] J. Newsome, B. Karp, and D. Song. Polygraph: Automatically generating signatures for polymorphic worms. In *Proceedings of the 2005 IEEE Symposium on Security and Privacy*, pages 226–241, Washington, DC, USA, 2005. IEEE Computer Society.

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

[18] W. Scheirer and M. Chuah. The Strength of Syntax Based Approaches to Dynamic Network Intrusion Detection. In *Information Sciences and Systems, 40th Annual Conference on Volume*, March 2006.

[19] K. Sjölander, K. Karplus, M. Brown, R. Hughey, A. Krogh, I. Mian, and D. Haussler. Dirichlet mixtures: a method for improved detection of weak but significant protein sequence homology. *Computer Applications in the Biosciences*, 12(4):327–345, 1996.

[20] T. F. Smith and M. S. Waterman. Identification of common molecular subsequences. *Journal of Molecular Biology*, 147(1):195–197, 1981.

[21] G. Szabó, D. Orincsay, I. Szabó, and S. Malomsoky. On the validation of traffic classification algorithms. In *Proc. PAM*, Cleveland, Ohio, USA, April 2008.

[22] K. Takeda. The application of bioinformatics to network intrusion detection. In *Security Technology, 2005. CCST '05. 39th Annual 2005 International Carnahan Conference on*, pages 130 – 132, 11-14 2005.

[23] Y. Tang, B. Xiao, and X. Lu. Using a bioinformatics approach to generate accurate exploit-based signatures for polymorphic worms. *Computers & Security*, 28(8):827–842, 2009.

[24] G. Thijs, K. Marchal, M. Lescot, S. Rombauts, B. De Moor, P. Rouzé, and Y. Moreau. A gibbs sampling method to detect over-represented motifs in the upstream regions of co-expressed genes. In *RECOMB '01: Proceedings of the fifth annual international conference on Computational biology*, pages 305–312, New York, NY, USA, 2001. ACM.

[25] V. V. Vazirani. *Approximation algorithms.* Springer-Verlag New York, Inc., New York, NY, USA, 2001.

[26] M. Ye, K. Xu, J. Wu, and H. Po. Autosig-automatically generating signatures for applications. In *CIT (2)*, pages 104–109. IEEE Computer Society, 2009.