OpenFlow (D)DoS Mitigation

C. Dillon, M. Berkelaar

February 9, 2014

Abstract

This paper proposes a DDoS mitigation solution that utilizes OpenFlow. OpenFlow keeps statistics on traffic flows. These statistics can be monitored to detect a large spike in traffic, which could be an indication of a DDoS attack. OpenFlow can mirror traffic for the suspicious flow to an Intrusion Detection System, which could be integrated in the OpenFlow controller. The controller can then analyze the traffic and determine if a DDoS attack is in progress and what the sources of the attack are. After that, flows can be created to drop all traffic from those sources.

Because DDoS detection is not a part of OpenFlow, this paper proposes two methods for identifying DDoS attacks in traffic flows. The first method analyzes packet symmetry to identify malicious traffic. The second method temporarily blocks outgoing traffic and checks which sources continue to send packets. Source-based blocking of these identified sources is ultimately reached with explicit flows matching source addresses. These mechanisms are included in custom OpenFlow controller software.

Contents

1 Introduction **3**
1.1 Research question . . . 3
1.2 Related work . . . 3

2 Background **4**
2.1 DDoS attacks . . . 4
2.2 OpenFlow . . . 4

3 Utilizing OpenFLow in DDoS mitigation **5**
3.1 Flow statistics . . . 5
3.2 Traffic sampling . . . 6
3.2.1 Mirroring . . . 6
3.2.2 Packet-in . . . 6
3.3 Traffic dropping . . . 6

4 Proposed solution **7**
4.1 Initial detection . . . 7
4.2 Identification . . . 8
4.2.1 Packet symmetry . . . 9
4.2.2 Temporary blocking . . . 9
4.3 Blocking . . . 10

5 Proof of concept **11**
5.1 Experimentation setup . . . 11
5.2 Performed experiments . . . 11
5.2.1 Packet symmetry . . . 11
5.2.2 Outgoing traffic block . . . 12

6 Conclusion **12**

7 Future work **13**

8 Appendices **16**
8.1 Initial detection . . . 16
8.2 Identification: Packet symmetry . . . 16
8.3 Identification: Temporary block . . . 16
8.4 Mitigation: Blocking . . . 17

1 Introduction

Distributed Denial of Service (DDoS) attacks remain a popular method to degrade the availability of targeted services on the internet. Different DDoS attacks exist ranging from high volume flooding on the network to the misuse of application level vulnerabilities. Although the availability of a service or the congestion of the network is relatively easy to probe, the exact detection of attackers in a DOS introduces a bigger challenge because of the possible similarities with legitimate traffic.

Many DDoS mitigation solutions exist today. A popular choice is BGP Remote Triggered Black Hole (RTBH), which instructs routers to drop all traffic to the target in order to decrease the load on the network [1]. Also, a wide variety of in-line DDoS mitigation hardware appliances exist that filter different kinds of attacks.

Software Defined Networking (SDN) technologies like Openflow introduce more ways of controlling switching and routing[2]. This research explores the possibilities of utilizing the Openflow infrastructure itself to mitigate DDoS attacks. First we explain how OpenFlow features can be used for detecting traffic spikes, sampling suspicious traffic and dropping unwanted traffic. Then we propose our DDoS mitigation solution that uses these mechanisms and the custom algorithms that are implemented in our OpenFlow controller. We explain what experiments we performed to test our solution and what the results are. Based on these results we will draw a conclusion and make suggestions for future work.

1.1 Research question

The research question for this project is: *How can Openflow be used in DDoS mitigation?* The mitigation of a DDoS attack is further divided into the following subquestions about the detection of these and the blocking actions to perform.

- How can flow statistics be analyzed to detect DDoS attacks?
- Can packet symmetry in sample traffic be used to detect malicious traffic sources?
- Can malicious traffic sources be detected by temporarily dropping outgoing traffic?
- Can Openflow be used to efficiently block malicious sources while allowing legitimate traffic?

1.2 Related work

Utilizing SDN for DDoS mitigation and other security practices has been the subject of a number of recent studies. R. Braga, E. Mota and A. Passito use an DDoS detection mechanism based on Self Organizing Maps (SOM), which is an unsupervised artificial neural network. SOM is trained by feeding it statistics of OpenFlow traffic flows. It is then able to classify traffic as abnormal or normal[4].

T. Yuzawa uses sFlow to detect DDoS attacks and to determine which sources are malicious. Upon detection a regular BGP RTBH route is installed for the destination,

but OpenFlow is used to change the flow of legit traffic so that this does not get dropped. This is done by using Floodlight's static flow pushing API[3].

Chunghwa Telecom Co. propose an OpenFlow DDoS Defender that monitors flows on an OpenFlow switch. They define a number of thresholds, and start to drop incoming traffic when these thresholds are met[7].

S. Akbar Mehdi, J. Khalid, and S. Ali Khayam use SDN for traffic anomaly detection. They implement a number of detection algorithms in the NOX OpenFlow controller. Their solution only uses the first packet that is sent to the controller when a new connection is established, while allowing the rest of the traffic to continue at line-rate by pushing a flow for that connection. They found that this is an effective approach to keeping track what is going on in the network, without need for excessive sampling of traffic[16].

2 Background

2.1 DDoS attacks

There are numerous denial of service attack methods being used to degrade the performance or availability of targeted services on the internet. Usually these methods can be classified as either network- or application level attacks.

Network level attacks generally produce large volumes of network traffic that are detectable by their packet- or bandwidth-rate. Common examples for this are amplification and flood attacks.

Application level attacks misuse software in a malicious way, aiming to exhaust resources to process any further requests. These attacks are generally harder to detect on the network level as they show no clear deviation from legitimate traffic. Common examples here are expensive search queries and exhaustion of connection pools.

OpenFlow rules match against properties in link, network and transport layers of the TCP/IP model and is therefore not suited to detect application level attacks with characteristics present in higher layers. This research will explore the utilization of Openflow in the detection and mitigation of network level attacks.

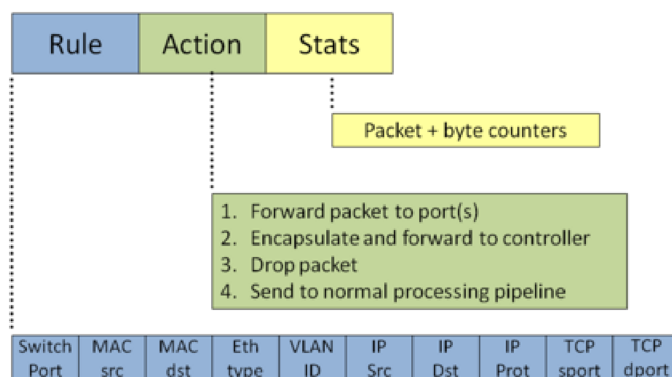
2.2 OpenFlow

Before we explain how OpenFlow can be used in DDoS mitigation, a basic understanding is needed of how OpenFLow works and what its limitations are. With OpenFlow the control plane of the switch is implemented in a separate machine, the OpenFlow controller. The controller and switch communicate via the OpenFlow protocol. The controller can install flows on the switch and the switch forwards traffic according to these flows.

A flow is stored as a 12-tuple with fields that match against incoming packets. Each flow has an action, a priority and statistics. This is illustrated in Figure 1.

[1] `http://yuba.stanford.edu/cs244wiki/index.php/Overview`

Figure 1: OpenFlow Table Entry[1]



Flows are stored in the Ternary Content-Addressable Memory (TCAM) table of the switch. In hardware switches the number of flows is limited by the size of the TCAM table. When a switch receives a packet, it tries to match it against a flow in the table, if a match is found the action associated with the flow is taken and the flow statistics are updated. If there are no matching flows, the switch will send a Packet-In message to the controller. The Packet-In message either encapsulates the entire packet or the header of the packet. The controller sends back a Packet-Out message to the switch to tell the switch what action to take. The controller can also push a new flow to switches in the network, so that future packets will match against the flow. [8]

3 Utilizing OpenFLow in DDoS mitigation

A number of methods have been researched in the field of detecting malicious activity using Openflow. These methods vary from the detection of infected hosts on the local network by comparing flows [16] to deterministic sampling using Openflow to inspect certain traffic classes. [6]

We researched the possibilities of using Openflow to detect Denial of Service attacks with the features available in Openflow version 1.0. The complete mitigation proposed is divided in the initial detection, sampling methods and blocking actions that can be taken with Openflow.

3.1 Flow statistics

Openflow keeps basic statistics for active flows, like the total amount of bytes and packets matched and the time since the flow was deployed and last used.

These statistics can be polled at regular intervals by the controller to create a plot of the network utilization for individual flows. Keeping a history of this data allows for detection of spikes in traffic that may indicate an attack on the network or a specific host.

3.2 Traffic sampling

Traffic sampling is an important part of all Intrusion Detection Systems (IDS). Some research has been done on how to utilize SDN in intrusion detection. sFlow is a popular choice for this task as it's specifically designed for traffic sampling[9]. It enables switches and routers to send 1 in every X packets to a server called an sFlow collector, where X is a predetermined number. It also keeps statistics on all interfaces, comparable to the OpenFlow flow statistics.

OpenFlow focuses on complete traffic flows and not individual packets, this makes it less attractive for sampling. S. Shirali-Shahreza and Y. Ganjali propose a solution for this problem in the form of an extension to the OpenFlow specification, which allows OpenFlow to sample individual packets, comparable to sFlow[6]. However, in the current OpenFlow specification such a solution is not available. OpenFlow does provide mechanisms which we can use for temporarily sampling traffic flows. And because OpenFlow allows for per-flow configuration of sampling, it can be more flexible than sFlow.

3.2.1 Mirroring

Flow mirroring is a method to sample all traffic that matches a flow that is believed to forward DOS traffic. The action for that flow is modified to not only forward packets to the real destination, but also to the controller. In comparison to conventional mirroring based on switchport numbers this allows for filtering using the Openflow flow entry, limiting the sample to only the traffic that is needed for further analysis.

3.2.2 Packet-in

The packet-in communication channel between the switch and controller can be utilized to send sample packets for analysis. An extra action in the flow entry specifies the traffic to not only be forwarded to the destination, but also be encapsulated and sent to the controller over the packet-in channel. The packet-in channel is normally used when the switch has no matching flow and the controller has to create and push one based on the information that unmatched packet provides. For these decisions it is generally not required to have any information beyond the L4 header. In order to not waste bandwidth and resources on sending parts of packets that are not needed it is possible to specify a byte range that should be sampled, stripping unneeded payloads.

3.3 Traffic dropping

OpenFlow offers a lot of flexiblity in dropping traffic by setting the action of a flow to drop all packets for that flow. When sources of malicious traffic are identified, flows that match against the IP addresses of those sources can be pushed to a switch so that traffic from those sources is dropped. This of course will only work if the DDoS attacker does not spoof the source addresses. If the sources are not spoofed the effectiveness of sources based filtering depends on the scale of the attack and the size of the TCAM table in the

switch, as every flow will take up space in the table. However, there are switches which can store over 100,000 flows[10].

If source based filtering is not an option, destination based filtering can still be done to avoid congestion in the network. While traditional destination based filtering like BGP RTBH drops all traffic for that destination, OpenFlow can match specific classes of traffic. If an attack only generates UDP traffic on a specific port, a flow can be installed that drop all traffic for the destination on that UDP port. This allows other traffic to continue as usual.

4 Proposed solution

We propose a DDOS mitigation method that utilizes OpenFlow to assist in the detection and blocking of attacks. To do this we divide the complete mitigation into a number of components:

- Initial detection
- Identification
- Blocking

The initial detection will utilize OpenFlow flow statistics to detect anomolies in network usage. The identification will then detect the sources that are performing an attack through the analysis of packet samples. Identified attackers will finally be dropped with OpenFlow drop flows.

4.1 Initial detection

The flow statistics offer data that can be used to detect potential DDoS attacks, although it may not result in explicitly detecting the attacking sources. For this reason we propose the use of the flow statistics merely to trigger further detection mechanisms.

The initial detection will primarily focus on anomaly detection with the byte- and packet counters that are kept for every active flow. Every second these statistics are polled by the controller and the difference between the current and previous value is prepended to data set Q, which has a fixed size of 60 entries. Every entry in Q represents the packet/byte rate for that interval. A dataset size of 60 entries was chosen to make sure that sudden deviations in this rate are easily detectable.

The standard deviation of this list ($\sigma(Q)$) represents the apparent deviation between the packet and byte rates over the last 60 entries.

$$\sigma(Q) = \sqrt{\frac{1}{60}\sum_{i=1}^{60}(Q_i - \mu)^2}$$

With a comparison of the expected deviation, based on the standard deviation, and the real deviation in data set Q it is possible to detect anomalies that may potentially

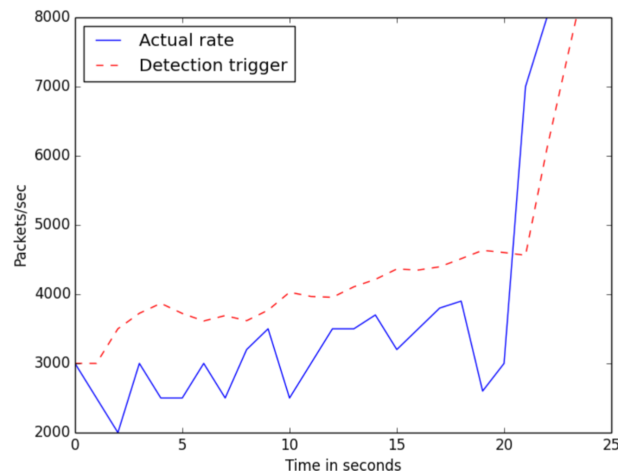
be DDoS attacks. The first entry of Q, (Q_1), is the newest value from the flow statistics. This value is subtracted with the mean of Q to get the deviation D of Q_1. This value is compared with variable tresholds to determine if it falls out of the expected boundaries and is thus considered a potential attack. The detection of a potential attack will trigger further detection mechanisms, denoted with Z.
The following items form the tresholds used during the experiment:

- Deviation (D) from the mean is positive, with $D > 0$
- Minimum deviation (M) to trigger is three times the standard deviation, where $M = (Q_1 - \mu(Q)) > (3 \cdot \sigma(Q))$
- The minimum value (V)(Value of Q_1b (Bytes) is larger than 5000000 Bytes or Q_1p (Packets) is larger than 5000 packets, with $Q_1b > 5000000 \vee Q_1p > 5000$, preventing likely false positives at low bandwidth rates.

$$D \wedge M \wedge V \rightarrow Z$$

Figure 2 shows a plot of a simulated initial detection using the tresholds mentioned before. A sudden burst of packets triggers the mechanism here to perform an analysis determining if this is a DDoS. The initial detection mechanism is implemented in the Ryu framework with a sample of the code included in section 8.1.

Figure 2: Initial detection



4.2 Identification

OpenFlow by itself is not entirely capable of detecting DDoS attacks, but can be leveraged in the process by external logic. The identification of malicious attackers is triggered by the initial detection and consists of two explored methods. The first method

looks into sampling specific flows and analyzing packet symmetry. The second method uses OpenFlow to sample traffic and temporarily block outgoing traffic to detect which sources continue to send requests while there are no replies.

4.2.1 Packet symmetry

Packet symmetry can be used to distinguish malicious from legitimate traffic[5]. An attacker will send huge amounts of packets to the victim, even if there are no replies. While normal protocols might not blindly continue to send data if there are no acknowledgements or replies. This is especially true for TCP as every packet has to be acknowledged. But even UDP traffic does not exceed a symmetry ratio of 8:1 under normal circumstances[5].

Using OpenFlow, we can create flows for incoming and outgoing traffic from the victim of a DDoS attack and mirror this traffic to the controller. Having sample traffic from and to the victim allows us to analyze the ratio between requests and replies per source. Sources with a highly asymmetric ratio can be considered malicious and flows can be pushed to drop traffic from these sources, as explained in chapter 4.3. In order for this to work we need to define a threshold for the symmetry ratio. For our experiments we used a threshold of 50:1, but we make no claims on what an appropriate threshold is for real world situations.

4.2.2 Temporary blocking

Openflow enables easy modification of forwarding rules that make it possible to temporarily halt traffic flows and register the behaviour of a set of sources. We propose a mechanism that drops outgoing traffic from the target for a short duration of time, enabling one to recognize well- and non-well behaving sources. We found that a temporary block with a duration of approximately one second is enough to detect non-well behaving clients, whilst not breaking common legitimate traffic like SCP- and HTTP transfers.

Based on the characteristics of common protocols one can expect sources to behave in a certain way. TCP streams may for example retransmit the previous segments when no acknowledgements are received and increase the retransmit timer resulting in a dropping rate of traffic.[14]

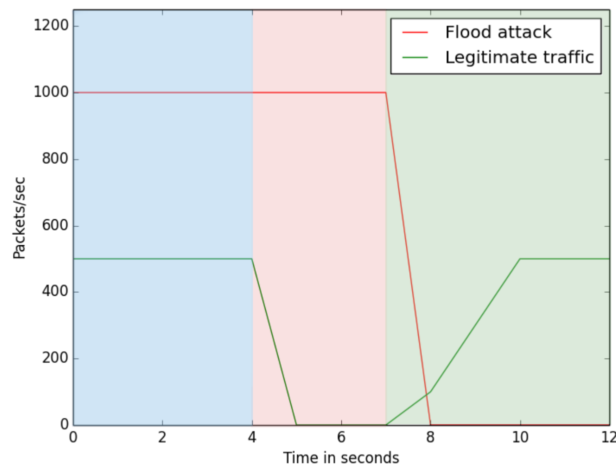
UDP traffic is not dependent on acknowledgements, but is likely to be request-response based from the application layer.[15] If no requests are sent out we expect no incoming responses. The behaviour of the latter holds true for services like DNS and NTP, often used in spoofed DDOS attacks[5], but may not be applicable to other UDP based communication. The behaviour of common services that utilize UDP and ICMP in contrast to this mechanism should be further researched.

Figure 3 represents the three stages in this mechanism, consisting of sampling, blocking and analysis. The first stage installs sampling and blocking flows after an initial detection (4.1) was made. The sample flows forward traffic to both the controller and

the destination, while the blocking flows only forward traffic to the controller and thus block traffic whilst active after the first sample flows expired. During this stage the controller will learn the rate of packets from every source communicating with the target. The second stage learns the packet rate from every source while the block flow is active.

The ratio R between incoming packets before (P_{before}) and after (P_{after}) the block can be analysed to determine if sources are either well-behaving flows or floods. R is simply calculated with $\frac{P_(after)}{P_(before)}$ Floods can be recognized with a ratio that is close or equal to $1:1$ For our experiments we used a threshold of $1:5$ as the lower boundary, but further research is required to find an optimal treshold for real network traffic. Sources with a higher ratio are marked as an attack and instructed to be blocked with an OpenFlow drop flow, while legitimate traffic will most likely recover after a short interruption [4.3].

Figure 3: Temporary block visualised



4.3 Blocking

OpenFlow enables filtering with flows that could potentially only drop sources communicating with specific protocols and/or ports.[3.3] During this research, the goal was to find an effective way to do source-based blocking with OpenFlow, leaving fine grained filters to be further researched.

Block flows preventing DoS sources from further communicating with hosts on the local network will be created on the OpenFlow switch closest to the uplink. Block flows will match on the source IP address of the attacker and will be given the highest priority of `0xFFFF`, ensuring this flow to overrule any other matching flows currently active. To prevent the TCAM table of switches to fill with unused blockflows the idle-timeout of these flows will be set to 15 minutes. This automatically deletes block flows that haven't shown activity for this duration. The code to install blocking flows is included in section 8.4.

5 Proof of concept

5.1 Experimentation setup

We used two different experimentation setups for our Proof of Concept. One with a software switch and one with a hardware switch. The first environment consists of a single machine running Linux KVM and OpenVSwitch. We created two Virtual Machines (VM) on KVM, one to perform the attacks and the other as victim. Both VMs are connected to an OpenVSwitch interface. The KVM host is also connected to OpenVSwitch and runs the controller software.

For the second environment we used a VMware ESXi server and an Arista 7050 switch. We installed three VMs on VMware. An attacker, a victim and a controller. The attacker has a 10gb link to the switch and the controller and victim have separate 1gb links.

We wrote our own controller software using the Python Ryu SDN framework[11]. The controller includes our DDoS detection and mitigation mechanisms as explained in Chapter 4. The source code is available on GitHub[12] and some important parts are included in the appendix.

We installed the nginx webserver on the victim VM which was used as source for legitimate traffic. We simulated DDoS attacks using the traffic generator hping3[13] from the attacker VM.

5.2 Performed experiments

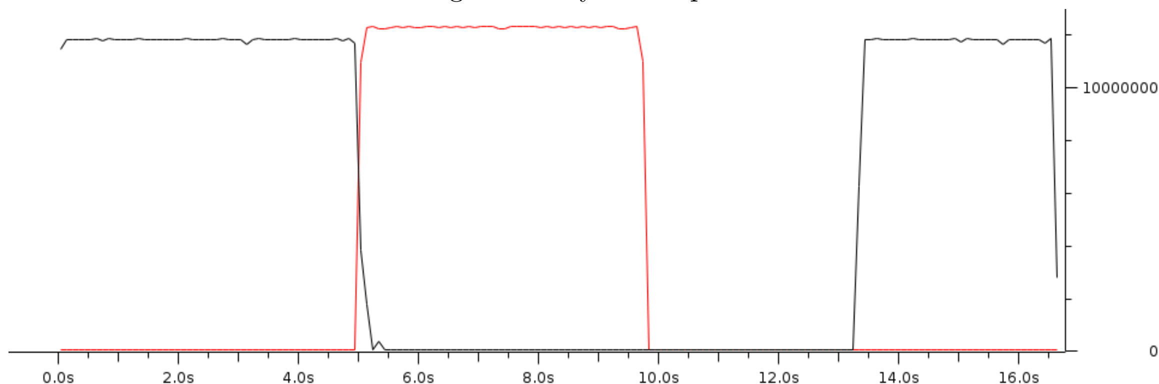
5.2.1 Packet symmetry

Our first experiment was successfully performed on hardware and software switches. Here we explain the experiment on the hardware switch setup. First, from the attacker, we started a large file transfer over HTTP on the victim. This saturates the 1gbps uplink of the victim machine. This is shown as the black line in Figure 4. Then we started a DDoS from five spoofed IP addresses from the attacker to the victim. The DDoS traffic completely suppresses the HTTP traffic. This is shown as the red line in Figure 4.

The controller polls the flow statistics every two seconds and notices the high rise in traffic on the flow towards the victim. This triggers the controller to push two new flows to the switch: one from the attacker to the victim and one from the victim to the attacker. Both flows output traffic to the regular port and also to the controller. These flows automatically expire after 10 seconds.

While traffic is being sampled to the controller, the controller counts the ratio between incoming and outgoing traffic per IP address. The DDoS traffic has a high incoming traffic ratio because the victim does not reply to packets and the attacker keeps sending packets, while the regular HTTP traffic stops because the victim does not reply. Once the ratio counters exceed a treshold of 50:1, new flows are pushed to drop traffic from the source IP. This is visible in after the 9th second in Figure 4. Shortly after that, the TCP HTTP session is restored and the transfer of the file continues. This is visible after the 13th second in Figure 4. The mechanism for the packet symmetry detection is worked out in Python code included in section 8.2.

Figure 4: Async dump



5.2.2 Outgoing traffic block

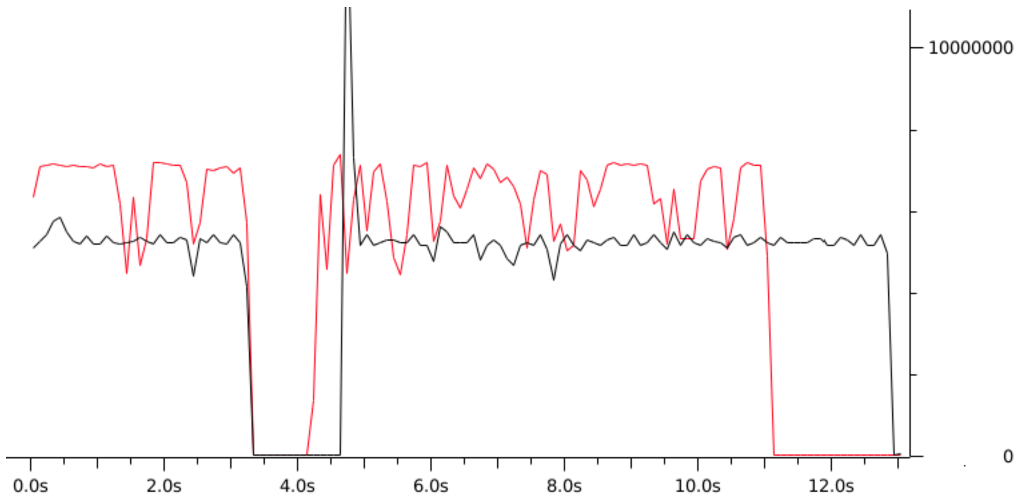
The second experiment was only successfully performed on the software switch, because the timing of installing flows on the hardware switch was found to be unreliable. Note that because of the software switch, the DDoS traffic was not able to fully suppress the HTTP traffic in Figure 5. This experiment started in the same way as the first: the attacker starts a HTTP file transfer from the victim with a DDoS simulation closely followed.

The initial detection trigger the creation of two sample flows and two block flows. The sample flows mirror the traffic going to the target for a duration of two seconds after which they expire. On expiration of the sample flows the block flows become active. These flows only forward traffic to the controller, essentially blocking all communication to the target. The flooding based attack shows no sign of slowing down while no responses from the target are sent. The HTTP transfer essentially halts because no acknowledgements are received, thus only retransmitting previous segments at a declining rate. Figure 5 doesn't show the flood to continue at the same rate as this plot is made from a TCPdump on the DDoS target, thus missing all traffic that was sent while the block flow was active. On expiration of the block flow the original flows restore communication between all sources and the DDoS target. The controller will now analyse all traffic that was mirrored to identify the sources that did not show a declining rate of traffic while the drop flow was active. Because the software switch environment was not capable of analysing at line rate it took the controller till 11 seconds into the expirimentation to block the attacking sources. The code used for temporary blocking is included in section 8.3.

6 Conclusion

During this research we looked at how OpenFlow can be utilized in the field of DDoS mitigation. The features of OpenFlow are leveraged with the initial detection to isolate DDoS victims and trigger further analysis. The identification of attackers is performed

Figure 5: blockflow dump



with sample flows to either perform packet symmetry analysis or block outgoing traffic with additional flows to detect flooding based attacks. On positive detection, blocking flows will be pushed to the switch to prevent the attack from congesting the network.

Attack simulations show that it's possible to do fairly accurate detection of floods with packet symmetry analysis, keeping the victim available without interruption. Temporarily blocking the outgoing traffic with OpenFlow shows potential to get even more accurate detections with connection oriented- or request-response based protocols, at the cost of short interruptions to availability. Experiments of the latter detection mechanism on the hardware switch showed timing related issues, limiting us to the software platform for testing.

OpenFlow shows potential in DDoS mitigation with easy control over individual flows in the network and the capacity to filter at line-rate. However, with the limited size of TCAM tables in hardware switches currently available it leaves the application in real environments as something that has yet to be explored.

7 Future work

The tresholds used throughout this research are mostly based on simulations that may not be well suited for real world scenarios. Tests of these mechanisms should be performed to confirm the tresholds set during this research, or find more appropiate ones, with real live data.

This research focussed on the protection from DDoS attacks originating from outside of the local network. We believe that the explored mechanisms could be used to detect attacks that originate from the local network and participate in a DDoS attack. Scenarios for this could be infected zombie hosts or services being leveraged in amplification attacks.

Temporary outgoing blocks that we propose simply look at the ratio of packets before and after a blocking flow is installed. A more advanced detection mechanism could also look at characteristics of the traffic being dropped, like looking for retransmissions or retransmission timers.

References

[1] W. Kumari, D. McPherson, *RFC 5635: Remote Triggered Black Hole Filtering with Unicast Reverse Path Forwarding (uRPF)*, 2009
`http://www.hjp.at/doc/rfc/rfc5635.html`

[2] N. McKeown, T. Anderson, H. Balakrishnan, G. Parulkar, L. Peterson, J. Rexford, S. Shenker, J. Turner, *OpenFlow: Enabling Innovation in Campus Networks*, 2008
`http://www.net.t-labs.tu-berlin.de/teaching/ss09/IR_seminar/papers/openflow-wp-latest.pdf`

[3] T. Yuzawa, *OpenFlow 1.0 Actual Use-Case: RTBH of DDoS Traffic While Keeping the Target Online*, 2013
`http://packetpushers.net/openflow-1-0-actual-use-case-rtbh-of-ddos-traffic-while-keeping-the-target-online/`

[4] R. Braga, E. Mota, A. Passito, *Lightweight DDoS Flooding Attack Detection Using NOX/OpenFlow*, 2010
`http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5735752`

[5] M. Wood, *Preventing Denial-of-Service Attacks with Packet Symmetry*, 2008
`http://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.186.4788&rep=rep1&type=pdf`

[6] S. Shirali-Shahreza, Y. Ganjali, *FleXam: Flexible Sampling Extension for Monitoring and Security Applications in OpenFlow*, 2011
`http://conferences.sigcomm.org/sigcomm/2013/papers/hotsdn/p167.pdf`

[7] C. YuHunag, T. MinChi, C. YaoTing, C. YuChie, C.YanRen, *A Novel Design for Future On-Demand Service and Security*, 2010
`http://ieeexplore.ieee.org/xpl/login.jsp?tp=&arnumber=5689156`

[8] Open Networking Foundation, *OpenFlow Switch Specification v1.0*, 2009
`https://www.opennetworking.org/images/stories/downloads/sdn-resources/onf-specifications/openflow/openflow-spec-v1.0.0.pdf`

[9] P. Phaal, M. Lavine (sFlow.org), *sFlow Version 5*, 2004
`http://sflow.org/sflow_version_5.txt`

[10] IBM, *IBM System Networking RackSwitch G8264CS*, 2013
`http://www.redbooks.ibm.com/abstracts/tips0970.html?Open#contents`

[11] Open Source Software Computing Group, *Ryu SDN framework*, 2014
http://osrg.github.io/ryu/

[12] C. Dillon, M. Berkelaar *OpenFlow DDoS mitigation controller*, 2014
https://github.com/ConnorDillon/openflowddos

[13] hping.org, *http://www.hping.org/hping3.html*, 2014
http://www.hping.org/hping3.html

[14] Tcpipguide.com, *TCP adaptive retransmission calculations*, 2005
http://www.tcpipguide.com/free/t_TCPAdaptiveRetransmissionandRetr
ansmissionTimerCal-2.htm

[15] Tcpipguide.com, *UDP operation*, 2005
http://www.tcpipguide.com/free/t_UDPOperation.htm

[16] S. Akbar Mehdi, J. Khalid, and S. Ali Khayam, *Revisiting Traffic Anomaly Detection using Software Defined Networking*, 2011
http://www.xflowresearch.com/docs/Revisiting_Traffic_Anomaly_
Detection_using_Software_Defined_Networking.pdf

8 Appendices

The full source code of our OpenFlow controller can be found at:
https://github.com/ConnorDillon/openflowddos

8.1 Initial detection

```
def detect(self, cookie, treshold, data):
    difference_list = []
    for i in range(0, (len(data) - 1)):
        difference = data[i] - data[i + 1]
        difference_list.append(difference)
    if len(difference_list) == 0:
        return 0
    average_difference = sum(difference_list) / len(difference_list)
    std = numpy.std(difference_list)

    for item in difference_list:
        if item - average_difference > 3 * std and item > treshold and item - average_difference >= 0:
            # print item, " is > then 3 times the standard deviation", std, " item: ", item
            if not cookie in self.detected_flags:
                self.detected_flags[cookie] = 1
                return 1
            else:
                if self.detected_flags[cookie] == 1:
                    return 0
                elif self.detected_flags[cookie] == 0:
                    self.detected_flags[cookie] = 1
                    return 1
        else:
            return 0
```

8.2 Identification: Packet symmetry

```
def ratio_counter(self, packet):
    pair_list = sorted([packet['ip_src'], packet['ip_dst']])
    pair = '-'.join(pair_list)
    if pair in self.ratio_count:
        self.ratio_count[pair][packet['ip_src']] += 1
        ratio = float(self.ratio_count[pair][pair_list[0]]) / float(self.ratio_count[pair][pair_list[1]])
        self.ratio_count[pair]['ratio'] = ratio
        if ratio > 50:
            self.create_blocking_flow(packet['ip_src'])
    else:
        self.ratio_count[pair] = {packet['ip_src']: 1, packet['ip_dst']: 1, 'ratio': 1}
```

8.3 Identification: Temporary block

```
def outgoing_block_v2(self, packet):
    mac_dst = packet['mac_dst']
    if mac_dst in outgoing_block_mode_v2_dict:
        packet_time = int(packet['time'] * 1000)  # msec
        pair_list = sorted([packet['ip_src'], packet['ip_dst']])
        pair = '-'.join(pair_list)
        if outgoing_block_mode_v2_dict[mac_dst][0] < packet_time < \
                (outgoing_block_mode_v2_dict[mac_dst][0] + 2000):  # 2000 msec
            if pair in self.block_packet_count:
                # Reset old data
                if self.block_packet_count[pair]['time_created'] < (time.time() - 30):
```

```
                self.block_packet_count[pair] = {'packets_before': 1, 'packets_after': 1,
                                                 'time_created': time.time()}
            else:
                self.block_packet_count[pair]['packets_before'] += 1
        else:
            self.block_packet_count[pair] = {'packets_before': 1, 'packets_after': 1,
                                             'time_created': time.time()}

    block_time = outgoing_block_mode_v2_dict[mac_dst][0] + 2000
    if packet_time > block_time:
        if not pair in self.block_packet_count:
            pass
        else:
            if block_time < packet_time < (block_time + 250):
                # First 250 ms is discarded
                pass
            elif packet_time > block_time and (block_time + 250) < packet_time < (block_time + 750):
                self.block_packet_count[pair]['packets_after'] += 1
            elif (block_time + 750) < packet_time < (block_time + 1100):
                p_before = self.block_packet_count[pair]['packets_before']
                if p_before > 1000:
                    ratio = float(self.block_packet_count[pair]['packets_before']) \
                        / float((self.block_packet_count[pair]['packets_after'] * 4))
                else:
                    ratio = 1
                if ratio < 5:
                    # Switch flood protection
                    if not packet['ip_src'] in self.already_blocked:
                        self.create_blocking_flow(packet['ip_src'])
                        self.already_blocked.append(packet['ip_src'])
                        print "Created blocking flow for: ", packet['ip_src']
```

8.4 Mitigation: Blocking

```
def create_blocking_flow(self, ip_src):
    if not ip_src in self.blocked_sources:
        self.blocked_sources.append(ip_src)
        match = datapath.ofproto_parser.OFPMatch(dl_type=0x0800, nw_src=ipv4_text_to_int(ip_src), nw_src_mask=32)
        mod = datapath.ofproto_parser.OFPFlowMod(
            datapath=datapath, match=match, cookie=random_int(),
            command=datapath.ofproto.OFPFC_ADD, idle_timeout=900, hard_timeout=0,
            priority=0xffff, flags=datapath.ofproto.OFPFF_SEND_FLOW_REM)
        datapath.send_msg(mod)
        print 'creating blocking flow for source: {0}'.format(ip_src)
```