Detecting and Mitigating DDoS Attack in SDN using POX

Donghun Kim

June 2020

1 Introduction

Software Defined Network (SDN) divides network equipment (router) into Control Plane and Data Plane.[2] The Control plane functions as your 'brain' and decides how to transfer the data while the Data Plane works as 'hands and feet' that forward the data. Before the emergence of SDN, the router had to calculate the route and forward the data through the resulted route. This combined architecture was inefficient in a way that the complexity and cost of the controller increase with the development of data transferring technology. Thus, the SDN brings a number of benefits to the IT community by increasing flexibility and speed and reducing the costs of network implementation. Although the separation brings considerable benefits, it makes the network vulnerable to Distributed Denial of Service (DDoS) attacks at the same time. In order to deeply understand the vulnerability, it is worth drawing attention to the communication between switches and the controller. Switches in SDN do not process the data but simply tries to forward the data in regard to forwarding tables. If a switch receives packets that cannot be forwarded according to the table, it sends 'packet-in' to the controller to ask where to forward.[2] The controller then decides and tells what to do with the packet. The major vulnerability occurs in this context because if the connection between the switches and the controller is lost, it will induce the network to malfunction. DDoS attacks are conducted with numerous packets being intentionally sent to a host in a network. When the switch receives a number of malicious packets by the attacker, it will render them to the controller. Subsequently, the flooding packets and traffic will hinder other connections. Thus, DDoS attacks can easily impede the connection between a controller and its switches. The increasing number of IoT devices puts the network in more frequent threats from DDoS attacks, and thus an efficient and robust solution is needed. However, since the attacker tries to imitate the traffic that looks like the real data as best as possible, it is hard to detect them in an early stage with the human eye or a simple program. Therefore, with the aid of Artificial Intelligence such as Deep Neural Network (DNN), we will build a system that can detect DDoS attacks and hence successfully defend such attacks. A networking software platform in python, POX, will be used as Controller in SDN, and the DNN model (those may include ‘Keras’ API for deep learning framework) will be implemented into the controller to detect the malicious attack. The input data for the DNN model will be the ‘packet-in’ of which the switch does not have the corresponding routing information in its forwarding table. Therefore, the model will evaluate several data in the packet-in as parameters, such as source IP address, source port number, destination IP address and ports number, and others. To obtain a solid and accurate DNN model, not only do we need enough real data that contains both malicious and benign traffic but also need to select the parameters having high correlation that can lead to accurate detection. Thus, excluding several parameters that have a lower correlation below a certain threshold will let us achieve the desired model. The threshold will be precisely set after sufficient conduction of experiments. After we build a reasonably accurate DNN model in the POX controller, we will modify the POX controller so that if the model judges an input pack-in as malicious, it makes switches update the forwarding table to drop such traffics. In this solution, the experimental process is the most critical factor in the project to achieve the desired goal. Determining the number of neurons, the number of layers, and activation functions to apply maybe the most crucial part of the final project.

2 DNN Model

In this project, Bidirectional LSTM model is used to detect malicious DDoS attack since its feed-forward neural network can selectively remember or forget pattern for long duration of times.[1] Not only does it fit for continuously changing source address in attack but it also can deal with long duration data, overcoming the limitation of common RNN models. LSTM has cell-state in its hidden layer which functions as conveyor belt. Consequently, its gradient in cells can be flowed with long period of time. The following is the formula for LSTM cell

$$\begin{aligned}
f_t &= \sigma(W_{xh\_f}x_t + W_{hh\_f}h_{t-1} + b_{h\_f}) \\
i_t &= \sigma(W_{xh\_i}x_t + W_{hh\_i}h_{t-1} + b_{h\_i}) \\
o_t &= \sigma(W_{xh\_o}x_t + W_{hh\_o}h_{t-1} + b_{h\_o}) \\
g_t &= \tanh(W_{xh\_g}x_t + W_{hh\_g}h_{t-1} + b_{h\_g}) \\
c_t &= f_t \odot c_{t-1} + i_t \odot g_t \\
h_t &= o_t \odot \tanh(c_t)
\end{aligned}$$

Figure 1: LSTM Formula

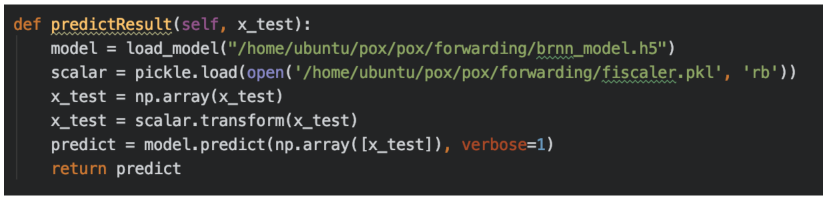
the ’f’ refers the gate to forget about past data. the forget gate will result in value by applying sigmoid function on ht-1 and xt. In this context, the sigmoid function will disregard the past information if the value is 0, and remember it if the value is 1. The ’i’ refers the gate to remember the current state. Similarly, the cells receive ht-1 and x and apply sigmoid function on to it. In order to scale the data to be fed into the model, I used the StandardScaler which automatically scale the value between -1 and 1 with the calculated standard deviation and mean value.[1]

3 Environment

For the experiment, the virtual environment for python 2.7 and 3.4 was required. I used an Virtual Machine with SDN, POX, OpenvSwitch and Openflow. To provide SDN environment, mininet is used which supports openflow protocol allowing users to freely construct software design networks. On Ubuntu 14.04, installing Pycharm will allow you to run both Keras module and POX source code which runs in Python 3.4 and 2.7 respectively in virtual environment.

4 DDoS Attack Detection

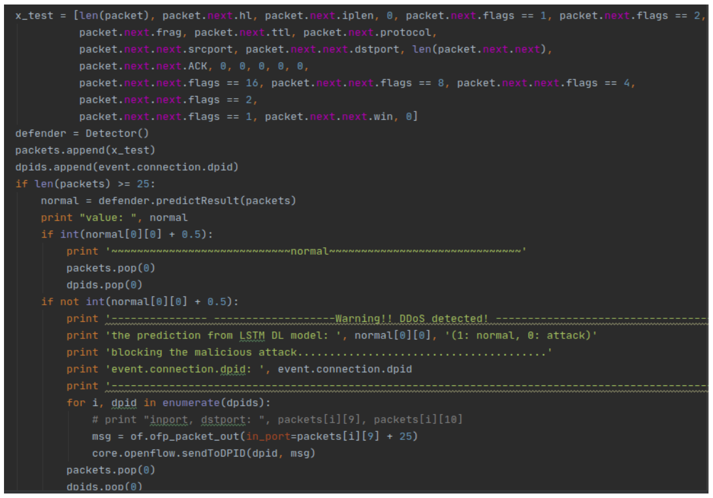
In pox, the 'l3_learning' is modified in order to detect and mitigate the DDoS attack.[3] The l3_learning works in a reactive mode and it learns the corresspondence between IP and MAC addresses to use this knowledge to implement the rule that replaces a destination MAC address. The attack is performed with flooding TCP SYN packets with random sources and ports to specific destination address. Since the switch does not calculate the routing, it will 'packet-in' to the controller to ask the routing information and thus, the controller will be over-consumed with a number of new 'packet-in' from unknown sources. 'hping3' allows you to perform such TCP-SYN flooding attack, with the command 'hping3 10.0.0.1 -c 10000 –flood -rand-source -w 64 -p 0 -S -d 120' while -c refers the number of packets being sent, –flood to flood, -rand-source to be flowed from random sources, -w as window size, -p as port number, -S as SYN, and -d as data size. For the DDoS data, both the real-world data and hping3 attack data captured by WireShark are used to train the DNN model. Through the POX controller, the various packet information from hping3 attack can be extracted. In the other research, the source IP and destination IP addresses played significant role in detecting attack; the Machine Learning detecting model with entropy value comes with calculating the variance of source addresses since the DDoS attack generates a number of random sources. However, in Bidirectional LSTM model,[4] the random source IP resulted in confusion in DNN model, making the accuracy fall to 0.54 percents. The continuously changing source IP cannot be fit into the LSTM model, since the model keeps remember the changing over a long period of time, and the series of different random source addresses does not give a clear distinction between the normal traffic and its destination addresses. Consequently, through several experimental result, I decided to exclude the source and destination IP addresses to train the model and analysis the packet data. Instead, the other specific information about the packet are inserted: frame length, header length, address's length, and various flags data from IP and TCP header. The input for the DNN model requires (25, 25) shape to detect the DDoS with high accuracy. Therefore, the array of packet information is gathered with length of 25, and it is sent to the model to be predict. Subsequently, the new packet is added to the array, popping out the first element and thereby maintaining the required shape by shifting the element to the left.



```python
def predictResult(self, x_test):
    model = load_model("/home/ubuntu/pox/pox/forwarding/brnn_model.h5")
    scalar = pickle.load(open('/home/ubuntu/pox/pox/forwarding/fiscaler.pkl', 'rb'))
    x_test = np.array(x_test)
    x_test = scalar.transform(x_test)
    predict = model.predict(np.array([x_test]), verbose=1)
    return predict
```

Figure 2: Predict Method

The above code shows how the model is loaded from pre-trained file and how a packet data is fed into the model to predict the result.



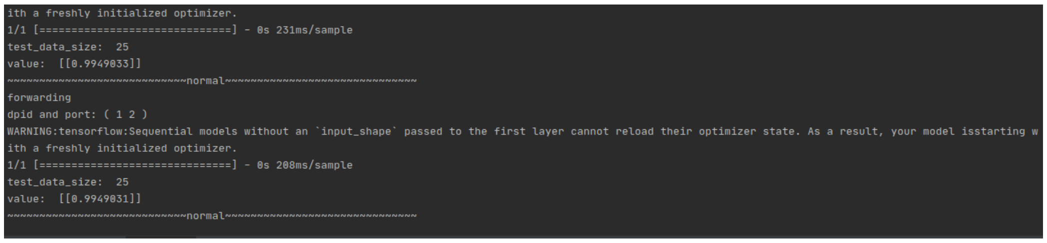
```python
x_test = [len(packet), packet.next.hl, packet.next.iplen, 0, packet.next.flags == 1, packet.next.flags == 2,
          packet.next.frag, packet.next.ttl, packet.next.protocol,
          packet.next.next.srcport, packet.next.next.dstport, len(packet.next.next),
          packet.next.next.ACK, 0, 0, 0, 0, 0,
          packet.next.next.flags == 16, packet.next.next.flags == 8, packet.next.next.flags == 4,
          packet.next.next.flags == 2,
          packet.next.next.flags == 1, packet.next.next.win, 0]
defender = Detector()
packets.append(x_test)
dpids.append(event.connection.dpid)
if len(packets) >= 25:
    normal = defender.predictResult(packets)
    print "value: ", normal
    if int(normal[0][0] + 0.5):
        print '~~~~~~~~~~~~~~~~~~~~~~~~~~~~normal~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~'
        packets.pop(0)
        dpids.pop(0)
    if not int(normal[0][0] + 0.5):
        print '--------------- -------------------Warning!! DDoS detected! ------------------------------------
        print 'the prediction from LSTM DL model: ', normal[0][0], '(1: normal, 0: attack)'
        print 'blocking the malicious attack......................................'
        print 'event.connection.dpid: ', event.connection.dpid
        print '----------------------------------------------------------------------------------------------
        for i, dpid in enumerate(dpids):
            # print "inport, dstport: ", packets[i][9], packets[i][10]
            msg = of.ofp_packet_out(in_port=packets[i][9] + 25)
            core.openflow.sendToDPID(dpid, msg)
        packets.pop(0)
        dpids.pop(0)
```

Figure 3: code details

The Figure 3 shows the detail code implementation in l3_mySwitch which is a modified version of l3_learning. The predict function in LSTM model is imported in the pox file and the new packet is combined with rest of 24 other packets, being passed as an argument in the function.

5 DDoS Attack Mitigation

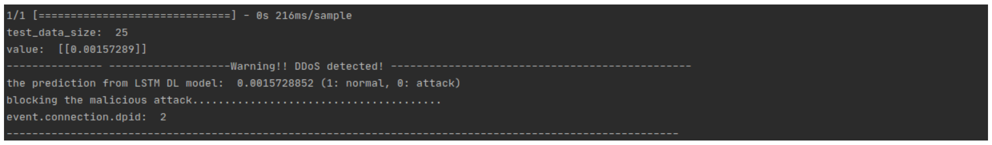
The mitigation is performed by using a method provided by openflow library module, ofp_packet_out. Since the model requires 25 length of packets, the mitigation will be performed after the 25 packets of malicious messages, which still meets the meaning of early detection and mitigation concept, since the packet is being sent per 10 millisecond. To send the packet out message to the core, you need the source port and dpid information which both can be obtained easily with POX module.



```
ith a freshly initialized optimizer.
1/1 [==============================] - 0s 231ms/sample
test_data_size:  25
value:  [[0.9949033]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~normal~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
forwarding
dpid and port: ( 1 2 )
WARNING:tensorflow:Sequential models without an `input_shape` passed to the first layer cannot reload their optimizer state. As a result, your model isstarting w
ith a freshly initialized optimizer.
1/1 [==============================] - 0s 208ms/sample
test_data_size:  25
value:  [[0.9949031]]
~~~~~~~~~~~~~~~~~~~~~~~~~~~~normal~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

Figure 4: Normal traffic

The Figure 4 shows how the l3_mySwitch treats the normal traffic. Since there is no reason to block the normal TCP traffic, it just indicates that it has received a normal packet. The normal traffic is generated using hping3 command, 'hping3 10.0.0.4'. The predict value is 0.994 which is very close to 1 which indicates that the packet is a normal packet.



```
1/1 [==============================] - 0s 216ms/sample
test_data_size:  25
value:  [[0.00157289]]
--------------- -------------------Warning!! DDoS detected! ----------------------------------------------
the prediction from LSTM DL model:  0.0015728852 (1: normal, 0: attack)
blocking the malicious attack.......................................
event.connection.dpid:  2
----------------------------------------------------------------------------------------------------------
```

Figure 5: DDos attack

The Figure 5 shows how the DDoS attack is detected and mitigated in the project. Along with normal traffic, the DDoS attack is captured with the prediction 0.00157 which refers that the packet is malicious.

Consequently, both the Figure 4 and Figure 5 shows how the prediction is polarized, meaning that the model is well-trained so that it can clearly distinguish between the normal and malicious packets.

6 Evaluation

The result has shown a remarkable accurate detection and successful mitigation in SDN environment. This result shows how the DNN model can show a better performance than other existing Machine Learning model. The accuracy is unbelievably high as you can see in the Figure 6 in Appendix. the Train slope skyrockets after a couple of epochs and it tells how the DNN model can effectively distinguish DDoS attack from normal traffic. For the loss function, mean absolute error is applied, since it gives errors between paired observations with the same phenomenon. As you can see in the Figure 7, the loss significantly drops as the epoch increases. Therefore the model has very low chances to do a false detect a DDoS attack. The false positive and the false negative are remarkably low as the count are close to 0. The table Figure 8 shows how the result is divided and it refers that the model can perfectly detect the malicious attack and the normal traffic.

7 Conclusion

The SDN network provides a flexible and various methods that the developer can implement. Not only does it opens the easy gate for attackers to harm the network, but it also means that it opens the same gate for defender to maintain the networks. The current project has focused on the UDP and TCP flood attacks and it has been figured out that such attack can be successfully mitigated with DNN model. Nevertheless, the attacker can further imitate the packets as the normal packets, and probably also can use the DNN model to attack and generate the defense system. Counting all of the above discussion into account, I came to conclusion that we consistently need to reinforce the defense system, researching more DNN layers so that we can react to any form of mutated DDoS attack in the future.

8 Future Work

There exists a number of types of DDoS attack including Smurf Attack, Fragile Attack, Application Layer Attack and etc. My model is not trained to detect such other form of attacks but only UDP and TCP floods. Conseqeuntly, the future work may include detecting and mitigating such other form of attacks, and integrating them into one defense network system and implementing on SDN network environment to encourage the useful and beneficial system.

9 Supporting Material

The source code is available in Github repository below. You can clone the entire repository and follow the instruction in README.md
https://github.com/coodal/DDoS-Detection-and-Mitigation-in-SDN-environment-using-POX-and-LSTM-DNN-model

10 appendix

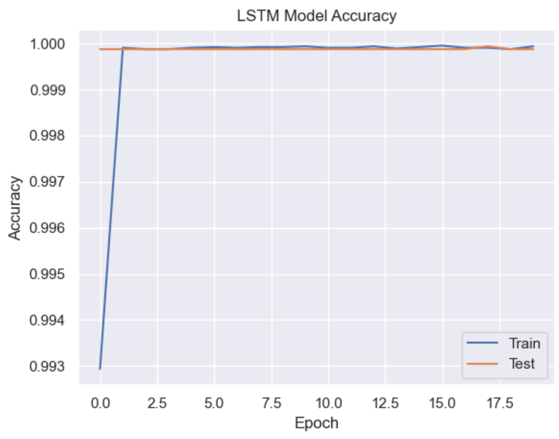


Figure 6: Accuracy

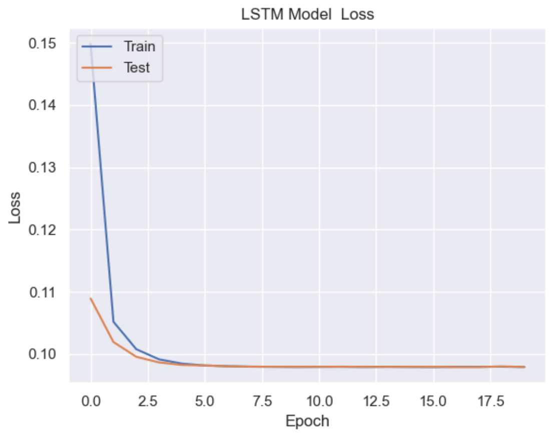


Figure 7: Loss

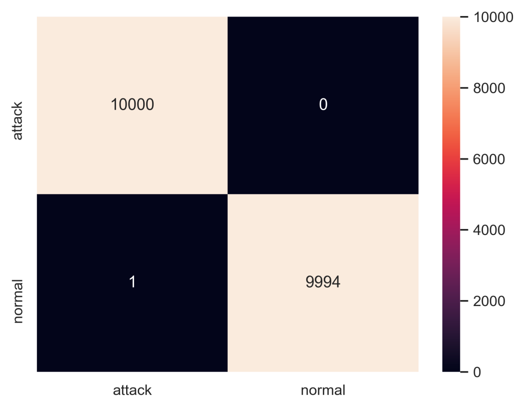


Figure 8: Confusion Matrix

11 References

[1] Huseyin Polat Onur Polat Aydin Cetin, 2020. ”Detecting DDoS Attacks in Software-Defined Networks Through Feature Selection Methods and Machine Learning Models,” Sustainability, MDPI, Open Access Journal, vol. 12(3), pages 1-16, February.

[2] S. M. Mousavi and M. St-Hilaire, ”Early detection of DDoS attacks against SDN controllers,” 2015 International Conference on Computing, Networking and Communications (ICNC), Garden Grove, CA, 2015, pp. 77-81, doi: 10.1109/ICCNC.2015.7069319.

[3] D. Kreutz, F. M. V. Ramos, P. E. Veríssimo, C. E. Rothenberg, S. Azodolmolky and S. Uhlig, ”Software-Defined Networking: A Comprehensive Survey,” in Proceedings of the IEEE, vol. 103, no. 1, pp. 14-76, Jan. 2015, doi: 10.1109/JPROC.2014.2371999.

[4] Li, Chuanhuang Wu, Yan Yuan, XiaoYong Sun, Zhengjun Wang, Weiming Li, Xiaolin Gong, Liang. (2018). Detection and defense of DDoS attack-based on deep learning in OpenFlow-based SDN. International Journal of Communication Systems. 31. e3497. 10.1002/dac.3497.