Operating System Fingerprinting using IPv6 Packets and Machine Learning Techniques

Adrian Ordorica
Computer Science and Computer Engineering Department
University of Arkansas
Fayetteville, AR 72701
1(918) 978-5400
aordoric@uark.edu

Dale R. Thompson
Computer Science and Computer Engineering Department
University of Arkansas
Fayetteville, AR 72701 USA
1(479) 575-5090
drt@uark.edu

ABSTRACT
In this paper, we discuss a new approach on operating system (OS) fingerprinting using IPv6 packets and supervised machine learning techniques. OS fingerprinting tools are essential for the reconnaissance phase of penetration testing. While OS fingerprinting is traditionally performed by passive or active tools that use fingerprint databases, very little work has focused on using machine learning techniques. Moreover, significantly more work has focused on IPv4 than IPv6. We introduce a collaborative neural network system that uses a voting design to deliver accurate predictions. This method uses IPv6 features as well as data link features for OS fingerprinting. Our experiment shows that our approach is valid and we achieve an average accuracy of 86% over 100 sets of neural networks with a highest accuracy of 96%. Finally, we explore the impact of additional training for poor neural network accuracy, and we show that our system can achieve an average accuracy of 92%, which is a 6% improvement over the previous approach.

Categories and Subject Descriptors
C.2.3 [**Network Monitoring**]; F.1.1 [**Self-modifying machines (e.g., neural networks)**];

General Terms
Measurement, Design, Reliability, Experimentation, Security.

Keywords
OS fingerprinting, Supervised Machine Learning, IPv6, Computer Networks, Neural Networks.

1. INTRODUCTION
Internet Protocol version 6 (IPv6) is the most recent numbering system that provides more IP addresses than Internet Protocol version 4 (IPv4). The growing need for IPv6 is slow but inevitable with rising IP address consumption. The new address space uses eight sets of four hexadecimal addresses separated by a colon (:) like: xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx:xxxx (x would be a hexadecimal value) providing up to 3.42 x 10^{38} total addresses. IPv6 simplified header structures lead to faster routing compared to IPv4. Different operating systems (OS) have different implementations of IPv6 that exhibit slight variations in the protocol. These features can be used to do passive identification of the operating system, which is sometimes called OS fingerprinting.

OS fingerprinting is important to network security with its relationship to the reconnaissance phase of penetration testing. Knowing the OS is essential for attackers to accordingly use tools and programs when gaining access to their targets. The network layer of the Open Systems Interconnection (OSI) model does not contain any explicit information about the operating system of the network device generating traffic. However, certain features are unique to each operating system.

Machine learning focuses on the ability for computers to learn without being explicitly programmed. This is achieved with a combination of algorithms and simulated neural networks. These simulated neural networks are intertwined weights that adjust after passing in features (input) transformed by an activation function and taking the difference between the actual labels and the prediction (output). A greedy algorithm known as backpropagation provides a fast solution to pattern recognition in this supervised learning experiment.

Present passive OS fingerprinting methods match the gathered network traffic with previously developed IPv6 signature databases. The approach in this work is to use supervised machine learning techniques to learn the slight variations in the IPv6 network implementations of different OS's.

The rest of the paper is organized as follows: Section 2 covers the foundation of IPv6 and the contrast to IPv4, as well as an overview of OS fingerprinting and machine learning techniques. The methods for performing passive OS fingerprinting are discussed in Section 3. Section 4 presents the results comparing a variety of setups. Future work is discussed in Section 5 and the conclusions are in Section 6.

2. RELATED WORK
As shown in Figure 1, the format of IPv6 packets is designed to be simpler than the IPv4 format. The IPv6 header is reduced to 8 fields from 14 fields in the IPv4 header. Four of the field names and positions have changed but functionality remained the same in the IPv6 header. The flow label is the only new field that is a quality of service mechanism to avoid congestion of the network [9]. There are 7 fields removed whose functionality has been grouped and moved to a next header field. The next header field is optional, yet can chain additional headers if the packet requires additional options or information.

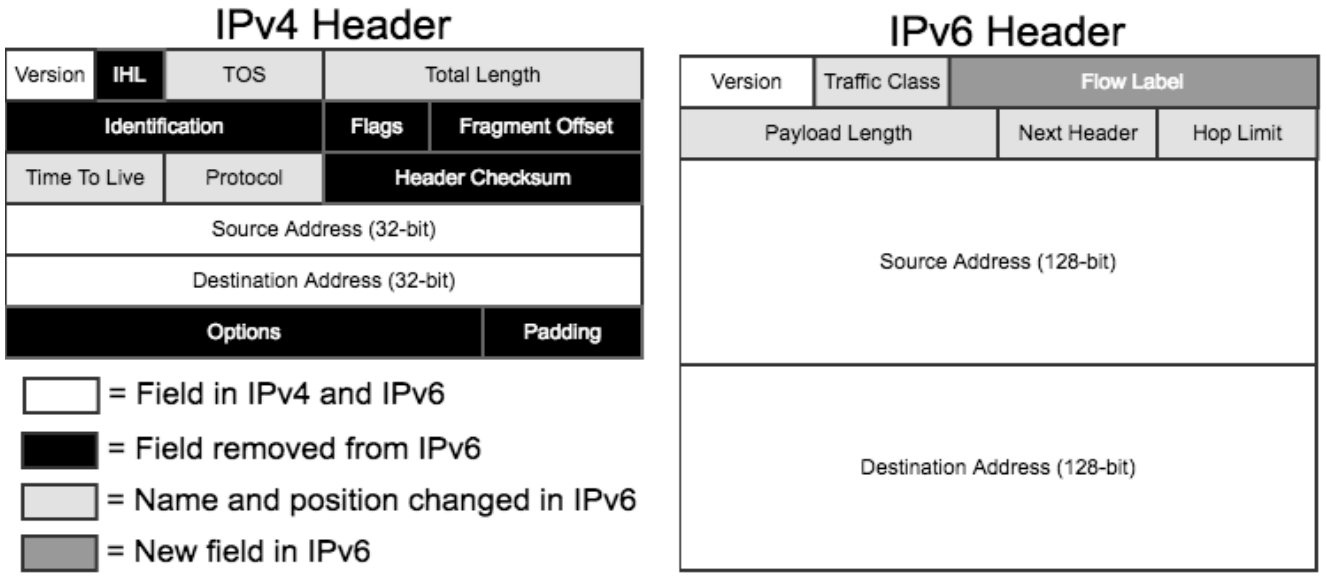

Figure 1. Comparison of IPv4 and IPv6 Headers [6]

In IPv6, Neighborhood Discovery Protocol (NDP) is responsible for the auto configuration of nodes on the network. NDP is meant to replace the Address Resolution Protocol (ARP), the Internet Control Message Protocol (ICMP), and the Internet Router Discovery Protocol (IRDP) from IPv4. The NDP uses five ICMPv6 packet types: Router Solicitation (RS), Router Advertisement (RA), Neighbor Solicitation (NS), Neighbor Advertisement (NA), and Redirect [8]. NDP conducts the Stateless Address Autoconfiguration (SLAAC) method as a device joins the network. There is an opportunity for passive OS identification when a device uses NDP to join a network. In addition, observing the NS and NA packets can identify the OS of nodes that have already joined the network and are communicating.

OS fingerprinting techniques can be categorized as passive or active. Passive techniques do not send any traffic and rely on collecting regular traffic for analysis. Active techniques send crafted packets to identify systems based on responses, or the lack thereof. Active scanning and probing of the IPv6 space is impractical and an open problem due to the large address space unlike the IPv4 address space [3]. Fingerprinting tools, such as Nmap, require manual query development and accurate classification models. Nmap mentions that IPv6 OS detection is used just like IPv4: send probes, collect responses, and match a set of responses against a database [4]. However, these probes are seeking responses from higher-level protocols like TCP and require an expert-user to craft packets that can accurately identify nodes on the network.

Passive tools, such as p0f, use purely passive mechanisms to identify the operating system and software of both ends of a vanilla TCP connection without interfering with the communication in any way [14]. P0f uses a fingerprint database to lookup signatures of collected packets. These signatures will include data extracted from the IPv4 and IPv6 headers, TCP headers, dynamics of the TCP handshake, and application payloads. While there is a database of IPv6 signatures, there is a stronger focus to expand the database of IPv4 signatures. Another passive tool, siphon, is a network mapping tool whose behavior is similar to p0f as it identifies network hosts using TCP headers, UDP headers, and the IPv4 header. However, this tool is outdated and does not include IPv6 network traffic identification [11].

An artificial neural network is a computational model that simulates the structure and functionality of a biological neural network. Neural networks are often characterized by their topology and ability to change interconnected weights in-order to form a response to input patterns. The backpropagation algorithm is commonly used to train multi-layer neural networks and has been effective for supervised learning [7, 12, 13]. Machine learning techniques have been used to uncover the IPv6 structure [10]. In [2], Weka, a well-known data mining toolkit, is used to generate OS fingerprints based on IPv4 and TCP features. However, to our knowledge using neural networks to perform passive OS fingerprinting based on IPv6 is not found in the literature.

3. METHODOLOGY

3.1 Data Collection

Our dataset consists of all traffic received by a network consisting of twenty computers, a router, and a switch with one port mirroring traffic to the collecting computer running Wireshark. All the machines can dual-boot Windows 10 and Linux Ubuntu. The data is collected in separate instances where all the machines are running the same OS. The machines are booted and collection starts before powering on the router and switch. After data collection, the data is filtered to contain only IPv6 and packets with a source address from the router and collecting computer are removed. The data is converted from byte code to a PDML XML file, where a parser extracts features from the link-local layer up to the IPv6 protocol and translates them into an Attribute-Relation File Format (ARFF).

Table 1 shows the number of input packets per OS. Although there are more Linux packets that could cause bias in the neural network, the Linux characteristics required more data to establish a distinction from Windows characteristics. All machines were powered on and left at the login screen to prevent any services from starting upon login and causing noise from application data. The majority of packets include ICMP packets from the NDP, with the remaining packets consisting of UDP standard queries, solicitations, DNS queries, and errors.

Table 1. Number of Packets per OS

	Overall	Average Training Set	Average Test Set
Windows	6,482	5,186	1,296
Linux	9,494	7,595	1,899
Total	15,976	12,781	3,195

3.2 Features

Based on manual inspection, the six features chosen to identify an OS show subtle differences in two layers of the OSI model, namely the link layer and the network layer. Excluding the source address and destination address, there are six fields in the IPv6 header. Since the neural network is learning distinctions, the version field is not used as a feature. In addition, the traffic class field was not included as a feature since no prioritization would have taken place. RFC 6437, IPv6 flow label specification, strongly recommends using a uniform, pseudo-random value when using a flow label [1]. Therefore, the flow label was not used as a feature because a randomly generated value will not contribute to defining OS characteristics. The three remaining IPv6 header fields, payload length, next header, and hop limit are used as features. The additional three features are in the link-local layer which contains the packet size, protocols used, and transport-layer protocol in the data-link frame as shown in Figure 2.

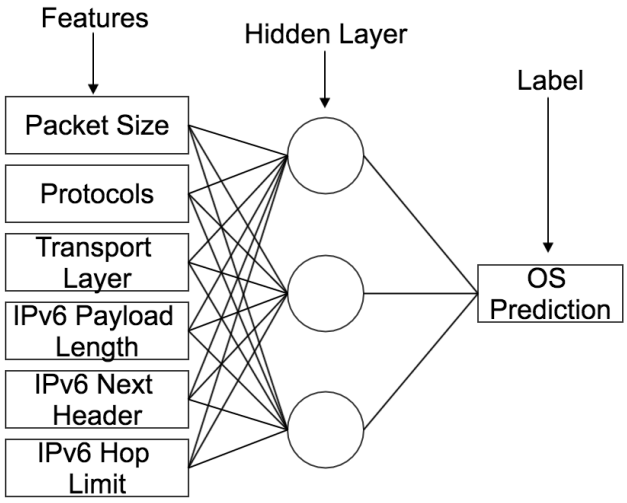

Figure 2. Layout of a neural network

3.3 Neural Network System

We used a Java neural network toolkit based on the free machine learning toolkit Waffles [5] to build an IPv6 packet classifier. The goal is to find a trained neural network layout for passive OS identification. The initial experiment consisted of training one neural network with a feed-forward backpropagation gradient descent using varying hidden units, momentum, and learning rates. An experiment consists of initializing the neural network weights to small random values, randomly re-ordering the data, splitting data to a training set and testing set, training the neural network configuration with the training set, and measuring accuracy of the OS prediction with the testing set. The varying hidden units are used to expand the expressiveness of the neural network to be able to represent complex models. The activation function of any layer used the hyperbolic tangent "tanh". Other activation functions were explored, such as identity, logistic, arctan, and rectified linear unit, but often gave much poorer results. Before the neural network can start training, the initial weights are set to small random values calculated using the Gaussian distribution. These random values, as well as the randomly selected training data, can influence and modify the weights of the neural network as it may get stuck in a local optimum. The process is repeated over a hundred experiments to ensure finding the global optima for best accuracy.

For each experiment, the data is loaded into matrices and rows are randomly swapped. Then, the data is split into a training set (80%) and a testing set (20%). Although several neural network layouts were explored, containing multiple layers with hidden units {1,2,3,4,5,6} and without hidden units, the average accuracy for a single neural network was approximately 65%. Accuracy is defined as the number of misclassified packets from the testing set over the total number of packets from the testing set. Upon a closer inspection of the misclassifications, the Linux packets were the only OS that would fall under a misclassification. It appeared that some setups were able to classify specific Linux packets over other setups that were better classifying other types of Linux packets. A voting system will take advantage of these various setups, so a collection of three neural networks are used together for classification. All neural networks train on the same training data, and validate on the same testing data. Instead of a majority rule, the voting system took a unanimous approach when classifying Windows packets. To classify as a Windows packet, there must be a consensus across all neural networks. If any or all neural networks predicted the packet OS as Linux, then the packet OS will be classified as Linux.

To pipeline the OS identification process, there is one rule in the process that will always classify a packet as a Windows OS if the protocols feature includes the Link-Local Multicast Name Resolution (LLMNR) protocol. LLMNR is included in all Windows versions back to Windows Vista. LLMNRD is a daemon implementing the LLMNR protocol that can only be supported on Linux to respond to name resolution queries sent by Windows client. However, this is an additional package that must be installed separately if desired. The dataset shows that only Windows clients contained the LLMNR protocol. This is the only rule which purpose is to demonstrate that there can be additional rules to further increase the accuracy of OS classification beyond the neural network prediction.

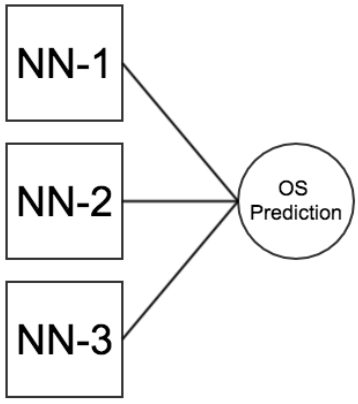

Figure 3. Neural Network System for Passive OS Fingerprinting Using IPv6

4. RESULTS

4.1 Experiment Results

The neural network system used for passive OS fingerprinting is shown in Figure 3. Three neural networks of varying configurations each contribute an OS prediction. The voting system rule requires a unanimous vote for a Windows OS prediction. Otherwise, the prediction will be for a Linux OS.

Table 2. OS Classification Accuracy

	Windows Accuracy	Linux Accuracy	Overall Accuracy
Average	100%	76%	86%
Best	100%	93%	96%
Worst	100%	2%	42%

As shown in Table 2, this voting system has no misclassifications for Windows machines despite the larger number of Linux packets used in training. With the neural network voting system, the average overall accuracy over a hundred experiments is approximately 86% with a 21% improvement from using one neural network at 65% accuracy. However, for any given Linux packet the average accuracy is 76%. In the best-case scenario, Linux accuracy increased to 93%, increasing the overall accuracy to 96%. While in the worst-case scenario, almost any Linux packet was classified incorrectly. This worst-case neural network configuration occurs rarely, which may be caused by poor random weight initialization.

Table 3. Accuracy of Neural Network System

Accuracy	Approximate Percentage (%)
Average (100 experiments)	86
Maximum	96
Median	95
Minimum	45
Standard Deviation	13

The average, maximum, median, minimum, and standard deviation of the accuracy are shown in Table 3. The average accuracy is taken from all the experiments. Although the average is 86%, the median is approximately 95% which shows that the accuracy data points have a skewed left distribution. The average is decreased due to a few outlier experiments that occurred either because of an unusual initialization of random weights or unusual distribution of training data during data randomization. These outliers occur 5%, or less, of the time. The weights for the highest accuracy setup are recorded for future OS classification without additional training.

4.2 Discussion

Table 4. OS Classification Accuracy with Extra Training

	Windows Accuracy	Linux Accuracy	Overall Accuracy
Average	100%	87%	92%
Best	100%	94%	96%
Worst	100%	4%	43%

We explored the impact on the Windows accuracy, Linux accuracy, and overall accuracy using additional training on the neural network system. For additional training, the training data set is reused in a feed-forward backpropagation gradient descent in order to explore local optimum that potentially results in more accurate classification. Our intuition is that the Linux classifier could be improved with more data as it may learn more characteristics for Linux identification. We considered above 90% overall accuracy a good experiment so in the event that the experiment achieved less than 90% accuracy the neural network system will do another iteration of training on the same training data, then measure accuracy with the same validation data. The average, best, and worst results are shown in Table 4. With this additional step, the average accuracy for Linux predictions increased to 87% with an improvement of 11% from 76%. This raised the overall accuracy from 86% to 92% with the extra training for select experiments. Furthermore, the maximum accuracy for Linux predictions is 94% with an increase of 1% from 93%, which did not notably change the overall accuracy of 96%. The worst accuracy for Linux predictions is 4% with an increase of 2% from 2% without the extra training. These results show an improved Linux classifier which may be due to fitting the neural network models with more accurate configurations, while still facing the possibility of evolving towards an over-fitted model.

Table 5. Accuracy of Neural Network System with Extra Training

Accuracy	Approximate Percentage (%)
Average (100 Experiments)	92
Maximum	96
Median	95
Minimum	43
Standard Deviation	10

The average, maximum, median, minimum, and standard deviation of the accuracy with extra training are shown in Table 5. The average accuracy with extra training is 92%, a 6% improvement from 86%. The maximum and median accuracies remained the same at 96% and 95%, respectively. The minimum accuracy decreased to 43% from 45%. One reason for this is that when the extra training occurred, occasionally the neural network system got worse with predictions, most likely due to overfitting. However, this did not occur often and provided more benefits as a whole increasing average accuracy. With improved accuracies, the standard deviation decreased to 10% with an improvement of 3% from 13%.

As shown in Figure 4, the left chart shows the distribution of experiment accuracy per experiment in intervals. Approximately 67% of neural network systems achieved an accuracy between 90.6% to 100%, followed by smaller distribution in the lower accuracy intervals with 4 instances within an accuracy of 71.8% to 81.2% and a cluster of 24 sets of neural networks with an accuracy in the range of 62.4% to 71.8%. There are a few remaining sets of neural networks on the lower end of accuracy ranging from 40.5% to 53.1%.

In comparison to the distribution of experiment accuracy with extra training, there is a major improvement of 92 sets of neural networks with an accuracy between 90.6% to 100%. This is a major improvement with an overall increase of 25% from 67%. The remaining interval clusters significantly decreased with only one set of neural networks with an accuracy between 71.8% to 81.2%, five sets of neural networks with an accuracy between 62.4% to 71.8%, and two worst-case sets of neural networks with an accuracy range of 40.5% to 53.1%. This distribution shift shows a significant improvement by using additional training on the neural networks in the event of a classification accuracy less than 90%.

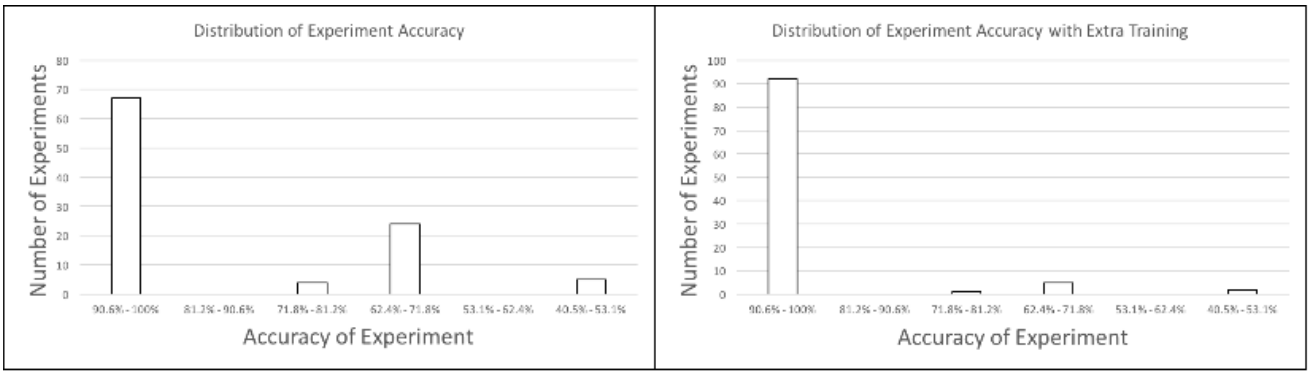

Figure 4. Distribution of Experiment Accuracy Comparison

5. FUTURE WORK

This work can be extended and improved. First, regularization techniques should be applied to prevent the common neural network problem of overfitting. In addition, a larger and more diverse data set should be collected to verify the accuracy in a production network having multiple versions of the operating systems. While traffic was gathered from twenty separate machines, the OS image for all the machines are the same for the Windows image and the same for the Linux image. IPv6 traffic from a variety of Windows and Linux versions needs to be gathered to determine if this technique can identify the OS and release versions. Finally, it is unknown if operating systems running as virtual machines on a host machine can be identified with the same level accuracy. The virtual interfaces may affect the IPv6 features.

6. CONCLUSIONS

In this paper, we develop and test a passive OS fingerprinting technique using three machine learning based classifiers and a voting scheme trained on IPv6 features found in the network layer and data link layer. Unlike some OS fingerprinting techniques, the method is passive, that is, it does not require sending packets to the machine. Instead, normal traffic is gathered by passively listening. In addition, this method is developed to use only IPv6 features for OS fingerprinting, not IPv4. The average accuracy of 100 sets of neural networks is 86%, with a maximum of accuracy of 96%.

7. REFERENCES

[1] Amante, S., Carpenter, B., Jiang, S., and J. Rajahalme. 2011. IPv6 Flow Label Specification. RFC 6437.

[2] David W. Richardson, Steven D. Gribble, and Tadayoshi Kohno. 2010. The limits of automatic OS fingerprint generation. In *Proceedings of the 3rd ACM workshop on Artificial intelligence and security (AISec '10)*. ACM, New York, NY, USA, 24-34. DOI=http://dx.doi.org/10.1145/1866423.1866430

[3] Durumeric, Z., Wustrow, E., and Halderman, J. A. 2014. An Internet-wide View of Internet-wide Scanning. In *USENIX Security Symposium (2014)*.

[4] Fyodor. The art of port scanning. Phrack 51, 7, September 1997. http://www.phrack.org/issues/51/11.html.

[5] Gashler, Michael S. Waffles: A machine learning toolkit. Journal of Machine Learning Research, 12:2383–2387, July 2011. ISSN 1532–4435. http://www.jmlr.org/papers/volume12/gashler11a/gashler11a.pdf.

[6] Infobidouille. Comparison of IPv4 and IPv6 headers. 2014.

[7] Lulseged Ayalew, Dietmar P. F. Möller, and Gerhard Reik. 2007. Using artificial neural networks (ANN) for real time flood forecasting, the Omo River case in southern Ethiopia. In *Proceedings of the 2007 Summer Computer Simulation Conference (SCSC '07)*. Society for Computer Simulation International, San Diego, CA, USA, , Article 19 , 7 pages.

[8] Narten, T., Nordmark, E., Simpson, W., and H. Soliman. 2007. Neighbor Discovery for IP version 6 (IPv6). RFC 4861.

[9] Octavio J. Salcedo Parra, Angela Patricia Rios, and Gustavo López Rubio. 2011. IPV6 and IPV4 QoS mechanisms. In *Proceedings of the 13th International Conference on Information Integration and Web-based Applications and Services (iiWAS '11)*. ACM, New York, NY, USA, 463-466. DOI=http://dx.doi.org/10.1145/2095536.2095631.

[10] Pawel Foremski, David Plonka, and Arthur Berger. 2016. Entropy/IP: Uncovering Structure in IPv6 Addresses. In *Proceedings of the 2016 Internet Measurement Conference (IMC '16)*. ACM, New York, NY, USA, 167-181. DOI: https://doi.org/10.1145/2987443.2987445.

[11] The Subterrain Security Group. 2000. Siphon Project. https://github.com/unmarshal/siphon.

[12] Wonje Choi, Karthi Duraisamy, Ryan Gary Kim, Janardhan Rao Doppa, Partha Pratim Pande, Radu Marculescu, and Diana Marculescu. 2016. Hybrid network-on-chip architectures for accelerating deep learning kernels on heterogeneous manycore platforms. In *Proceedings of the International Conference on Compilers, Architectures and Synthesis for Embedded Systems (CASES '16)*. ACM, New York, NY, USA, , Article 13 , 10 pages. DOI: https://doi.org/10.1145/2968455.2968510.

[13] Yogesh Singh and Pradeep Kumar. 2010. Application of feed-forward neural networks for software reliability prediction. SIGSOFT Softw. Eng. Notes 35, 5 (October 2010), 1-6. DOI=http://dx.doi.org/10.1145/1838687.1838709.

[14] Zalewski, M. 2012. p0f v3 (version 3.09b). http://lcamtuf.coredump.cx/p0f3/.