1. Analysis of a DDoS detection and mitigation system in an SDN environment

1.1. Introduction and objectives of the study

This paper discusses a software-defined networking (SDN)-based network architecture designed to automatically detect and mitigate distributed denial of service (DDoS) attacks, with a specific focus on flooding attacks at the network infrastructure layer. The primary goal of the project was to build a realistic test environment using Mininet to emulate the network, sFlow-RT as a real-time traffic collector and analyzer, and an SDN controller (in the final configuration, Ryu) to dynamically orchestrate the defense.

The system implements a closed control loop: the network switch sends traffic samples (sFlow) to the analyzer, which detects anomalous flows based on a predefined threshold. When the threshold is exceeded, a custom application sends a mitigation request to the SDN controller, which installs a blocking rule directly on the switch, neutralizing the source of the attack.

1.2. Test environment architecture

The experimental infrastructure is logically split across two virtual machines (VMs) to separate the control and monitoring plane from the data plane.

- **VM01 - Control & Monitoring Plane**: This VM houses the central components of the system:
    - **SDN Controller (Ryu):** The "brain" of the network, translating safety instructions into OpenFlow flow rules. The final solution is based on Ryu.
    - **sFlow-RT Collector**: Receives the sFlow datagrams and runs the `ddos.js` script, which implements the detection logic and communication with Ryu via REST API.
- **VM02 - Data Plane**: This VM runs the **Mininet** network emulator, creating a virtual topology with:
    - **Open vSwitch (OVS):** A virtual switch that samples traffic and sends it to sFlow-RT.
    - **Virtual Hosts**: Nodes that simulate servers, clients, victims, and attackers.

The operational flow involves the sFlow agent on OVS sending the data to sFlow-RT. The script `ddos.js` analyzes them and, in the event of an attack, contacts Ryu's REST API to program the OVS and block malicious traffic.

1.3. Deployment and configuration details

The configuration of the environment follows precise steps to ensure its operation.

1. **SDN Controller Startup**: On VM01, the Ryu Controller is booted with modules `simple_switch_13` (for L2 learning switch functionality) and `ofctl_rest` (to expose the REST API needed for mitigation).

```
ryu-manager ryu.app.simple_switch_13 ryu.app.ofctl_rest
```

2. **Mininet Topology Creation**: On VM02, the topology is created by specifying the IP of the remote controller.

```
sudo mn --topo single,9 --controller remote,ip=<IP_DELLA_VM_CONTROLLER>
```

3. **Configuring the sFlow Agent**: The `s1` virtual switch is configured to send traffic samples to sFlow-RT (running on VM01).

```
ovs-vsctl -- --id=@sflow create sflow target=\"<IP_VM_SFLOW-RT>:6343\"
header=128 sampling=10 polling=20 -- set bridge s1 sflow=@sflow
```

4. **Starting sFlow-RT and mitigation script**: On VM01, the sFlow-RT collector starts. When it starts, it automatically loads the `ddos.js` script. This script is the heart of the defense logic and defines key parameters:
    1. `threshold`: The threshold of packets per second beyond which the alarm is triggered (e.g. 1000 pps).
    2. `blockSeconds`: The duration in seconds of the block applied to the attacker's IP (e.g. 60s).
    3. `dpid`: The ID of the switch to be programmed (e.g. 1).
    4. `ryuApiUrl`: The URL of the Ryu controller API.
5. **Attack simulation**: The attack is generated by a host in Mininet with `hping3` to create a flood of UDP packets.

```
mininet> hX hping3 --flood --udp hY
```

1.4. Analysis of the results

The effectiveness of the system has been validated through multiple tests. The analysis was extended to include the results of a second scenario with a significantly more intense attack, comparing them to the initial test.

1.4.1. Traffic detection and mitigation (initial test)

The first test, visible in the graph, clearly shows the success of the mechanism.

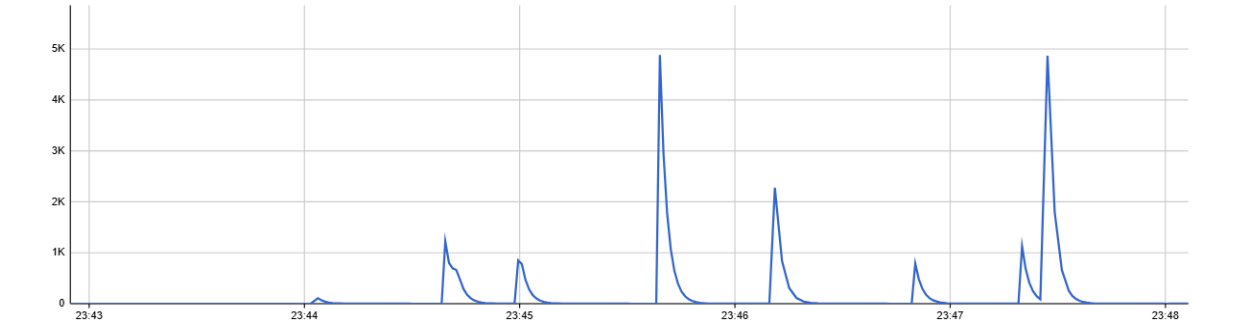

Figure 1: Traffic detection and mitigation

The cyclical trend is evident:

- **Traffic spike**: The attack begins, quickly exceeding the 1000 packets/second threshold.
- **Drop to zero**: The system detects the anomaly and sends a `DROP` rule to Ryu, which installs it on the switch. Traffic from the attacker is immediately blocked.
- **Cyclicity**: When the 60-second lock expires, the rule is removed. If the attack persists, it is immediately detected and blocked again, confirming the continuous operation of the system.

1.4.2. Analysis of results with intensified attack (stress test)

To verify the robustness of the system, a second test was performed using a much more aggressive UDP flood attack, characterized by a **bitrate of 6.83 Mbits/sec**. The results of this stress test confirm and reinforce the initial conclusions.

Network traffic during the stress test

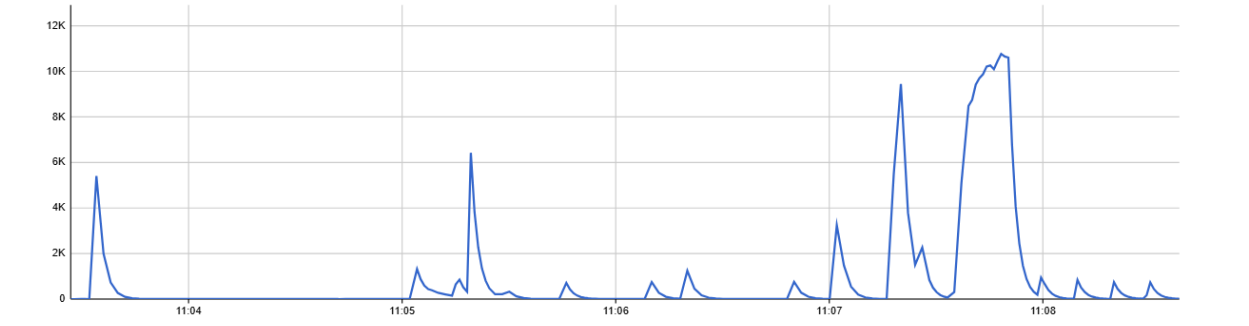

Figure 2: Traffic detection and mitigation - intensified attack

The traffic graph shows traffic spikes that reach and exceed **10,000 packets per second**, an order of magnitude higher than the initial test. Despite the violence of the attack, the behavior of the mitigation system remains impeccable: each peak corresponds to a vertical and immediate drop in traffic to zero. This shows that, even under extreme load, the detection logic and

communication with the Ryu controller are fast and efficient enough to neutralize the threat almost instantly.

Impact on CPU resources during the stress test

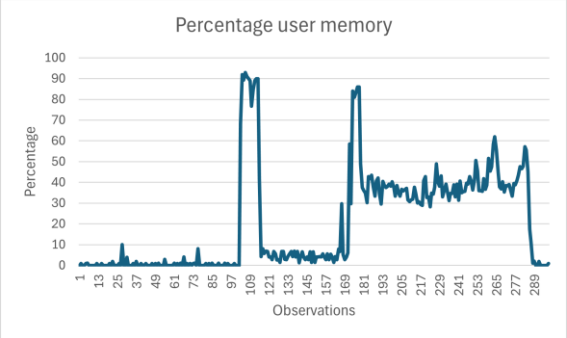

Figure 3: CPU usage in User Space during the stress test

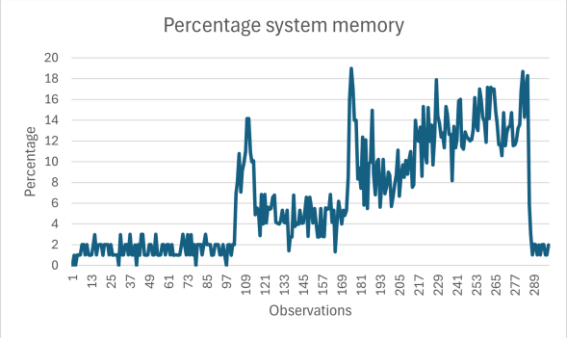

Figure 4: CPU usage in System Space during the stress test

Analyzing CPU resource usage during the escalated attack reveals important data:

- **Correlation with the attack**: As in previous tests, CPU usage spikes, both in user space (`Percentage user memory`) and in system space (`Percentage system memory`), are perfectly synchronized with the traffic spikes of the attack.
- **Increased load**: The increase in CPU utilization in *user space* is even more marked, due to the intensive work of the sFlow-RT process to analyze the massive volume of incoming sFlow samples. Similarly, *system space* usage increases due to the load on the VM kernel, which must handle packet flooding at the network interface layer before the block rule is active.
- **Offload effectiveness**: The crucial aspect is that, even in this high-impact scenario, as soon as the block rule is installed on the switch, **the CPU utilization plummets to the baseline levels**. This is the ultimate demonstration of the advantage of SDN architecture: mitigation is delegated entirely to the data plane (the switch's optimized hardware or software), which manages packet drop efficiently, completely freeing the control plane CPU and operating system resources from the burden of processing malicious traffic.

1.4.3. Network latency analysis

A new analytics metric has been introduced: network latency, measured during stress testing to understand the impact of the attack on quality of service for legitimate traffic.

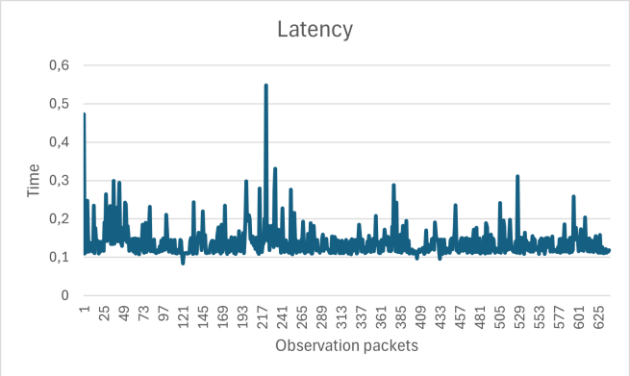

Figure 5: Network latency

The graph clearly shows:

1. **Normal Latency**: In the absence of an attack, latency is low and stable (about 0.1-0.2 units of time).
2. **Latency spikes**: Significant latency spikes (up to over 0.5 units) occur before each DDoS flood starts before the block is triggered. This represents network degradation caused by resource saturation.
3. **Return to Normal**: As soon as the mitigation system installs the `DROP rule`, the latency immediately returns to the baseline values.

This analysis shows not only that the system blocks the attack, but that in doing so **it actively preserves the quality of the network**, restoring normal operating conditions and ensuring that legitimate user traffic is not penalized by malicious activity.

1.5. Conclusion

The project successfully demonstrated the design, implementation and validation of an automated security system for SDN networks. The test results, especially the analysis of traffic and CPU performance, unequivocally confirm the effectiveness of the chosen architecture in detecting and mitigating real-time flooding DDoS attacks.

The results obtained during the **stress test with an intensified attack** further reinforce this conclusion, proving the robustness and scalability of the solution. Even in the face of ten times the volume of traffic, the system responded just as effectively and quickly. Network **latency** analysis added a critical dimension to validation, demonstrating that mitigation not only protects the infrastructure but also restores quality of service for users.

The ability to delegate the blocking action to the switch data plane proves to be a high-performance strategy, which protects the critical resources of the control plane. Finally, the use of the Ryu controller proved to be an efficient choice for the simulated test conditions.