Operating System Fingerprinting via Automated Network Traffic Analysis

Ahmet Aksoy  
Computing Science and Engineering  
University of Nevada, Reno  
Email: aksoy@nevada.unr.edu

Sushil Louis  
Computing Science and Engineering  
University of Nevada, Reno  
Email: sushil@cse.unr.edu

Mehmet Hadi Gunes  
Computing Science and Engineering  
University of Nevada, Reno  
Email: mgunes@cse.unr.edu

Abstract—Operating System (OS) detection significantly impacts network management and security. Current OS classification systems used by administrators use human-expert generated network signatures for classification. In this study, we investigate an automated approach for classifying host OS by analyzing the network packets generated by them without relying on human experts. While earlier approaches look for certain packets such as SYN packets, our approach is able to use any TCP/IP packet to determine the host systems’ OS. We use genetic algorithms for feature subset selection in three machine learning algorithms (i.e., OneR, Random Forest and Decision Trees) to classify host OS by analyzing network packets. With the help of feature subset selection and machine learning, we can automatically detect the difference in network behaviors of OSs and also adapt to new OSs. Results show that the genetic algorithm significantly reduces the number of packet features to be analyzed while increasing the classification performance.

Index Terms—Genetic algorithm, Machine learning, OS classification

I. INTRODUCTION

Operating System (OS) fingerprinting is the process of remotely detecting a system’s OS through packet signatures generated by it. Detecting OS is important for determining system vulnerabilities, improving cyber security, and identifying potential attacks. OS detection also helps network administrators better manage and monitor large numbers of hosts [1]. Such detection can also be extended for classification of host roles [2].

There are two types of OS fingerprinting: active and passive. Active fingerprinting detects OS by probing (generating and sending network packets to) the system and analyzing the received response [3]. Passive fingerprinting sniffs packets originating from the system and analyzes information extracted from the sniffed packets. These fingerprinting techniques have complementary advantages and disadvantages. Active fingerprinting tools usually generate high-accuracy results since they have the ability to access or request information that is useful to distinguish OSs. However, it is possible for such systems to be blocked out by firewalls or intrusion detection systems (IDS) and not be able to send packets to the probing system. Active methods might also disrupt network traffic as they incur probing. Passive fingerprinting tools are advantageous in the sense that they do not get blocked out as they passively sniff packets. However, the disadvantage of passive tools is that they are stuck with what they get. If the system they’re analyzing does not generate packets that are helpful for distinguishing the OS, they might not be able to accurately detect the OS.

In this paper, we present a completely machine learning dependent OS classification approach and perform single-packet OS classification on network packets which originate from host devices. Specifically, we perform OS classification on a single sniffed packet by extracting and checking the packet’s protocol header information.

Different types of network packets yield different features that can be used for classification. For each type of network packet, and for each of the three classifiers that we use, a genetic algorithm (GA) evolves a subset of packet-type dependent features relevant to OS classification. The evolved feature subsets have fewer features and the lower numbers of features enable faster system classification performance. Furthermore, these evolved features generally lead to improved classification accuracy and lower classifier complexity. We conjecture that the GA eliminates features that do not contribute to accuracy or add noise and thus decrease accuracy. The reason why we used GA is due to large search space of feature combinations in the protocols. For instance, for TCP, there are 61 features which makes $2^{61}$ unique combinations to consider. We tried a hill-climber but it did not yield as high classification results as GA. Also, the presented approach is applicable to both active and passive fingerprinting but since we collected packets passively, the results presented in this paper are for passive fingerprinting.

Based on results from a prior network measurement study that explores various protocols and machine learning algorithms, we use four machine learning algorithms in this paper [3]. These algorithms, also supported by the open source WEKA tool set, are ZeroR, OneR, Random Forest and Decision Trees (called J48 in WEKA) [4]. ZeroR is a simple classification algorithm in WEKA which focuses merely on the classes in the dataset. It performs predictions on the class that appears the most in the dataset. OneR, however, selects a single rule with the lowest error rate and performs classification on this rule alone.

Feature subset selection is computationally expensive but cheap, distributed computing power makes this a non-issue in practice. Collecting the data needed for training however is a different issue. Our approach’s accuracy depends on the quality and quantity of data collected.
Our approach has several unique features. First, we generate our own set of signatures to perform OS classification on newly seen packets and do not depend on any other fingerprinting tools or their databases. Second, unlike other approaches to passive OS fingerprinting that hand select “useful” features like TTL, window size, and de-fragmentation flag, we have no bias. The GA generates the set of most useful features based on the data collected and the specific bias of the machine learning approach used. Third, existing tools such as p0f, ettercap and siphon seek a perfect exemplar data match when classifying a packet [5], [6]. However, it is possible for a non-perfect match to occur in some cases. Rather than discarding such packets, our machine learning techniques learn to classify the packets. Fourth, an advantage of not depending on expert signatures, but rather using machine learning techniques is that we can dynamically adapt our generated models to different OSs and other dynamic environmental changes. By providing training data from the newly introduced OSs, we can automatically re-generate our classifier system’s set of signatures to include these new OSs. When newly created proxy firewalls tamper with packet header information [6], the system can learn to adapt. Fifth, many passive fingerprinting tools such as p0f, ettercap and siphon depend on specific packet types such as SYN, ACK or SYN-ACK [6]. Unlike these systems with such strong constraints, we can perform classification on any packet. We also do not depend on specific network protocol packets for performing OS classification. Any TCP/IP header protocols can be used for performing OS classification using our approach. In [3], we have presented the protocols which contribute the most for performing OS classification. Finally, tools such as SinFP, nmap, p0f, etc. usually have preset number of features that they use in order to generate their signatures for OSs. A high preset number has the potential to increase the number of redundant features that do not necessarily contribute to the classification. According to [7], the second-generation Nmap’s database contains 4,766 signatures where 17% of these belong to different varieties of Linux and introduce much redundancy. A low preset number may lose critical features and degrade accuracy. With our approach, however, the GA evolves the number of features based on a fitness function that moves towards high classification accuracy and small feature subset size. The provided data and the machine learning algorithm determine feature selection and signature size reducing administrator burden.

The rest of the paper is organized as follows. Section II describes related work. Section III provides details on our proposed approach and experimental methodology. Section IV shows performance results of our approach and the last section provides conclusions and future work.

II. RELATED WORK

Both active and passive fingerprinting approaches have received much attention over the last few years and many tools have been developed for these approaches. As noted earlier, active fingerprinting tools can be blocked by firewalls and IDSs. Such tools also require numerous probes in order to accurately classify OSs. Even though there is work on trying to reduce the number of probes required, e.g. [8], we cannot guarantee that the system will not block packets generated by these tools before the system gets enough information to accurately classify the system. Often, such tools are not even aware of whether it is the firewall or the actual system that they are scanning [9].

Passive fingerprinting is more limited than active fingerprinting since passive systems cannot choose which type of packets and therefore which type of information to use for classification. Since passive fingerprinting techniques sniff packets, these systems are limited to the information they are provided.

A. Active fingerprinting tools

Nmap is an active OS fingerprinting tool introduced by Gordon Lyon [9]. It has received multiple improvements over the years which gave it the ability to classify various OSs. However, since it makes up to 16 probes to be able to make a decision, it becomes easily detectable and blockable.

Xprobe2 is another active fingerprinting tool [7] and mostly uses ICMP protocol probes to perform fingerprinting. Xprobe2 is able to perform partial matching and using ICMP enables Xprobe2 to distinguish similar OSs such as different Windows versions.

Carlos Sarraute and Javier Burroni presented a system which uses neural networks for fingerprinting [10]. In addition to Nmap’s signature database, they train a neural network to distinguish different Windows versions. They try to initially distinguish the main OS type (Windows, Linux, Mac) of packets using neural networks, and then Nmap’s signature database to further classify the specific OS version, for example as Windows 7.

B. Passive fingerprinting tools

p0f was introduced by Michal Zalewski [9] and is widely used. p0f extracts header information from TCP SYN packets which then are compared to a database of signatures for OS classification.

Ettercap, detects man-in-the-middle attacks and is also able to perform OS fingerprinting. Like p0f, Ettercap uses TCP SYN packet information for OS classification. This tool, however, is not a pure passive tool since it sends SYN packets to the system and checks responses.

Another novel method that performs OS fingerprinting using DNS log analysis was presented by [11]. There also exist hybrid systems such as SinFP which try to reap the benefits of both approaches [12]. SinFP also introduced methods such as using signatures collected from one system to perform classification on another. They also perform active and passive OS fingerprinting with IPv6. Like p0f, SinFP relies on TCP SYN packets for fingerprinting.

Another hybrid approach was introduced in [13]. In this study, authors use the answer set programming where the problem of OS classification is solved through automated reasoning.
There also exist tools which focus on performing OS fingerprinting using IPv6 packets [14] where they analyze data obtained from passive measurements in order to perform comparison between IPv4 and IPv6 data.

C. Mobile OS fingerprinting tools

There is some work in Mobile OS classification as well. In [15], the authors try to improve the classification of Mobile OSs by introducing new features. Their approach implements Bayes’ rule to perform classification. [16], in addition to trying to improve p0f’s classification capabilities further using machine learning, include Mobile OS classification such as smartphones.

D. Genetic algorithms for Feature Subset Selection

Genetic algorithms (GA) can be used to select feature subsets from a dataset. Feature subset selection is the process of selecting a smaller set of features from a larger set based on an optimization criteria [17]. Feature subset selection helps increase classification performance by selecting features which contribute to the classification the most and increase efficiency due to less number of features to process [18].

There exist many work on feature selection using GA. Most of these work depend on wrappers where different machine learning algorithms are used to evaluate subsets of features selected by GA [19]. However, there are work on applying GA on clustering using filter methods as well [20]. Due to classification performance and simplicity, SVMs and KNN are among the most preferred algorithms. Researchers analyzed how different values for population size, mutation and crossover effect the performance [21].

Studies such as [19] and [22] do not consider feature interaction. Elimination of feature interaction potentially creates an issue of elimination of features which might yield better classification performance together as opposed to being used individually. There are traditional approaches to GA feature selection as well. For example, in [19], authors combine both SFFS and GA to select features. The results are claimed to be improved with such hybrid approaches. However, this introduces extra computation overhead.

There also exist work where GA is used along with a classifier to improve the original classifier’s performance. Kelly and Davis [23] use GA and K Nearest Neighbor (KNN) algorithms in conjunction to improve the performance. GA helps increase KNN’s accuracy by searching a weight vector. Results show that KNN along with GA performs better than KNN alone [24].

III. OS FINGERPRINTING OF NETWORK TRAFFIC

This paper presents an entirely automated machine learning approach for classifying OSs using TCP/IP packets. We use genetic algorithm (GA) feature subset selection to determine relevant packet header features for learning to classify operating systems. To the best of our knowledge, our approach is the first to use GA for feature subset selection in OS fingerprinting. In this paper we investigate classification accuracy using TCP, IP, and UDP packets and extract features from these packet headers. Broadly speaking, each member of the GA specifies the set of features to use for classification and the fitness of this individual is obtained by running a classifier with the individual specified feature set on training data and using performance on a test set as fitness. We describe the process in more detail below.

A. Data Collection

We setup and used a local network consisting of three PCs, three Macs and three Raspberry Pi’s. To eliminate hardware bias we collect packets from three instances of each OS on each hardware platform. Each instance, on average, contains around 79K packets from a single machine for every protocol and OS. We used Raspberry OS, Xubuntu 14.04, Windows 7, Windows 8, Mac OSX Elcapitan and Mac OSX Lion. As shown in Figure 1, we dedicated two instances for each OS for training and validation, and the third for testing.

Data from these machines was generated by visiting the top 10 websites mentioned on Alexa’s The top 500 sites on the web (http://www.alexa.com/topsites). These websites were: www.google.com, www.youtube.com, www.facebook.com, www.baidu.com, www.yahoo.com, www.amazon.com, www.wikipedia.org, www.twitter.com, www.google.co.in and www.qq.com. We also collected 30 seconds of Youtube video streaming packets from every OS on every device. To generate FTP and SSH protocol packets, we connected to GoDaddy and CSE UNR’s servers (our department) and uploaded files to these servers. To generate ICMP packets, we used the traceroute application to connect to these top 10 websites mentioned above. We also performed ping tests on the top 10 websites. However, since Amazon servers did not allow ping tests, we used the 11th website on Alexa’s website list, www.live.com. We also performed 30 seconds of Skype calls on these OSs. Finally, we sent and received e-mails from these OSs.

Our setup enables us to collect IP, ICMP, UDP, DNS, HTTP, IGMP, TCP, FTP, SSH and SSL protocol header packets and enables testing using every possible header field of these protocols. In addition, we can use header fields selected by the GA. Our prior work has shown that IP, TCP, and UDP were among the best for OS fingerprinting [3]. In this study, we classify using these protocols’ header fields. Data collected is stored in two pcap (a specific format for storing network data) files for each OS for training and one pcap file for testing. We extract all possible features for every packet in the pcap files.
These features are saved to files and converted to arff format in order to make them compatible with the WEKA tool [4]. We then run our GA for selecting relevant features.

B. Feature Selection

We use 2 instances of each OS for selecting features using GA. We want maximum classification accuracy therefore the fitness function used by our GA implementation is $Fitness = \sum_{i=1}^{n} Accuracy_i$, where $n$ is the number of instances of OSs, $Accuracy$ is the classification performance with the provided machine learning algorithm.

For every chromosome that the GA wants to evaluate, an arff file for two out of the three OS instances is created containing the GA specified features. Let us call the first instance $I_1$ and the second $I_2$. Weka builds a classifier by training on $I_1$ and testing on $I_2$. We then train on $I_2$ and test on $I_1$. Fitness is the sum of testing performances of the two combinations. The results presented on this paper are based on a single GA evolutionary process. However, we observed consistency in our results when we ran it multiple times.

The GA converges when we have five consecutive generations with no classification performance improvement. Although we investigated other termination criteria, our strategy works well in practice.

We use a population size of 50 for the GA and simple binary encoding for the chromosome. Each bit represents a feature, 1 means the feature is selected and 0 that it is not. The GA we used implements Elitism selection. Initially, the best individual is saved for the next population. For each of the remaining individuals in the population, two sets of 5 randomly selected individuals are chosen and the best ones among the two sets are selected for performing crossover. Uniform crossover method is used where bits are randomly exchanged between two individuals. The resulted individual is saved for the next population. Then each of the newly generated individuals are looped over and mutated. For mutation, each bits other than the best one, is inverted with a probability. The mutation rate is 0.015 and the crossover rate is 0.5.

C. OS Classification

We use machine learning algorithms in the WEKA tool [4] for OS classification and in this paper, use J48, RandomForest, OneR, and ZeroR. We generated four separate classifiers in a two level hierarchy as shown in Figure 2. The first classifier learns to classify the OS genre: Linux, Windows or Mac. Once we know a packet’s OS genre, the packet is passed on to one of three next level classifiers. The three level two classifiers, one for each OS genre, learn to specify OS version assuming that the packet belongs to a specific OS genre. Basically, if the classifier in the first layer classifies a packet as Mac, then it is sent to the Mac classifier in layer two to find out if it is Mac ElCapitan or Mac Lion. This architecture helps improve classification performance since different protocols and different machine learning algorithms perform better than the others for possible packet origin scenarios.

IV. EXPERIMENTAL RESULTS

As mentioned in Sections III-A and III-B, we dedicated three pcap files for each OS for training, validating and testing our approach’s performance. Two of the pcap files were used to validate and train the machine learning models. Validation is the process of determining relevant features using GA where we used one of two pcaps to train and the other to test the classification performance with the GA selected subset of features. After determining relevant features, both of the pcaps used for validation are merged into a single pcap file. The merged file is then used to train the machine learning model with the selected subset of features and then is tested with the remaining third pcap file which is only used to test the overall performance. Since we have three pcap files, there are three possible different combinations of selecting among these pcap files (e.g. pcap 1 & 2 for training, pcap 3 for testing; pcap 1 & 3 for training, pcap 2 for testing, etc.). We applied our approach for all three possible combinations and recorded the average performances. The results presented in this section are averages of all three combinations.

In this section, we present the results for the optimum performances for each of the classifiers shown in Figure 2. The optimum case is the highest classification performance each of the classifiers would yield with the data at hand. Since there exist two levels of classifiers, the optimum case would be when the second layer classifiers receive packets from the first layer which belong to the OSs that the classifiers in the second layer are responsible for classifying at 100% confidence.

A. OS Genre results

For the classifier in the first layer, we merged packets which belong to the same genre of OSs. After merging, the data contains packets which belong to three classes: Linux, Mac and Windows. Figure 3 shows the classification performances of the genre classifier for IP, TCP and UDP protocols, respectively. IP packets generally give nice results. Both J48 and RandomForest generate classification performance greater than 80%. However, the highest performance was generated with the use of TCP packets. J48, along with GA feature selection was able to generate the highest performance at the rate of 86%. Even though it was not as high as J48, RandomForest was able to generate good performance at 84%. UDP however was not as useful as the other two protocols. The highest performance was at 66% with J48 with GA feature subset selection. The GA was able to select features which helped increase classification performances significantly in many cases. For IP, classification performance with GA selected features was either equal to or very close to the
classification performance using all features. For TCP, when J48 algorithm was used, the performance without GA feature selection was at 69% and with GA was at 86%. When OneR algorithm was used, performance increased to 76% with GA feature selection from 35% without. For UDP, GA selected features helped improve the performance of J48, RandomForest and OneR compared to performance using all features.

B. Linux OS results

Figure 4 shows the classification performances for Linux. For IP, features selected by the GA seem to generate better results than the full set. For three of the algorithms used, GA selected feature set provide similar performances at a rate of 76%. The highest rate was achieved with the OneR algorithm at 76.3%. For TCP, except for the Random Forest algorithm, the rest were able to perform at higher rates when GA is used. However, OneR algorithm results far surpasses the others at a rate of 95.3%. This is also the highest classification rate among all three protocols. For UDP, the GA in most cases was not able to select subsets of features which would perform better than the full set except for with J48. The highest performance achieved was again by J48 at 75%. Since the highest performance was yielded by the TCP protocol with OneR, as listed in Table V, TCP along with OneR were set to be used to perform classification for Linux OSs. For IP packets, the GA was able to increase classification performance from 51% to 76% for J48, from 64% to 76%, for RandomForest, and from 51% to 76% for OneR. For TCP, the GA was able to perform the best with the OneR where the performance was increased from 58% to 95% which yielded the best performance for classifying Linux OSs. Since the UDP protocol is not very contributing to OS classification, GA was not able to find the optimal subsets for RandomForest and OneR algorithms. However, GA for the J48 algorithm was still able to increase the performance from 72% to 75%.
C. Mac OS results

Figure 5 shows the classification performances for Mac OS. IP packets do not seem to contain features usable by our machine learners to achieve good accuracy. Deviation levels as shown in the figures seem to be high for almost all the algorithms indicating that IP packets may not be able to reliably classify Mac versions. The deviation levels for TCP packet data are also high for when all the features are used. However, with the help of GA feature selection, deviation levels decrease considerably especially for J48. J48 is also the best performing algorithm for TCP protocol at a rate of 96%. OneR also gives as nice results as J48 at a rate of 95%. Even though UDP packet data gives better results than IP, performance is still low compared to TCP. The highest performance achieved was 55% which is a lot worse than what TCP provides. Therefore, for Mac OS classification, TCP along with J48 were set to be used. It is also important to note that according to p0f v3 signatures [5], p0f can identify Mac versions when they are either 10.x or 10.9. Mac OS X Lion is version 10.7 and Mac OS X ElCapitan is version 10.11. According to these signatures, p0f should not be able to distinguish between Lion and ElCapitan where for the first signature, both of these algorithms fall into the same category and in the second, only ElCapitan falls into the category. However, with our approach, we can tell these two versions apart at a rate of 96%. The GA was able to increase the classification in all algorithms for TCP protocol packets. Specifically, GA feature selection was able to increase performance from 67% to 96% for J48, 86% to 92% for RandomForest and 63% to 95% for OneR algorithm. It should also be noted that when the GA is used, the deviation levels seem to decrease significantly which yields more reliable results. The GA was also able increase the classification performances for UDP protocol packets.
TABLE I
NUMBER OF SELECTED FEATURES (OS)

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Protocol</th>
<th>Algorithm</th>
<th>GA</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>J48</td>
<td>10</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OneR</td>
<td></td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>J48</td>
<td>28</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>29</td>
<td>61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OneR</td>
<td></td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDP</td>
<td>J48</td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>4</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OneR</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE II
NUMBER OF SELECTED FEATURES (LINUX)

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Protocol</th>
<th>Algorithm</th>
<th>GA</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>J48</td>
<td>6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OneR</td>
<td></td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>J48</td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>24</td>
<td>59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OneR</td>
<td></td>
<td>26</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDP</td>
<td>J48</td>
<td>3</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>3</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OneR</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE III
NUMBER OF SELECTED FEATURES (MAC)

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Protocol</th>
<th>Algorithm</th>
<th>GA</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>J48</td>
<td>11</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OneR</td>
<td></td>
<td>9</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>J48</td>
<td>25</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>29</td>
<td>61</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OneR</td>
<td></td>
<td>27</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDP</td>
<td>J48</td>
<td>4</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>3</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OneR</td>
<td></td>
<td>3</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

TABLE IV
NUMBER OF SELECTED FEATURES (WINDOWS)

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Protocol</th>
<th>Algorithm</th>
<th>GA</th>
<th>ALL</th>
</tr>
</thead>
<tbody>
<tr>
<td>IP</td>
<td>J48</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OneR</td>
<td></td>
<td>5</td>
<td></td>
<td></td>
</tr>
<tr>
<td>TCP</td>
<td>J48</td>
<td>23</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>27</td>
<td>59</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OneR</td>
<td></td>
<td>29</td>
<td></td>
<td></td>
</tr>
<tr>
<td>UDP</td>
<td>J48</td>
<td>1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>RF</td>
<td>1</td>
<td>7</td>
<td></td>
<td></td>
</tr>
<tr>
<td>OneR</td>
<td></td>
<td>2</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

D. Windows OS results

Figure 6 shows the classification performances for Windows OS. As the results show, the classification performance of distinguishing between Windows 7 and Windows 8 are almost 50%. This suggests that the behaviors of these two versions of OSs are very alike. p0f’s signature database also has a single label for classifying these two versions of Windows OSs. This indicates that the network libraries of these Windows versions have no distinguishing feature. Due to such similarity, we have merged Windows 7 and Windows 8 packets and simply considered them to be Windows. However, if different versions of Windows OSs with distinguishable behaviors were included, this classifier can be used to further specify them.

The lowest entropy features are considered to be the most contributing ones to the OS classification since they provide the most information gain. The lowest entropy feature selected by the J48 algorithm for classifying the OS genres was related to the TCP window size. This is consistent with the signatures of p0f as well. In p0f’s signatures, the first term of the signatures is the window size. As it is known, OneR algorithm tries to find a single rule to perform classification. The only rule selected by the OneR algorithm for classifying Linux OSs was also the window size. Similarly, the lowest entropy feature selected by J48 algorithm for classifying Linux OSs was also the window size. This shows that our approach is able to determine features that are known to be helpful for classifying OSs. However, more specific features do get selected further down the process in order to better learn the differences of OS behaviors.

E. Feature Subset Selection

Tables I, II, III and IV show the number of features GA selected for each protocol for OS, Linux, Mac and Windows respectively. Although we have not specified the weight for reducing the number of features in our fitness function, GA was still able to select smaller subsets for each scenario. This shows us that there exist many features which either do not contribute at all or little to the OS classification.

After comparing classification performances of different protocols with different algorithms for each classifier, we have selected the ones which yield the highest classification performances. The protocols and algorithms each classifier uses are shown in Table V.

We believe the reason behind certain algorithms performing better than the others is due to both the inbuilt biases within each machine learning algorithm and selection of features. For example, for the Linux OS classification, the GA selected a single feature for OneR, the TCP window size while the GA selected multiple features for J48 along with the TCP window size. Although features other than TCP window size might yield better results during training, it may not necessarily yield better results during testing.

TABLE V
CLASSIFIER SETTINGS

<table>
<thead>
<tr>
<th>Classifier</th>
<th>Protocol</th>
<th>Algorithm</th>
<th># of features</th>
<th>Performance</th>
</tr>
</thead>
<tbody>
<tr>
<td>OS</td>
<td>TCP</td>
<td>J48</td>
<td>28/61</td>
<td>85.9%</td>
</tr>
<tr>
<td>Linux</td>
<td>TCP</td>
<td>OneR</td>
<td>26/59</td>
<td>95.3%</td>
</tr>
<tr>
<td>Mac</td>
<td>TCP</td>
<td>J48</td>
<td>25/59</td>
<td>95.8%</td>
</tr>
</tbody>
</table>
V. CONCLUSION AND FUTURE WORK

We developed a genetic algorithm (GA) feature selection mechanism for machine learning for OS fingerprinting. With this approach, we can perform single-packet OS classification with high classification accuracy. The GA is able to find smaller and thus more efficient sets of features to perform OS classification. Unlike expert signature generation for OS classification, our machine learning algorithms can automatically and dynamically generate classification signatures. The fact that we use GA feature selection for machine learning algorithms to generate classifiers, allows us to dynamically adapt our approach to different OSs without expert input and unlike many available approaches to OS fingerprinting, we do not depend on specific types of packets to perform classification. Our results show that GA feature subset selection was able to increase OS classification performances significantly for OS genre classification for Linux and Macs. We were able to increase the overall classification performance from 69% to 86% for the OS genre, 58% to 95% for the Linux OSs and 58% to 95% for the Mac OSs. These results were achieved with much fewer features than the initial set of features.

In the future, we would like to convert this work into a system that can be used without individual training of OSs. We would also like to see how well our approach would work with a sequence of packets classification rather than a single packet. This would allow us to achieve higher accuracy by combining results from a sequence of packets. Note that even though we have not tested our approach on IPv6, we believe that our approach can be applied to perform classification on IPv6 packets as well.

ACKNOWLEDGEMENT

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

REFERENCES