Big Data Analytics for Real-time DDoS Detection and Mitigation in SDN

Rahul Prabhu
CISE Dept.
University of Florida
rprabhu@ufl.edu

Sanil Sinai Borkar
CISE Dept.
University of Florida
sanilborkar@ufl.edu

Sile Hu
CISE Dept.
University of Florida
husile@ufl.edu

Umar Majeed
CISE Dept.
University of Florida
umarmajeed@ufl.edu

***Abstract*—Distributed Denial of Service (DDoS) is a major security threat on the Internet and the lack of real-time DDoS detection and mitigation tools can render servers vulnerable. In this paper, we propose a system to detect and counter DDoS for Software Defined Networks (SDN) that learns from historical DDoS attack data and deploys counter measures in real-time. The availability of centralized control and decoupling of the control and the data plane in SDN which allow for reprogramming routing decisions on the go make this possible. The accuracy of the proposed system exceeded 97% using a good mix of DDoS attack and normal network traffic dataset, and the latency added by the extra computation was found to be minimal. As SDN is gaining popularity, the system that we propose here can be used to detect and counter DDoS attacks, in real-time, that networks are vulnerable to.**

***Index Terms*—Software Defined Networks; SDN; Big Data Analytics; DDoS**

I. Introduction

Distributed denial of service (DDoS) is an attack to consume victims' resources such as bandwidth, memory and TCP connections to make the victims' services unavailable [**?**]. Attackers control malicious botnets or utilize defects or loopholes in Internet protocols to launch attacks. Utilizing valid services over the Internet and leveraging amplifying effects of certain protocols, attackers can quickly exhaust victims' bandwidth. Attacking techniques include TCP SYN flooding, ICMP flooding, reflective DNS/NTP/SNMP attacking etc. [**?**]. Attacks are typically short bursts of heavy traffic which makes it particularly difficult to detect and defend against such attacks. The lack of a centralized view of a network adds to this already serious problem.

Software Defined Networking (SDN) is an approach to networking where the control plane and the data plane are decoupled [**?**]. The ability of SDN to filter network packets with the help of the controller, when they enter a network, gives it the ability to implement DDoS detection and countering mechanisms. SDN switches can act as a firewall at the edge of a network thus providing security to the end hosts. They also have the capability to dynamically respond to application requirements, optimize the utilization of the network without compromising the QoS which allow it to implement extra defense mechanisms without any significant adverse effects to network performance.

In this paper, we describe a solution to the problem of real-time detection and mitigation of DDoS attacks in SDNs. The fact that the SDN controller has a centralized view of the network and can reprogram network switches on the go makes such a solution possible. The following are some of the features of our solution:

1) SDN-enabled DDoS detection
2) Use of big data analytics to enable time detection of DDoS attacks
3) Use of SDNs ability to reprogram networks on the go for real-time mitigation of DDoS attacks

In traditional networks, every switch decides where an incoming packet has to be routed. In a large network, this translates to lot of entities acting independently which make collective decision making very hard. In SDNs, the routing decisions are made by the controller for each switch. This gives the controller a centralized view of the network and gives it the ability to make collective decisions for the network as a whole [**?**]. We make use of this unique feature of SDNs to build our system.

When a packet arrives at a switch in a SDN, the switch communicates to the controller about the incoming packet and requests the controller for the action to be performed. The controller then makes the decision on where the packet must be sent and relays this information back to the switch. Here, the SDN controller can do some computation to detect if a packet is a malicious attack

packet or a benign packet, and based on this output, instruct the switch to either drop the packet or forward it.

In large networks, the SDN controller may have to deal with thousands of switches and hundreds of thousands of packets at any given point of time. This makes the computation to determine whether a packet is malicious or not infeasible to do in real-time. To address this, we move the computation to a DDoS Detection Engine (DDE) which runs on a cluster of commodity computers. The controller extracts certain features from incoming packets and sends it in batches to the DDE which does the necessary computation for determining if any packet is malicious or not and sends back the decision to the controller. Now, the controller is only left with the task of making the decision of dropping or forwarding the packet, and sending these instructions to the appropriate switch.

The DDE takes in historical DDoS attack data and builds a classification model which classifies an incoming packet as malicious or benign. The process of building a classification model is mostly done offline as it takes in large amounts of data and is difficult to build in real time. However, since this is done very infrequently compared to the classification task itself, it does not affect the performance of the system. The classification task is carried out in real time and is what enables our solution to respond to threats on the fly.

In this paper, we use an Apache Spark cluster for building the DDE as it gives near real time performance and provides support for streaming data. We use decision trees for the classification tasks because of the ease it offers to classify categorical attributes. Apache Spark provides the Machine Learning library (MLlib) which closely integrates with the Spark framework. Therefore, we use the Decision Tree implementation provided by MLlib.

One of the major assumptions that we have considered is that the controller would not be under attack in any circumstances. This is a strong assumption to make when dealing with a DDoS-related area. But this system can be thought of as a stepping stone to building a more robust and secure SDN DDoS attack countering system.

Secondly, the DDE will be running on an Amazon Web Services Elastic Cloud 2 (AWS EC2) cluster comprising of a master node and a small number of slaves. This processing power, although small, will reduce the latency to some extent but can be further reduced by increasing the cluster size.

The rest of the paper is organized as follows. Section II discusses the related work. Section III discusses the System Architecture. Section IV discusses implementation details. Section V discusses the experiments carried and presents the results. Sections VII and VI present our conclusions and some directions for future work in this field, respectively.

II. Related Work

Traditionally, various techniques exist to defend DDoS attacks from independent routers' point of view, including access control lists, protocol hardening, packet filtering based on IP addresses, TCP and UDP port numbers, and packet lengths and content matching [**?**]. These mechanisms are effective once the attacking mechanisms are well understood, but suffers from: (1) lack of real-time response; (2) need to be applied on per-router basis.

SDN provides the ability of centralized control and real-time response over a set of routers, and there are a few vendors working on integrating DDoS mitigating features into the SDN controllers, such as Brocade [**?**] and Radware [**?**]. These applications provide attack detection and control over a whole SDN, but the detection and access rules are still largely packet filtering based.

sFlow-RT by InMon implements the sFlow protocol which acts as a link between the network devices and the SDN flow controller [**?**]. They use the sFlow analytics engine for DDoS mitigation. The SDN flow controller communicates with the DDoS engine which instructs the SDN controller to drop packets that are malicious.

Lee and Lee worked on detecting DDoS attacks at scale by using a Hadoop cluster for computation [**?**]. However, they were not able to tackle this in real-time as Hadoop is a batch processing framework.

Gavrilis et al. [**?**] designed a DDoS attack detection system for Radial-basis-function Neural Netowrks (RBF-NN).

A PHD student from Carleton University came up with a DDoS attack detection scheme in SDNs using the notion of entropy of window size and thresholds [**?**]. His study, however, was able to only detect an attack and there were no mitigation procedures proposed or in place.

For training the data and classification purposes, a good feature set is required which can be derived using various methods. The most promising method was the one specified by Ganapathy et al. [**?**] wherein they used an intelligent rule-based attribute selection algorithm to find out the feature set. Moreover, Seo et al. [**?**] presented a clustering-based method to find out the feature set to train the model.

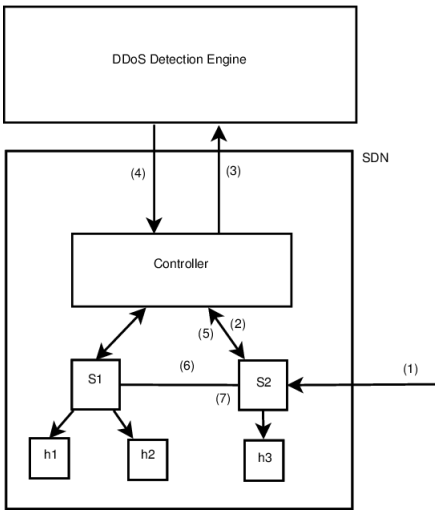

Fig. 1: Architecture

III. Architecture

The architecture of our system shown in Figure **??** comprises of the SDN network and the DDoS Detection Engine (DDE). An incoming packet reaches one of the switches of the SDN (1). This switch is called a boundary switch and serves as the entry point to the SDN. Most of the malicious packet filtering happen in such boundary switches thus preventing attacks from impacting other nodes in the network. This switch requests the controller for directions on what to do with the packet (2). The controller extracts packet information from all the packets it gets and streams it to the DDoS Detection Engine (3). The DDE receives this stream of packet information and evaluates each packet against the classification model built and gives a decision whether a particular packet is malicious or not (4). It relays this information back to the controller. Based on the decision of the DDE, the controller sends appropriate flow commands to the appropriate switch (5). Depending on the flow command received, the switch either forwards the packet enroute to its destination (6) or drops it (7).

The following components represent the building blocks of our proposed system:

DDoS Detection Engine (DDE) - The DDE is the main component of the entire system as it performs the most difficult part of DDoS attack detection. It is an external cluster of commodity machines running Apache Spark that can only communicate with the OpenFlow SDN controller. The DDE accepts the network packets' information that was sent by the OpenFlow controller. It feeds this information to the already trained classification "Decision Tree" model and relays the result back to the controller. The DDE performs this computation for every packet information that the controller sends it. Since the DDE has to do high magnitudes of computation, we have deployed the DDE on a cluster of machines in EC2. For the computation of such seemingly large magnitudes of data, we have used Apache Spark as it deals with in-memory computing which is nearly 10 times faster than its counterpart Hadoop[**?**]. In addition to this advantage of faster data processing, Apache Spark has support for streaming data which comes as an advantage to us as we deal in data streaming across network from the controller to the DDE.

Another reason to chose Apache Spark was the inclusion of Machine Learning libraries (MLlib) within it. MLlib provides support for various machine learning algorithms that can be made use of directly by using the APIs provided by MLlib [**?**].

Software Defined Networks (SDN) - SDN is the area of interest for our research. We are building our system in simulated SDNs. The controller plays a crucial role in the DDoS attack and mitigation process. It acts as a link between the DDE and rest of the network. The switch sends a request to the controller if it encounters a new incoming network packet. The controller, in our case, sends this information to the DDE and receives the response from the DDE. Depending on this response, it instructs the switch to either drop or forward the packet. This is done by the controller by sending add and/or modify flow rules to the switch in order to insert into or update the switch's flow tables. The controller's ability of reprogramming the switches comes to an advantage here to implement such a system architecture.

Communication Modules - The individual modules are interfaced in a way to reduce latency and enhance performance. We have a daemon running on the controller that reads the requests sent by the switch, extracts the required packet data from it and relays the same to the DDE which is located externally on a cluster of machines. On computation, the DDE sends the results back to the controller which reads it, and fires a flow rule back to the switch. The switch then takes the appropriate action as instructed by the controller in the form of the flow rule(s).

IV. System Design and Implementation

For the purpose of developing our system, we built an SDN network using Mininet [**?**]. Our experimental setup is as shown in Figure **??**.

Our environmental setup consists of the following components:

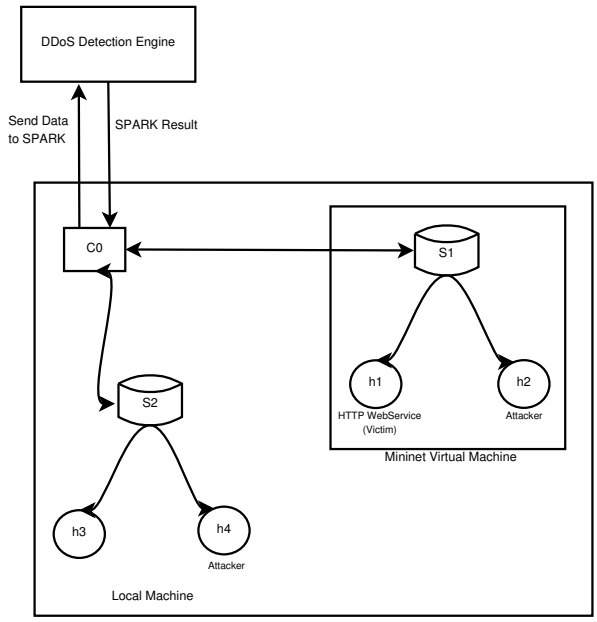

Fig. 2: Experimental Setup

- The host (local) machine hosts the controller (c0), two hosts (h3 and h4) and one OpenFlow-enabled switch (s2) of a SDN.
- A guest Mininet VM running on the host machine contains two hosts (h1 and h2) and an OpenFlow-enabled switch (s1). This switch *s1* is connected to the remote controller *c0* hosted by the local machine.
- The DDE is setup on an AWS EC2 cluster.

This setup was implemented on a local (host) machine and a Mininet VM that was hosted on this local machine, as can be seen in Figure **??**. The local machine contained a SDN network using Mininet with one switch, two hosts and one controller. The Mininet VM hosted another SDN that consisted of two hosts and one switch. This switch was connected to the remote controller present on the local machine. This can be seen in Figure **??** wherein the controller was present at the default port of 6633 on the local machine (127.0.0.1), and the switch within the Mininet VM was connected to this controller as can be seen from the right-hand side screenshot in the Figure **??** (RemoteController c0: 192.168.56.1:6633 pid=1115).

In this setup, host *h1* was setup to run a python HTTP web server which acted as our victim. The host *h2* acted as the attacker which carried out TCP SYN flooding attack on *h1*. The switch *s1* communicated with the remote controller *c0* to get the flow-rules at the switch. The data received by the controller from the switch was then sent to the DDE running on the AWS EC2 cluster. The DDE is the heart of our project makes a decision from the input it receives from the controller making use of Apache Spark and the MLlib present within it.

Once the controller received this decision back from EC2, it relayed this information to the switch in the form of add/modify flow rules. The switch then took the necessary action(s) as per the instructions given by the controller.

The first time when the host *h2* requested a webpage from the web server *h1*, this request arrived at the switch *s1*. Since the switch *s1* did not have a flow rule corresponding to this flow in its flow tables, it sent this data to the (remote) controller *c0* seeking for an answer. The controller got this request data from the switch and relayed it to the DDE running on the EC2 cluster.

The main computation took place at the DDE on EC2 cluster. Depending on the feature set selected, the DDE predicted if the newly received packet from the controller was a malicious packet or a benign packet. Once this decision was taken, it notified the controller about it and sent this information to the controller. The controller then accepted this decision and instructed the switch by adding and/or modifying flow-rule(s) in its switch table. The switch now aware of what has to be done, either forwards/drops the packet as per the flow-rules set by the controller.

While designing our system, we selected Apache Spark which is a faster big data processing framework than Hadoop's MapReduce paradigm, and performs up to hundred times faster than Hadoop. Spark is an open source data analytics, cluster computing framework [**?**]. Spark supports in-memory clustering as opposed to involving disk operations within Map/Reduce processes [**?**]. For training a model and future predictions, we made use of the machine learning algorithms that are provided by Spark.

In addition to all the advantages and pros mentioned above, Spark has good support for AWS EC2 implementations. Spark comes with EC2 scripts that can be used to directly set up clusters along with certain command-line arguments [**?**]. It gives us the flexibility to even initiate a cluster on EC2 with a specific version of Spark as per what is input on the command-line.

To build up the DDE, we trained it with sample data consisting of a good mix of normal and attack data. Training the model made use of the "Decision Tree" machine learning algorithm, and this algorithm could also be used effectively for DDoS filtering [**?**]. We made use of the MLlib, which is an in-built machine learning library framework provided by Apache Spark, to train

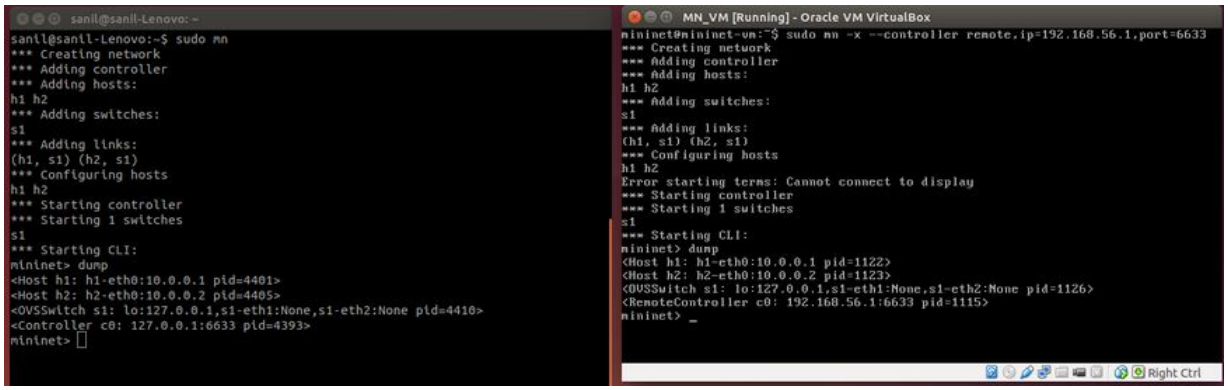

Fig. 3: SDN Setup

a Decision Tree model. Since we were dealing with classifying each incoming network packet as malicious (having a value of 1) or benign (having a value of 0), the Decision Tree model was the most promising implementation. For every incoming network packet that was transmitted by the controller, the DDE fed the features information present in each packet to this already trained model and relayed this decision back to the controller.

In case of a malicious packet being encountered, the DDE sent a decision value of 1 (malicious). In cases of benign packets, it sent a decision value of 0 (benign). Depending on whether a 1 or 0 was received, the controller then instructed the switch to drop the packet (in case of the former) or forward the packet (in case of the latter).

One of the major assumptions that we have considered is that the controller would not be under attack in any circumstances. This is a strong assumption to make when dealing with a DDoS-related area. But this system can be thought of as a stepping stone to building a more robust and secure SDN DDoS attack countering system.

V. Experiments and Results

The first and foremost thing we needed to decide is what information should be fed to the DDE from the controller. Since a normal TCP network packet contains numerous information

We carried out a number of experiments on the environment setup described earlier. Due to the University network security fair policies and firewall blocking, we were not able to carry out an actual massive DDoS attack. However, we have used the attack dataset provided by the Center for Applied Internet Data Analysis (CAIDA) that contains TCP SYN flooding attack data [**?**].

Web Server	Data Hosted	Attackers	Requests
apache2	10kb	1 HULK	100k
apache2	100mb	30 HULKs	10 million
python	100mb	30 HULKs	1 million

TABLE I: DDoS Attack Using HULK

For normal data generation, we used HTTP Unbearable Load King (HULK) which is a python program to generate random HTTP requests targeted for an HTTP web server [**?**]. HULK is a baseline HTTP request generation tool that is used to launch low capacity Denial of Service (DoS) attacks for research purposes.

We tested out HULK's ability to launch a DoS attack by hosting some web servers and trying to attack them. The results of this experimental setup is tabulated in Table I. During the experiments, we observed that HULK was not able to bring down powerful servers such as Apache2 even with heavy loads but was able to bring down simple HTTP servers we created using python. These results provided us a good tool for generating random data.

We ran the HULK program on the host *h2* in our Mininet VM network to generate random HTTP web requests to the web server hosted at *h1*. However, this program was used to only generate normal data in smaller magnitudes and no attack data of any sort.

Apache Spark and MLlib provide us with a number of machine learning algorithms viz. Decision Tree, Naive Bayes, Support Vector Machine (SVM) and . To come up with a good learning algorithm, we tested our datasets against some of the candidate machine learning algorithms provided by MLlib, the results of which are tabulated in Table II.

As can be seen from the Table II, SVM provides an error rate of more than 10% as compared to Decision

Algorithm	Feature Set Size	Dataset Records	Time for Training (s)	Time for Prediction (s)	Error
Decision Tree	43	36,646	0.0002520084	0.0040941238	2.46%
SVM	43	36,646	0.0002040863	0.0044519901	10.75%
Naive Bayes	43	36,646	0.0005002022	0.0016739368	8.84%
Logistic Regression	43	36,646	0.0003650188	0.0011081696	9.04%

TABLE II: Performance of Various MLlib Machine Learning Algorithms

Tree which exhibits an error of less than 3%. The time for training the model seems to be more than the time taken for prediction because the latter is for predicting about 36K records. The time for prediction per input record is thus obtained by dividing the given prediction times by 36K to yield time in the order of microseconds. The prediction time for Decision Tree for the given features set and dataset is 0.114 microseconds.

One more thing to note is that Decision Tree exhibits a very low error rate with the same feature set as compared to the others. On the other hand, Logistic Regression takes an even lesser time for prediction at the cost of a high error rate. There is a trade-off here and as per the requirements of our system, we would want a higher accuracy (lower error rate) as opposed to the time required for prediction.

The accuracy results are plotted in Figure **accuracy**. As can be seen from this figure and the argument specified above, the Decision Tree machine provides a high accuracy of more than 97% as compared to others which are lower than this value. Although these algorithms were run on datasets of size in the range of 36K, it works with almost the same accuracy with a Root Mean Square (RMS) error of less than 3%.

From the above results, we selected Decision Tree as the machine learning algorithm for our system due to its high accuracy and very low error rates even on huge datasets. Its ability to work with categorical attributes added to its advantages over others.

We monitored and analysed the decisions that were taken by the DDE as against the actual interpretation of data. Out of all the experiments that we conducted, we achieved a high degree of accuracy with minimal error rate (more than 97% accuracy with an error rate of less than 3%) in accordance to the Decision Tree model performance discussed above.

We apply the following defense mechanism. After the packet has been classified, we either forwarded the packet to its destination or blocked the malicious host from sending any further (attack) packets based on the decision from the DDE. OVS-OFCTL commands were used to achieve this. The following commands were sent from the controller to the switch based on the classification results.

- To enable forwarding in the switch: *ovs-ofctl add-flow s1 priority=10,action=normal*
- To block traffic from a malicious host *ovs-ofctl add-flow s1 priority=11,dl_type=0x0800,nw_src=10.0.0.1,action=drop*
- To restore the traffic back. *ovs-ofctl –strict del-flows s1 priority=11,dl_type=0x0800,nw_src=10.0.0.1*

Adding a rule with a higher priority trumps the previous rule and the packets from the host with the specific IP are dropped. To restore the traffic, we simply deleted the latest flow rule (to drop the packets) added at the switch.

Although the results were obtained as expected, the actual performance did not match the expected performance. This was due to the fact of many underlying limitations and external factors that were beyond our control and influence. During our research, there were many obstacles due to which an efficient implementation of the system was not feasible viz. the University network security “fair” policies and the use of virtual machines. Although EC2 provides us with a platform to take advantage of cluster infrastructure, due to underlying network bottleneck, the data transmission does not happen at high speeds leading to some latency in the data transmission. Moreover, the use of a smaller magnitude of processing power at the EC2 cluster also degraded the performance to some extent. These limitations of the underlying infrastructure and University policies made our results deviate from the expected high real-time performance. However, we did achieve a near real-time response and performance. These results will be further enhanced and the latency further reduced if this system implementation could be carried out in an actual hardware/software setup.

VI. Future Work

As part of the future research possible, there are a few areas where improvements can be made. The major assumption regarding the security of the SDN controller can be re-assessed. The controller can, just as the other network components, be a target of an attack. Since this is where the intelligence of the SDN is concentrated, it is

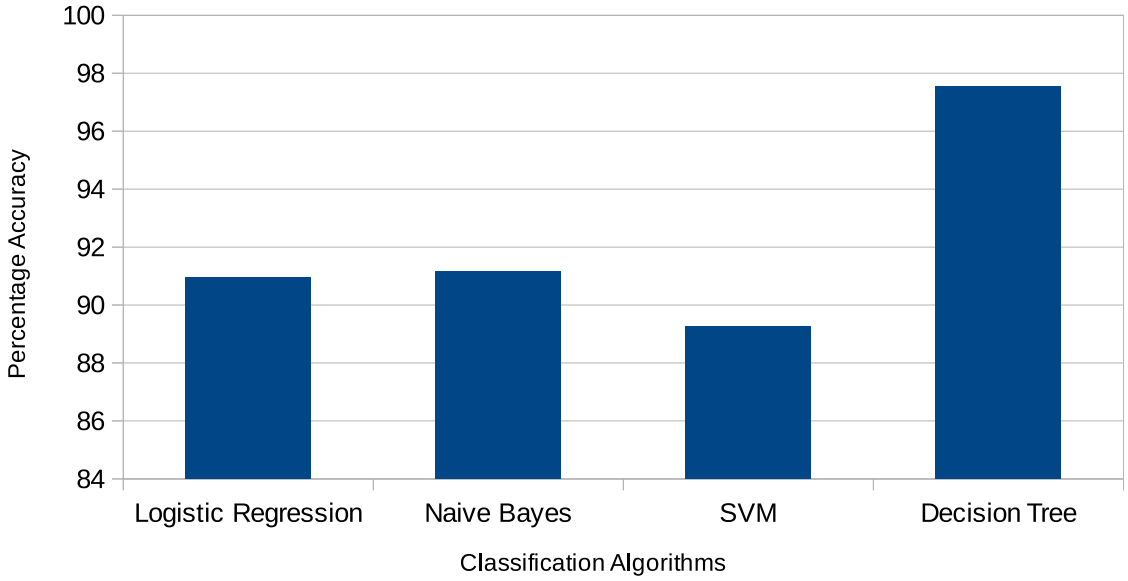

Fig. 4: Classification Accuracy of MLlib Machine Learning Algorithms

a very lucrative part to be an attack target. The robustness and security of the system can be further improved by making the controller resistant to any type of DoS and DDoS attacks.

In addition to this, although the system that we have proposed is able to detect different types of DDoS and DoS attacks, we have performed only TCP SYN flooding attacks on the SDN. This system could be further tested to detect and counter DDoS attacks from all possible DDoS/DoS means and sources viz. ICMP flooding, reflective DNS/NTP/SNMP attacking. This can be used to create a comprehensive dataset that could be used to analyse and/or evaluate such a system in greater depths.

Moreover, we have used a small cluster containing a few nodes on EC2 which induces significant latency in the data transfer and processing power. This can be improvised by increasing the size of the EC2 cluster or building an actual cluster in hardware. Though the latter options seems difficult, it would certainly reduce the latency thereby bringing the performance in par with real-time response systems.

VII. Conclusion

Distributed Denial of Service (DDoS) is a major security threat on the Internet and the lack of real-time DDoS detection and mitigation tools can render servers vulnerable. To address this ever-growing serious network security issue, we have proposed a system to detect and counter DDoS in Software Defined Networks (SDN) in real-time. The advantages of centralized control and decoupling of the control and the data plane in SDN makes the real-time aspect of the computation possible.

We set up a SDN network communicating with a remote controller, and also a DDoS Detection Engine (DDE) on a cluster of commodity machines. These machines were made to run Apache Spark as the latter is hundred times faster than its counterpart Hadoop, and also supports data streaming. Apache Spark along with the in-built Machine Learning library, we trained our model from historical DDoS attack data and normal traffic data. After this implementation, the DDE was able to detect and counter DDoS attacks in near real-time scenarios.

However, the network setup that we used did not perform exceptionally well due to the additional latency induced by the virtual machine setup and limitations of the University network security policies. Even though in the presence of these limitations, the accuracy of our proposed system was found to be more than 97% using a good mix of the attack data along with the normal data. The results were obtained within a reasonable amount of time with an RMS error rate of less than 3% on an average.

VIII. Acknowledgement

We are highly indebted to Dr. Andy Li, the Department of Computer Science & Engineering, and the Department of Electrical & Computer Engineering at the University of Florida for providing us with the opportunity, guidance and resources without which this research project would not be possible. We would like to extend our gratitude to the Center for Applied Internet Data Analysis (CAIDA) for providing us with the DDoS attack dataset. We are also grateful to all our friends and colleagues who have assisted and motivated us throughout the duration of our research, and provided constructive feedback on our work.

References

[1] J. Mirkovic and P. Reiher, "A Taxonomy of DDoS Attack and DDoS Defense Mechanisms," *SIGCOMM Comput. Commun. Rev.*, vol. 34, no. 2, pp. 39–53, Apr. 2004. [Online]. Available: http://doi.acm.org/10.1145/997150.997156

[2] C. Rossow, "Amplification Hell: Revisiting Network Protocols for DDoS Abuse." *Network and Distributed System Security Symposium, NDSS 2014*, 2014.

[3] "Software-Defined Networking: The New Norm for Networks," https://www.opennetworking.org/images/stories/downloads/sdn-resources/white-papers/wp-sdn-newnorm.pdf.

[4] P. Morreale and J. Anderson, *Software Defined Networking: Design and Deployment*. Taylor & Francis, 2014. [Online]. Available: http://books.google.com/books?id=4RUeBQAAQBAJ

[5] "How to defend against DDoS attacks," http://www.computerworld.com/article/2564424/security0/how-to-defend-against-ddos-attacks.html, accessed: 2014-09-15.

[6] "Real Time SDN and NFV Analytics for DDos Mitigation," http://techfieldday.com/video/real-time-sdn-and-nfv-analytics-for-ddos-mitigation, accessed: 2014-09-15.

[7] "DefenseFlow: The SDN Application that Programs Networks for DoS Security," http://www.radware.com/Products/DefenseFlow/, accessed: 2014-09-15.

[8] "InMon: sFlow-RT," http://www.inmon.com/products/sFlow-RT.php, accessed: 2014-09-15.

[9] Y. Lee and Y. Lee, "Detecting DDoS Attacks with Hadoop," in *Proceedings of The ACM CoNEXT Student Workshop*, ser. CoNEXT '11 Student. New York, NY, USA: ACM, 2011, pp. 7:1–7:2. [Online]. Available: http://doi.acm.org/10.1145/2079327.2079334

[10] D. Gavrilis and E. Dermatas, "Real-time detection of distributed denial-of-service attacks using RBF networks and statistical features," *Computer Networks*, vol. 48, no. 2, pp. 235–245, 2005.

[11] S. M. Mousavi, "Early Detection of DDoS Attacks in Software Defined Networks Controller," Ph.D. dissertation, Carleton University, 2014.

[12] S. Ganapathy, K. Kulothungan, S. Muthurajkumar, M. Vijayalakshmi, P. Yogesh, and A. Kannan, "Intelligent feature selection and classification techniques for intrusion detection in networks: a survey," *EURASIP Journal on Wireless Communications and Networking*, vol. 2013, no. 1, 2013. [Online]. Available: http://dx.doi.org/10.1186/1687-1499-2013-271

[13] J. Seo, J. Kim, J. Moon, B. J. Kang, and E. G. Im, "Clustering-based Feature Selection for Internet Attack Defense," *International Journal of Future Generation Communication and Networking*, vol. 1, no. 1, pp. 91–98, 2008.

[14] "Apache Spark - Lightening Fast Cluster Computing," https://spark.apache.org/, accessed: 2014-09-15.

[15] "MLlib Apache Spark," https://spark.apache.org/mllib/, accessed: 2014-10-13.

[16] M. Zaharia, M. Chowdhury, M. J. Franklin, S. Shenker, and I. Stoica, "Spark: Cluster Computing with Working Sets," in *Proceedings of the 2Nd USENIX Conference on Hot Topics in Cloud Computing*, ser. HotCloud'10. Berkeley, CA, USA: USENIX Association, 2010, pp. 10–10. [Online]. Available: http://dl.acm.org/citation.cfm?id=1863103.1863113

[17] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters," *Commun. ACM*, vol. 51, no. 1, pp. 107–113, Jan. 2008. [Online]. Available: http://doi.acm.org/10.1145/1327452.1327492

[18] "Running Spark on EC2," http://people.apache.org/ andrewor14/spark-1.1.1-rc1-docs/ec2-scripts.html, accessed: 2014-11-08.

[19] "Applying Machine Learning for DDoS Filtering," http://stats.stackexchange.com/questions/23488/applying-machine-learning-for-ddos-filtering, accessed: 2014-10-23.

[20] "Center for Applied Internet Data Analysis," http://www.caida.org/data/, accessed: 2014-11-11.

[21] "HULK, Web Server DoS Tool," http://www.sectorix.com/2012/05/17/hulk-web-server-dos-tool/, accessed: 2014-09-03.