2016 IEEE 41st Conference on Local Computer Networks Workshops

Operating System Classification Performance of TCP/IP Protocol Headers

Ahmet Aksoy
Computer Science and Engineering Department
University of Nevada, Reno
aksoy@nevada.unr.edu

Mehmet Hadi Gunes
Computer Science and Engineering Department
University of Nevada, Reno
mgunes@unr.edu

***Abstract*—Identification of operating systems in a local network is an issue for both network management and security. Network practitioners rely on some classifier tools, but those tools' rules are generated by an expert. Hence, existing approaches need to be manually updated for each new operating system. In this paper, we analyze the TCP/IP packet headers to automate operating system classification. To this end, we measure the classification performance of each protocol, and determine the unique features between operating systems. We utilize a genetic algorithm to determine the relevant packet header features. Then, we use several machine learning algorithms to generate set of rules that can differentiate operating systems. Overall, with IP, ICMP, TCP, UDP, HTTP, DNS, SSL, SSH, and FTP, protocol header information, on average, operating system classification can be performed at a rate of 68.0%, 51.6%, 98.4%, 71.1%, 78.7%, 29.2%, 25.0%, 22.5%, and 14.0%, respectively. In general, feature extraction with genetic algorithm further improves the results, e.g. to an average of 99.1% for TCP.**

***Index Terms*—OS Fingerprinting; Machine Learning; Genetic Algorithm**

I. Introduction

Especially with the Internet of Things, everything is getting connected to the Internet. With a plateau of devices attached to a network, management of the local networks is becoming more challenging. As various operating systems run these devices, identifying and patching vulnerable systems is crucial. Network managers adopt multiple security mechanisms to protect the network from malicious activities. An important step in securing a network is to be aware of the devices that are attached to the network. It is important to detect devices that might be using old or insecure versions of operating systems due to their vulnerabilities to remote attacks [14].

Operating system fingerprinting is the process of remotely detecting the operating system of a target device. There are two methods for performing operating system classification: active and passive [8]. Active fingerprinting, actively probes target devices and analyzes their responses. It is possible for certain systems to respond in an unusual way upon certain requests. Such scenarios can allow for active fingerprinting tools to narrow down possibilities or even directly determine the operating system of such systems. However, firewalls could block probes and active fingerprinting might lead to limited or no knowledge of target host. Passive fingerprinting, passively sniffs packets from target devices and analyzes the packet header information. As passive fingerprinting merely depends on the information extracted from regular TCP/IP packet headers, it is possible to perform operating system classification even when a firewall exists. On the other hand, passive fingerprinting might not be as accurate as active fingerprinting as it could not observe distinct features. Passive fingerprinting techniques can take advantage of less than one third of the features that active fingerprinting tools such as Nmap offers [14].

It can become very impractical for system administrators to manage and monitor huge amount of hosts which might be at different locations [13]. Therefore, operating system fingerprinting tools and techniques can ease network management and security [19]. These tools might also be useful for collecting statistical data on the operating systems that hosts within a network use. Operating system fingerprinting, however, can also be used for malicious purposes. It could come in handy for intruders to know devices and their vulnerabilities in a target network. When an intruder realizes presence of unpatched or outdated operating systems, they could easily deploy known malware to compromise those systems.

While current OS classification systems are expert based, automated detection and configuration of networked systems are valuable [2]. To this end machine learning approaches can provide such automated system configuration with minimal or no expert input [12].

In this paper, we analyze the performance of TCP/IP headers in classifying operating systems with machine learning. We perform single-packet operating system classification on packets originating from host devices. We compare the classification results for several protocols at layer 3 (i.e., IP and ICMP), layer 4 (i.e., TCP and UDP), and layer 5 (i.e., HTTP, DNS, SSL, SSH, and FTP) using multiple machine learning algorithms in order to determine the contribution they have in classifying OSes.

We tested the classification both with every non-null feature extracted from the protocol and with features that were selected by the genetic algorithm. Genetic algorithm allows the ability to perform operating system classification of packets with much less computational overhead without sacrificing the classification performance considerably. After selecting features that are helpful for classifying the packet, we use different machine learning algorithms to detect the operating system of the host devices. With the help of the genetic algorithm, we detected the features that contribute most to the classification of operating systems at hand. Using genetic algorithm and machine learning, we identify TCP/IP header features that can guide operating system classification.

Our results in general are consistent with expert system based operating system fingerprinting tools such as p0f, ettercap, and siphon [14]. While current tools that depend on specific packet types such as SYN, ACK and SYN-ACK, our classifier is able to detect operating systems of individual packets regardless of the packet type. In this measurement study, we performed operating system classification with all types of packet observed from host devices.

The rest of the paper is organized as follows: Section II describes the related work. Section III explains the methodology for machine learning of TCP/IP protocol headers. Section IV shows the performance results of different protocols. Section V concludes the paper.

II. Related Work

Operating system fingerprinting techniques are often classified as active and passive approaches [8]. In active fingerprinting approach, the target device is directly probed and depending on its response the operating system of the target device can be detected. In passive fingerprinting approach, operating system detection can only be performed by analyzing sniffed packets from the target device. There are also hybrid approaches that try to overcome the limitations of active and passive approaches by combining both. For instance, Sinfp uses signatures acquired from active fingerprinting to perform passive fingerprinting [3].

A. Active fingerprinting

Veysset et al. performs temporal response analysis based operating system detection [20]. Their system employs an active fingerprinting approach to elicit the TCP SYN packet responses which is then compared to the known signatures to detect the operating system of the target system.

Similarly, Arkin performs active operating system classification by analyzing the ICMP replies from target systems [1].

SYSNSCAN tries to determine the distinguishing information among different TCP implementations of operating systems to be able to determine the operating system of a target system [19]. In addition to incorporating many of the existing techniques, SYNSCAN also depends on features such as congestion control algorithm, congestion window size, don't fragment bit, default MSS value, IP identification field, TTL value, etc.

Greenwald et al. derive effective probe communications for operating system detection by evaluating fingerprinting probes in order to reduce the amount of packets to be exchanged with the target system [9]. In active fingerprinting, multiple probes to a target system might be needed to deduce the operating system of the target device. It is possible that tools such as IDS at the target network can detect these probes and prevent responses to such probes. By minimizing the probes to a target system, the authors try to evade such mechanisms.

Additionally, machine learning techniques have been adopted for performing operating system classification [17]. Authors employ neural networks and statistical tools with DCE-RPC endpoints and Nmap [4] signatures to improve detection analysis. The proposed system consists of two modules where one of them performs Windows operating system classification using DCE-RPC and the other one performs Linux-based operating system classification using Nmap. Authors have relied on features such as ACK flag responses: S, S++, O; DF flag response (yes/no); response flag: ECN-Echo, URG, ACK, PSH, RST, SYN, FIN; Options field and window size.

Finally, Nmap [4], SinFP and XProbe2 are among some of the active fingerprinting tools that are publicly available.

B. Passive fingerprinting

Spitzner performs passive operating system classification on pre-determined signatures such as, TTL, window size, DF and TOS [18].

Lippmann et al. determines the accuracy of passive operating system classification based on TCP/IP packets along with evaluating open-source tools for operating system classification [14]. They also evaluate suitable classifier techniques to increase to the classification performance.

Beverly developed a Naive Bayesian classifier for passive fingerprinting [5]. The presented machine learning based approach is compared to rule-based inference tools such as p0f's signature database and HTTP UserAgent data in terms of its classification performance.

Chen et al. analyze and identify series of TCP/IP header features to examine the effectiveness of such features for their contribution to operating system classification including mobile devices [7]. Authors propose to utilize features such as the stability of the clock frequency, presence of TCP timestamp option, and the default set of TCP window size scale.

Mavrakis develop a machine learning based system based on TCP/IP headers and classifier based on decision-tree learning [15]. It is shown that the UserAgent data outperformed p0f's signature database. Since authors use p0f's signatures, the selected features are similar to that of p0f including MSS, WS, and iTTL. They also consider features such as options layout and IP version.

Different from operating system classification from TCP/IP traffic analysis, Chang perform operating system classification based on DNS logs [6]. Author used chi-squared test to extract features from DNS logs to distinguish different OSes and used the hamming distance for classification.

Finally, p0f, ettercap, SinFP, NetworkMiner, NetSleuth, PacketFence, PRADS and Satori are among some of the publicly available passive fingerprinting tools.

III. Methodology

In this paper, we measure the operating system classification performance of TCP/IP protocol headers using machine learning. We use the genetic algorithm feature selection technique for determining the relevant features from TCP/IP protocol headers. After determining the protocol features, we used machine learning algorithms to populate a set of rules using the full and selected set of features.

We initially collected the set of all features to be used for performing operating system classification. We then eliminated features which contained null values along with the ones that were machine learning incompatible such as cookie value in HTTP, domain name in DNS, etc. and the ones that were machine dependent such as MAC address in DHCP, user agent in HTTP, etc.

For the fitness function of the genetic algorithm, we employed the wrapper method of feature selection [11]. The fitness function uses the trainer itself to determine the performance of feature combinations generated by the genetic algorithm. Wethen determined TCP/IP protocol header features that led to the best operating system classification.

A. Data Initialization

We set up a local network consisting of 4 computers. Three of these computers contain instances of Fedora 23, Xubuntu 14.04, Windows 7 and Windows 8 and the fourth computer contains an instance of OSX El Capitan. We collected packets from multiple devices to remove any possible bias that might have existed while collecting packets from one instance of the operating systems. Every instance of operating systems on the machines that we collected packets from were freshly installed. We did not use VirtualMachine in order to generate as realistic scenarios as possible and used the Wireshark tool to collect packets.

To generate HTTP protocol packets; we visited the same websites on every operating system. These websites were; `http://www.google.com`, `http://www.yahoo.com`, `http://www.unr.edu` and `http://www.youtube.com`. We also collected approximately 20 minutes of YouTube video streaming. To generate FTP protocol packets, we connected to `ftp://ftp.godaddy.com` FTP server and uploaded files to the server. To generate ICMP packets, we used the traceroute application to connect to all 4 of the domain names mentioned earlier. To generate SSH packets, we initiated connections to a SSH server and transmitted files to the server. The remaining protocols such as; IP, TCP, UDP, DNS and SSL were observed among the collection of packets for the protocols mentioned above.

The number of packets collected for each protocol are provided in Table I. The number of packets among different protocols differ as different protocols have different popularity in real network flows. 5% of the dataset was dedicated to the training of genetic algorithm's fitness function and another 5% was dedicated to the testing of genetic algorithm. The remaining 90% of the packets were split into 5 parts to perform 5-fold cross-validation test. 4 of the 5 parts of the packets were used to train the system and the remaining one was used to test it. This process was performed for every combination of these parts of data and their average performances were recorded.

TABLE I
NUMBER OF PACKETS

Protocols	GA train	GA test	Train	Test	Total
IP	4,711	4,711	339,280	84,820	433,522
ICMP	39	39	2,900	725	3,703
TCP	5,583	5,583	402,100	100,525	513,791
UDP	711	711	51,240	12,810	65,472
HTTP	308	308	22,320	5,580	28,516
DNS	561	561	40,580	10,145	51,847
SSL	201	201	14,660	3,665	18,727
SSH	-	-	60	15	75
FTP	-	-	100	25	125

For packet collection, packets containing certain protocol were generated and collected within this local network. These protocols include; HTTP, DNS, SSL, FTP, SSH, ICMP, UDP, TCP and IP. For each protocol, feature selection and machine learning classification was performed for two sets. One of them is using every possible feature of the provided protocol and the other one is using the genetic algorithm selected features of this protocol. Also, for each test, features of the provided protocols were collected and those features with only null values have been eliminated since they do not contribute anything to the classification.

SSH and FTP protocol packets seemed to have many packets with similar headers and after removal of the duplicate packets there were not enough number of packets to perform genetic algorithm. Therefore, we did not perform genetic algorithm feature selection technique for SSH and FTP protocols. We calculated the classification performances of these protocols using all available features.

B. Feature Selection

In addition to testing operating system classification performance with every feature determined from the protocols, we tried to reduce the number of features to be considered. We use genetic algorithm to determine subset of features that would keep the classification performance high. Genetic algorithm is the process of searching and testing the performance of a solution among a space of solutions [16]. The idea is based on the biological mechanisms of natural selection and reproduction. Genetic algorithm uses an objective (or fitness) function to evaluate every solution it finds. This process is performed until a certain criteria is met. The criteria in our case was the generation of 15 consecutive solutions which are the same. The fitness function to evaluate the solutions is:

$$\begin{aligned} Fitness = \ & 0.80 \times Accuracy + \\ & 0.15 \times \left(1 - \frac{|SelectedFeatures| - 1}{|AllFeatures| - 1}\right) + \\ & 0.05 \times \left(1 - \frac{|SelectedRules| - 1}{|AllRules| - 1}\right) \end{aligned}$$

TABLE II
NUMBER OF SELECTED FEATURES

Protocol	All	J48	JRip	Ridor	PART	DT	RF	NB	MP
IP	23	5	6	4	6	4	9	3	5
ICMP	20	4	10	5	3	1	5	5	5
TCP	62	7	8	7	12	4	10	12	-
UDP	7	1	3	1	1	3	1	2	2
HTTP	17	3	5	1	7	1	2	3	7
DNS	34	1	8	1	5	5	12	3	2
SSL	31	1	10	4	4	3	2	7	11
SSH	18	-	-	-	-	-	-	-	-
FTP	4	-	-	-	-	-	-	-	-

where $Accuracy$ represents the classification performance with the provided machine learning algorithm.

Different weight values for accuracy, features and rules were tested. Since we would like to perform classification with as high classification performance as possible, we have set the weight for accuracy to 80%. We also wanted to use as few features as possible while trying to keep the performance high. Therefore, we set the weight for the number of features to be selected to 15%. Finally, we wanted to end up with as small number of rules or exemplars at the end as possible, so we set the weight for the number of rules to be extracted by the classifier to 5%.

C. Rule Extraction

After determining the features to be used, we analyzed different machine learning algorithms to perform operating system classification. We used the WEKA tool [10] for classification. We utilized a set of algorithms in the WEKA tool, namely; J48, JRip, Ridor, PART, DecisionTable, RandomForest, NaiveBayes, and MultilayerPerceptron.

IV. Experimental Results

In this section, we provide the classification performance of different layer 3 (i.e., IP and ICMP), layer 4 (i.e., TCP and UDP), and layer 5 (i.e., HTTP, DNS, SSL, SSH, and FTP) protocols. We demonstrate classification performance with all the extracted features from protocol header and with features selected by the genetic algorithm. The fitness function for the genetic algorithm tries to increase the classification performance while trying to decrease the number of features and number of rules. Table II shows the number of features that originally existed in our dataset along with the number of features that genetic algorithm selected for each protocol. As seen in the table, genetic algorithm was able to decrease the number of features immensely.

The use of genetic algorithm for selecting features had a very huge impact on the results that we generated. As presented in Table II, the number of features that algorithms used for classification of operating systems are much lower than the initial number of features. It can be expected for the classification performance to drop when the number of features are immensely reduced, however, the genetic algorithm is able to select the ones which are the most helpful in terms of differentiating the operating systems. As shown in following protocol evaluations, there are even cases where the classification performance increases after applying the genetic algorithm.

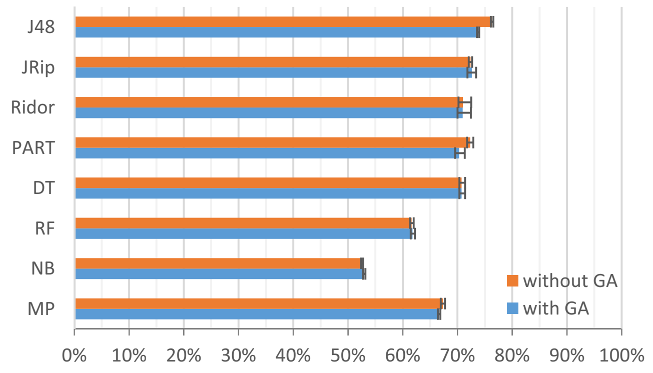

Fig. 1. IP Performance

A. IP protocol headers

IP protocol is among the ones that provide acceptable amount of uniqueness to the classification of operating systems as seen in Figure 1. While the bars present the average of 5 evaluations, error bars present the min and max performance of the algorithm. On average of all algorithms, we observe 68.0% performance when all features are considered, while performance becomes 67.5%, on average, when subset of features are utilized. At its best, it has 76.2% performance of classification with the J48 algorithm.

Table III presents IP protocol features that were selected by different machine learning algorithms. As seen in the table, among the features extracted from the IP protocol packets, the Checksum (checksum) feature seems to be the most preferred feature by all of the algorithms for determining the uniqueness of different operating systems. In general, the checksum contains distinguishing information about packets due to the fact that it contains a summary of protocol header.

TABLE III
SELECTED IP FEATURES

Features	J48	JRip	Ridor	PART	DT	RF	NB	MP	Σ
checksum	✓	✓	✓	✓	✓	✓	✓	✓	8
ttl		✓	✓	✓	✓	✓	✓	✓	7
id	✓	✓	✓	✓	✓	✓		✓	7
len	✓	✓	✓			✓		✓	5
proto	✓	✓		✓					3
opt.ra	✓							✓	2
dsfield.dscp				✓		✓			2
frag_offset				✓		✓			2
dsfield.ecn						✓			1
flags.df						✓			1
opt.type						✓			1
hdr_len							✓		1
opt.type.number					✓				1
checksum_bad		✓							1
Σ (features)	5	6	4	6	4	9	3	5	

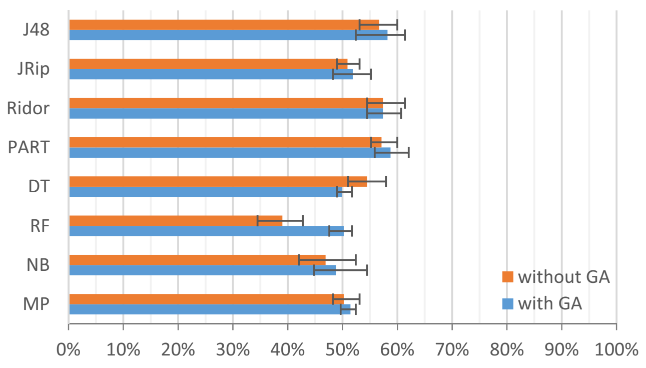

Fig. 2. ICMP Performance

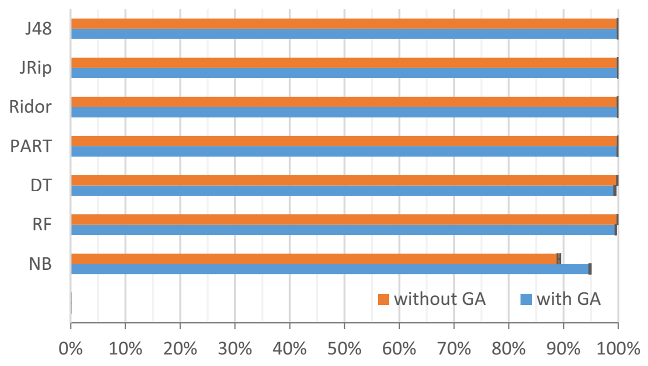

Fig. 3. TCP Performance

Along with checksum, the ID and TTL features are among the most selected features by the algorithms. IP ID is the identification number of a packet and it is often incremented for every packet sent from the operating system. As long as its values do not overlap for packets from multiple operating systems, it is expected for the feature selection algorithms to detect the uniqueness of this feature. One of the reasons for collecting packets from multiple devices was to eliminate such biases, but it is difficult to completely eliminate them.

TTL is used to prevent infinite loops on the Internet where every router decrements this value until it becomes 0 or reaches its intended destination. As the second most common differentiator, it seems operating systems prefer different initial values for this feature and hence TTL becomes useful to classify operating systems based on IP protocol header information.

The forth most important feature is the Total Length (len) feature, which contains the number of 32-bit words in the header. Also depending on the content's uniqueness in every packet, it is possible for this particular feature to be very helpful in determining which operating system the packet might have originated from.

Even though the number of features genetic algorithm extracted for each algorithm differs, Ridor seems to perform well with using just the 4 most popular features indicated earlier. It was even able to outperform the RandomForest algorithm using less than half of the features RandomForest selected.

According to [14], TTL and Total Length are among the features open-source tools use and we are able to observe their importance in our evaluations as well. However, another feature that was mentioned in this paper was the Don't Fragment bit, but it was only selected by a single algorithm in our experiements.

B. ICMP protocol headers

ICMP is a protocol that does not contribute much to the classification of operating systems as seen in Figure 2. On average of all algorithms, we observe 51.6% performance when all features are considered, while performance becomes 53.3%, on average, when subset of features are utilized. At its best, it has 58.8% performance of classification with the PART algorithm on selected features.

Similar to the IP protocol, the most preferred feature by the algorithms is checksum as seen in Table IV. However, the results for DecisionTable algorithm shows us that, a performance of 54.4% can be achieved with just the identification feature. The identifier feature for an ICMP packet provides unique classification, above average of all algorithms that have additional features.

C. TCP protocol headers

TCP is the most distinguishing protocol header to accurately classify operating systems as shown in Figure 3. Note that, MultilayerPerceptron algorithm could not complete its classification due to its considerably high complexity and the large number of features and number of TCP packets. Hence, it is omitted in TCP analysis. On average of all algorithms, we observe 98.4% performance when all features are considered, while performance becomes 99.1%, on average, when subset of features are utilized. At its best, it has 99.9% performance

TABLE IV
SELECTED ICMP FEATURES

Features	J48	JRip	Ridor	PART	DT	RF	NB	MP	Σ
checksum	✓	✓	✓	✓		✓	✓	✓	7
type		✓		✓		✓	✓	✓	5
mpls.s	✓	✓				✓		✓	4
checksum_bad		✓	✓					✓	3
ext.class	✓	✓							2
ext.ctype		✓	✓						2
ext.length		✓	✓						2
ext.res							✓	✓	2
ext.version				✓			✓		2
ident					✓	✓			2
ext		✓							1
ext.checksum		✓							1
mpls.label		✓							1
seq							✓		1
code						✓			1
mpls.ttl			✓						1
mpls.exp	✓								1
Σ (features)	4	10	5	3	1	5	5	5	

of classification with the PART, Ridor, Jrip, and J48 algorithms with both all features and selected features. Overall, we realize that TCP is a good differentiator of operating systems as it is a complex protocol with various features that are implemented differently by operating systems.

Although the number of features used by certain algorithms can reach up to 12 as seen in Table V, there are algorithms which were still able to classify at a very close performance rate with much less features. For example, the DecisionTable algorithm was able to perform classification at a rate of 99.4% using only 4 features which is just 0.5% less than the PART algorithm, which used 12 features. The four features extracted by the genetic algorithm using the DecisionTable algorithm are; source or destination Port (port), analysis.out_of_order, analysis.retransmission, and copy on fragmentation (options.type.copy).

According to [14], Window size (WS), TCP Max Segment Size Option* (MSS), TCP Window Scaling Option Flag* (WSO), TCP Selective Acknowledgments Options Flag* (SOK), TCP NOP Option Flag* (NOP) and TCP Timestamp Option Flag* (TS) are among the features open-source tools use. In our experiements, however, Window size was selected by only the PART algorithm and TCP Max Segment Size Option was not selected by any of the algorithms. As for the TCP Window Scaling Option Flag, similar to what the tools use, four of the algorithms extracted the (window_size_scalefactor) feature. As opposed to TCP Selective Acknowledgments Options Flag alone, our genetic algorithm implementation selected (options.sack.count), (options.sack_le), (options.sack_perm) and (options.sack_re) features. These features were also selected by single algorithms individually. Four of utilized machine learning algorithms selected the (options.timestamp.tsval) feature and this feature has an important impact on our results as well.

TABLE V
SELECTED TCP FEATURES

Features	J48	JRip	Ridor	PART	DT	RF	NB	Σ
stream	✓	✓	✓	✓			✓	5
options.timestamp.tsval	✓	✓	✓	✓				4
window_size_scalefactor	✓			✓		✓	✓	4
port		✓			✓	✓	✓	3
srcport	✓		✓	✓				3
analysis.duplicate_ack_frame				✓			✓	2
analysis.duplicate_ack_num		✓				✓		2
analysis.out_of_order					✓		✓	2
analysis.rto_frame				✓			✓	2
flags		✓		✓				2
flags.reset				✓		✓		2
option_kind		✓				✓		2
nxtseq		✓					✓	2
flags.push							✓	1
option_len							✓	1
options.wscale.multiplier							✓	1
options.wscale.shift							✓	1
pdu.size							✓	1
ack						✓		1
segment.count						✓		1
seq						✓		1
dstport						✓		1
flags.cwr						✓		1
options.type.copy					✓			1
analysis.retransmission					✓			1
window_size_value				✓				1
options.sack_perm				✓				1
analysis.zero_window				✓				1
analysis.bytes_in_flight				✓				1
options.timestamp.tsecr			✓					1
options.sack.count			✓					1
checksum_good			✓					1
analysis.duplicate_ack			✓					1
options.sack_le		✓						1
reassembled.length	✓							1
options.sack_re	✓							1
hdr_len	✓							1
Σ (features)	7	8	7	12	4	10	12	

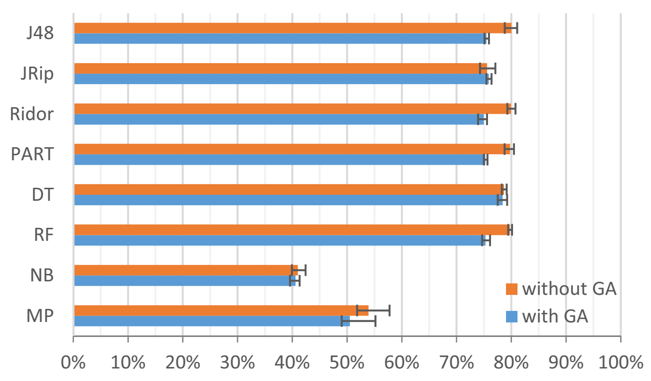

Fig. 4. UDP Performance

D. *UDP protocol headers*

UDP protocol yielded different performances with different machine learning algorithms as shown in Figure 4. On average of all algorithms, we observe 71.1% performance when all features are considered, while performance becomes 68.3%, on average, when subset of features are utilized. At its best, it has 80.0% performance of classification with the J48 algorithm while several algorithms have very similar results.

Overall, there are only four UDP features utilized by the algorithms as shown in Table VI yet these features allow classification at a rate of around 80%. The checksum feature is selected by every algorithm. As indicated earlier, this is likely due to the fact that the checksum contains a summary of the entire packet. Even though other features were also selected by certain algorithms, the results for J48 shows that even the checksum alone can perform classification performance

TABLE VI
SELECTED UDP FEATURES

Features	J48	JRip	Ridor	PART	DT	RF	NB	MP	Σ
checksum	✓	✓	✓	✓	✓	✓	✓	✓	8
srcport		✓			✓			✓	3
port		✓					✓		2
dstport					✓				1
Σ (features)	1	3	1	1	3	1	2	2	

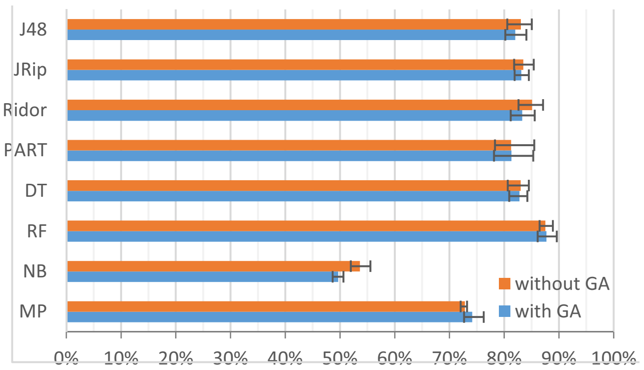

Fig. 5. HTTP Performance

similar to other algorithms that used additional features.

E. HTTP protocol headers

HTTP protocol yielded different good performance with most of the machine learning algorithms as shown in Figure 5. On average of all algorithms, we observe 78.7% performance when all features are considered, while performance becomes 78.0%, on average, when subset of features are utilized. At its best, it has 87.7% performance of classification with the RandomForest algorithm with subset of features.

For the HTTP protocol, as seen in Table VII, (prev_request_in) feature is selected for the majority of algorithms. This feature is the previous request in frame and is represented as a frame number. There are a few other features that were selected by most of the algorithms, namely; (accept_encoding), (cache_control) and (request.method).

F. DNS protocol headers

Among the protocols analyzed in this study, DNS was unsuccessful in correctly classifying operating system of the individual packets as seen in Figure 6. On average of all algorithms, we observe 29.2% performance when all features are considered, while performance becomes 30.0%, on average, when subset of features are utilized. At its best, it has 30.9% performance of classification with the J48 algorithm.

TABLE VII
SELECTED HTTP FEATURES

Features	J48	JRip	Ridor	PART	DT	RF	NB	MP	Σ
prev_request_in	✓	✓	✓	✓	✓	✓		✓	7
accept_encoding	✓	✓		✓			✓	✓	5
cache_control	✓			✓			✓	✓	4
request.method				✓			✓	✓	3
response		✓				✓			2
content_length		✓							1
connection		✓							1
content_length_header				✓					1
accept				✓					1
content_type				✓					1
notification								✓	1
prev_response_in								✓	1
request								✓	1
Σ (features)	3	5	1	7	1	2	3	7	

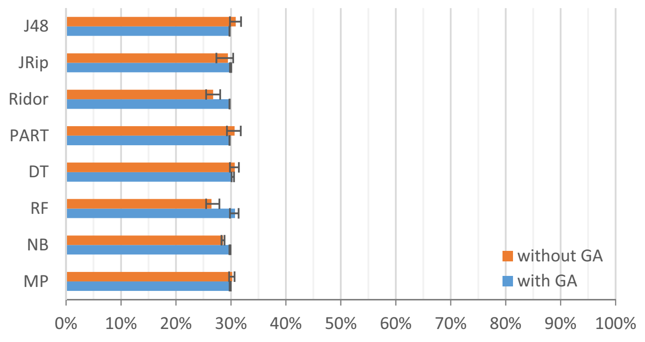

Fig. 6. DNS Performance

The features preferred by genetic algorithm for the DNS protocol, as seen in Table VIII, seem to be distributed among different machine learning algorithms. This inconsistent behavior is reflected on the poor classification results in Figure 6 as well. Genetic algorithm was not able to find very distinguishing features and therefore for each algorithm, selected features do not necessarily have a pattern. J48 and Ridor algorithms reached average performances by using only a single feature which is the (flags.truncated) and (flags.rcode) fields, respectively. Hence, while DNS header is not promising enough to be used by itself, certain features of the header can be used to determine the operating system of packets.

G. SSL protocol headers

On average, SSL protocol classification results were even lower than the DNS protocol as shown in Figure 7. On average of all algorithms, we observe 25.0% performance when all features are considered, while performance becomes 25.3%, on

TABLE VIII
SELECTED DNS FEATURES

Features	J48	JRip	Ridor	PART	DT	RF	NB	MP	Σ
soa.retry_interval		✓			✓	✓		✓	4
count.answers		✓			✓	✓			3
resp.cache_flush		✓			✓	✓			3
soa.serial_number		✓		✓		✓			3
flags.checkdisable		✓		✓					2
resp.len		✓		✓					2
flags.truncated	✓			✓					2
flags.conflict					✓	✓			2
flags.recdesired						✓	✓		2
qry.type						✓		✓	2
count.auth_rr						✓			1
flags						✓			1
flags.authoritative						✓			1
resp.class						✓			1
soa.expire_limit						✓			1
flags.opcode							✓		1
flags.recavail							✓		1
flags.authenticated					✓				1
resp.ttl				✓					1
flags.rcode			✓						1
qry.class		✓							1
soa.refresh_interval		✓							1
Σ (features)	1	8	1	5	5	12	3	2	

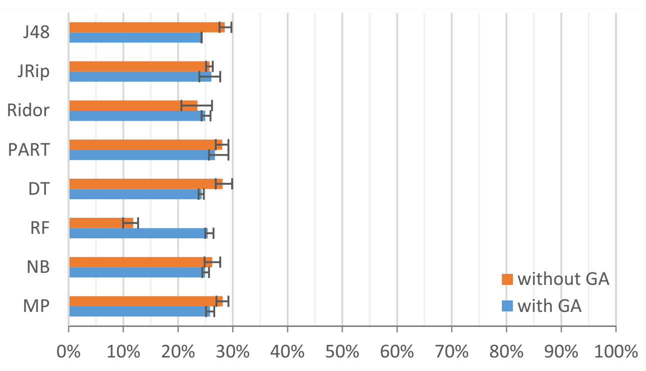

Fig. 7. SSL Performance

average, when subset of features are utilized. At its best, it has 28.5% performance of classification with the J48 algorithm.

Feature selection among different algorithms are also scattered around and there does not exist consistency among algorithms for the features as shown in Table IX. Only Cipher Suites Length (handshake.cipher_suites_length) is used by half of the algorithms for classification while Server Name length (handshake.extensions_server_name_len) gives close to the average performance by itself under the J48 algorithm.

H. SSH protocol headers

SSH is the second least successful in classification of the operating systems from packet header as shown in Figure 8. On average of all algorithms, we were able to correctly identify the operating system of 22.5% of SSH packets. This could be due to the fact that most of the packets' header information were identical which yielded very few unique packets to train the machine learning algorithms. We did not select features with genetic algorithm due to insufficient amount of packets remaining.

TABLE IX
SELECTED SSL FEATURES

Features	J48	JRip	Ridor	PART	DT	RF	NB	MP	Σ
handshake.cipher_suites_length			✓	✓		✓		✓	4
handshake.extensions_server_name_len	✓	✓						✓	3
handshake.session_id_length				✓	✓			✓	3
handshake.ciphersuite		✓					✓	✓	3
handshake.extension.len			✓	✓			✓		3
change_cipher_spec		✓	✓						2
handshake.length						✓	✓		2
record		✓					✓		2
handshake							✓	✓	2
handshake.extension.type							✓	✓	2
handshake.extensions_elliptic_curves		✓						✓	2
record.version		✓						✓	2
record.content_type		✓	✓						2
record.length		✓		✓					2
alert_message.desc		✓							1
alert_message.level								✓	1
handshake.extensions_elliptic_curves_length								✓	1
handshake.type								✓	1
handshake.ciphersuites					✓				1
handshake.extensions_ec_point_format					✓				1
handshake.extensions_ec_point_formats_length							✓		1
handshake.extensions_length		✓							1
Σ (features)	1	10	4	4	3	2	7	11	

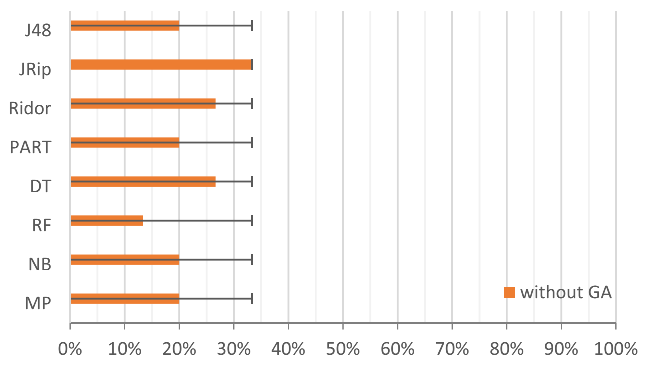

Fig. 8. SSH Performance

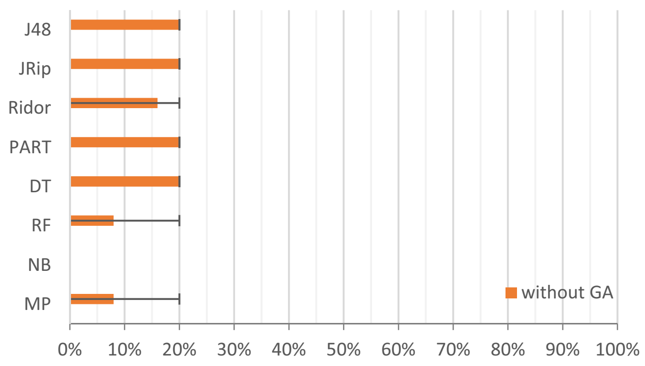

Fig. 9. FTP Performance

I. FTP protocol headers

FTP is the worst protocol header for classification of the operating systems from packet header as shown in Figure 9. On average of all algorithms, we were able to correctly identify the operating system of 14.0% of FTP packets. Similar to SSH, we did not select features with genetic algorithm due to insufficient amount of packets remaining.

V. CONCLUSION

In this paper, we presented the contribution level of popular protocols to classify the operating system of hosts from which the packets originated. We also presented how well certain machine learning algorithms performed for the operating system classification from TCP/IP protocol headers. By using genetic algorithm to select most distinguishing features, we demonstrated the amount of contribution that the selected features have in affecting the classification performance while reducing computation overhead in classifying OSes. Since classification was performed individually on the packets, results obtained are not bound to restrictions such as classification of certain packet types only (e.g. SYN packets in TCP). Overall, with IP, ICMP, TCP, UDP, HTTP, DNS, SSL, SSH, and FTP, protocol header information, on average, operating systems of 68.0%, 51.6%, 98.4%, 71.1%, 78.7%, 29.2%, 25.0%, 22.5%, and 14.0%, respectively, packets were correctly classified. Among analyzed protocols, TCP provided the best overall performance with all algorithms that completed analysis of TCP headers and we obtained close to perfect classification with all but one algorithm.

For future work, we would like to perform tests to see how well packets can be classified across different layers of TCP/IP protocol. For instance, one can analyze DNS, UDP, and IP headers of a DNS packet to see whether performance gains can be achieved. After performing layer-based tests, we would like to determine the features that can be used across the entire TCP/IP protocol which then would allow us to introduce a system for classifying operating systems of packets which would know exactly which features to check and provide as high classification performance as possible. We also would like to test the classification performance of background traffic and user generated network traffic. Finally, we plan to include mobile devices and other versions of the operating system families in the future.

Acknowledgement

This material is based upon work supported in part by the National Science Foundation under grant numbers CNS-1321164 and IIA-1301726.

We would also like to thank Mustafa Bilgic and George Bebis for their guidance on classification with machine learning and feature selection with genetic algorithm.

References

[1] O. Arkin, "A remote active os fingerprinting tool using icmp," *login: the Magazine of USENIX and Sage*, vol. 27, no. 2, pp. 14–19, 2002.

[2] E. Arslan, M. Yuksel, and M. H. Gunes, "Training network administrators in a game-like environment," *Journal of Network and Computer Applications*, 2015.

[3] P. Auffret, "Sinfp, unification of active and passive operating system fingerprinting," *Journal in computer virology*, vol. 6, no. 3, pp. 197–205, 2010.

[4] A. J. Bennieston, "Nmap-a stealth port scanner," 2004.

[5] R. Beverly, "A robust classifier for passive tcp/ip fingerprinting," in *Passive and Active Network Measurement*. Springer, 2004, pp. 158–167.

[6] D. Chang, Q. Zhang, and X. Li, "Study on os fingerprinting and nat/tethering based on dns log analysis."

[7] Y.-C. Chen, Y. Liao, M. Baldi, S.-J. Lee, and L. Qiu, "Os fingerprinting and tethering detection in mobile networks," in *Proceedings of the 2014 Conference on Internet Measurement Conference*. ACM, 2014, pp. 173–180.

[8] F. Gagnon and B. Esfandiari, "A hybrid approach to operating system discovery based on diagnosis," *International Journal of Network Management*, vol. 21, no. 2, pp. 106–119, 2011.

[9] L. G. Greenwald and T. J. Thomas, "Toward undetected operating system fingerprinting." *WOOT*, vol. 7, pp. 1–10, 2007.

[10] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, "The weka data mining software: an update," *ACM SIGKDD explorations newsletter*, vol. 11, no. 1, pp. 10–18, 2009.

[11] R. Kohavi and G. H. John, "Wrappers for feature subset selection," *Artificial intelligence*, vol. 97, no. 1, pp. 273–324, 1997.

[12] B. Li, M. H. Gunes, G. Bebis, and J. Springer, "A supervised machine learning approach to classify host roles using sflow," in *ACM HPDC - Workshop on High Performance and Programmable Networking (HPPN)*, New York City, NY, June 2013.

[13] B. Li, J. Springer, G. Bebis, and M. H. Gunes, "A survey of network flow applications," *Journal of Network and Computer Applications*, vol. 36, no. 2, pp. 567 – 581, 2013.

[14] R. Lippmann, D. Fried, K. Piwowarski, and W. Streilein, "Passive operating system identification from tcp/ip packet headers," in *Workshop on Data Mining for Computer Security*. Citeseer, 2003, p. 40.

[15] C. Mavrakis, "Passive asset discovery and operating system fingerprinting in industrial control system networks," Master's thesis, Technische Universiteit Eindhoven University of Technology, 10 2015.

[16] D. Michie, D. J. Spiegelhalter, and C. C. Taylor, "Machine learning, neural and statistical classification," 1994.

[17] C. Sarraute and J. Burroni, "Using neural networks to improve classical operating system fingerprinting techniques," *arXiv preprint arXiv:1006.1918*, 2010.

[18] L. Spitzner, "Passive fingerprinting," *FOCUS on Intrusion Detection: Passive Fingerprinting (May 3, 2000)*, pp. 1–4, 2000.

[19] G. Taleck, "Synscan: Towards complete tcp/ip fingerprinting," *CanSecWest, Vancouver BC, Canada*, 2004.

[20] F. Veysset, O. Courtay, O. Heen, I. Team *et al.*, "New tool and technique for remote operating system fingerprinting," *Intranode Software Technologies*, 2002.