High-Availability Firewall Scenarios

Segurança em Redes de Comunicações

Universidade de Aveiro

Paulo Gil 76361, Diogo Correia 90327

2022/2023

Departamento de Eletrónica, Telecomunicações e Informática

Contents

Load-Balancing with redundancy and state sync

1.1 Insight

State synchronization allows for High Availability and Load Balancing purposes, meaning that Firewalls that, for instance, implement protocols such as Virtual Router Redundancy Protocol (VRRP) can level the traffic load between them and even serve as a backup. When using Load Balancers along side the Firewalls, the state synchronization between the Firewalls doesn't need to exist because they no longer determine the flow of the traffic. This approach is favourable because it decreases the processing weight put on each Firewall.

The way Load Balancers distribute traffic is defined by algorithms. These algorithms may take advantage of the state synchronization that keep record of active connections and the network topology. For example, in VyOS, upon a request with a new IP address or port, a new sticky-connection entry is made, and subsequent packets from that flow will be routed through the same path. This may pose an issue when a user with bad intentions issues a large number of requests with different IP addresses or ports. An overwhelming number of entries are created in a short period of time leading to reduced perfomance and potential denial of service. If the synchronaziation is being made in the Load Balancers, the Firewalls are spared, but it's still not ideal. Instead, a better approach is to use stateless algorithms. An example is the IP Hash, where the routing is determined by the hash made with some data from the request. Another example is the Round Robin, where the next available Firewall is chosen, following a circular pattern. It is stateless because it only depends on the Firewalls availability at the moment, but it does require that the Firewalls share their state.

To take advantage of stateful mechanisms, a multi-layered defense approach can be taken. Stateless Firewalls are the first wall of defense when entering the network, suspicious sources can be deterred and the system becomes less susceptible to DDoS attacks (inner layers can therefore be stateful).

1.2 Contextualization

The scenario implemented for this project has one layer of protection with redundancy distributed through two Load Balancers and with the Border Router protected, as described in Figure 1.1. The structure in the diagram was provided by the professor of subject.

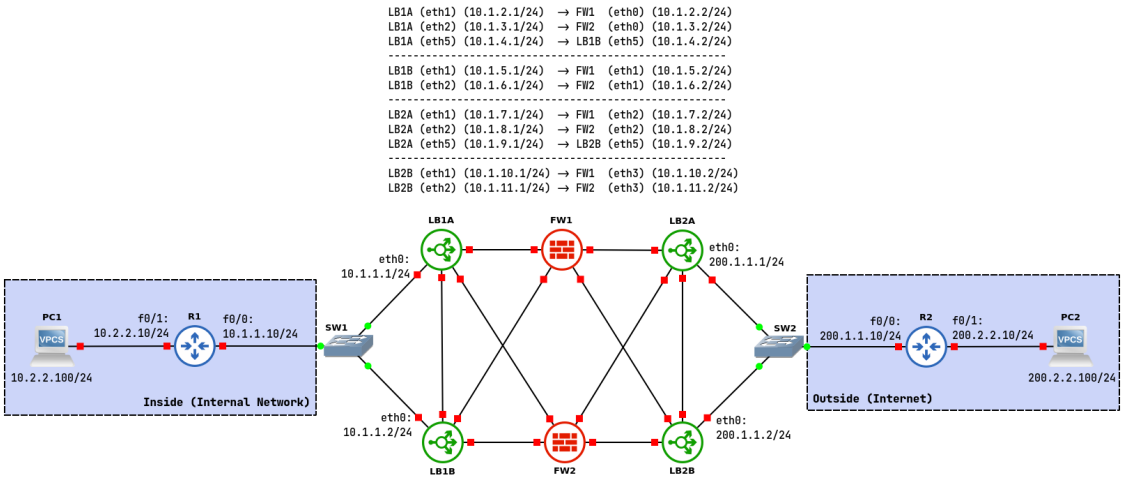

Figure 1.1: Diagram of the implemented network

1.3 Approach

The connectivity between PC1 and PC2 is assured by static routing defined in the Load Balancers, Firewalls and Routers.

To test the connectivity we used UDP pings (Figure 1.2), and registered the packets passing through Firewall 1 (FW1) (Figure 1.3).

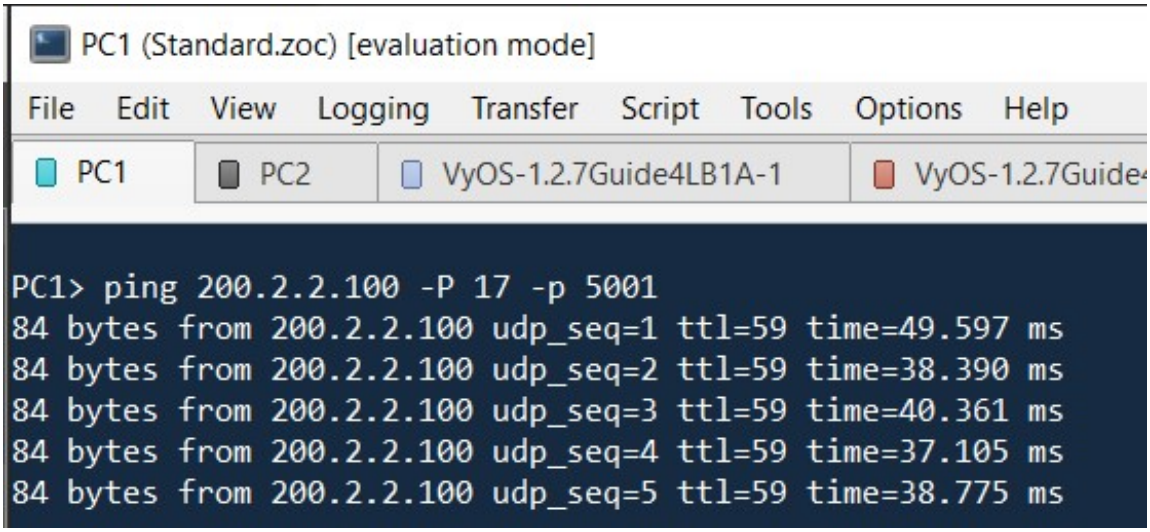

Figure 1.2: UDP pings between PC1 and PC2

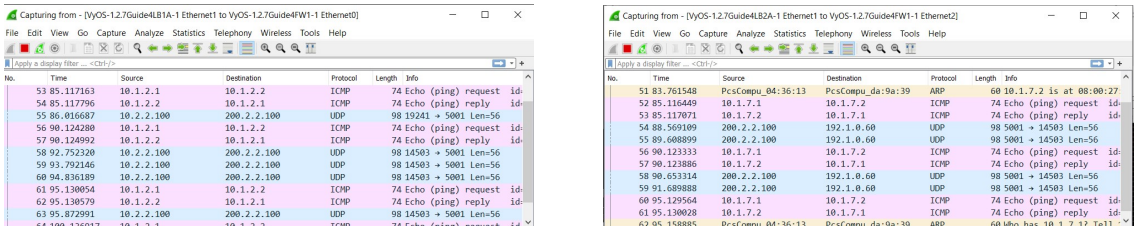

Figure 1.3: Packets going through the Firewall

Other than the Routing mechanism, the Load Balancers implemented VRRP, with a cluster for each side of the Firewalls, and the conntrack-sync service that used those clusters to synchronize the pairs of Load Balancers.

The synchronization can be seen by analysing the connection between, for example, LB1A and LB1B as shown in Figure 1.4

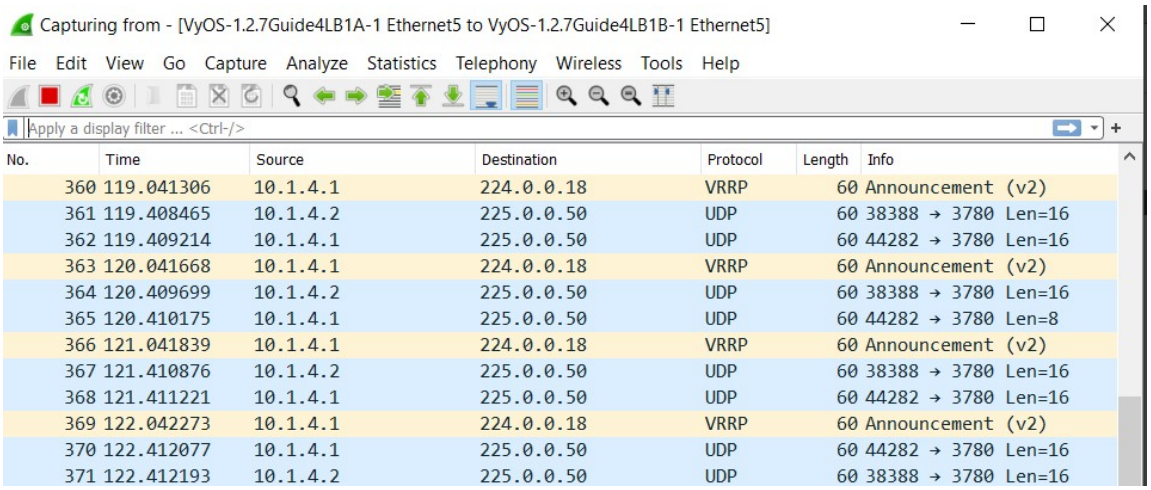

Figure 1.4: Packets between LB1A and LB1B from the synchronization

In the Firewalls, two zone were defined. The *INSIDE*, on the side of the interfaces Ethernet 0 and 1, and the *OUTSIDE*, on the side of the interfaces Ethernet 2 and 3.

Each direction of the flow was given a rule, *FROM-INSIDE-TO-OUTSIDE* and *FROM-OUTSIDE-TO-INSIDE*. The former dictated that only UDP requests to ports 5000-6000 are allowed out and the latter dictated that only traffic belonging to an established connection is allowed in (responses to requests). Figure 1.5 shows that a UDP ping to a port outside the allowed range does reach the Firewall from the inside, but the response is timeout.

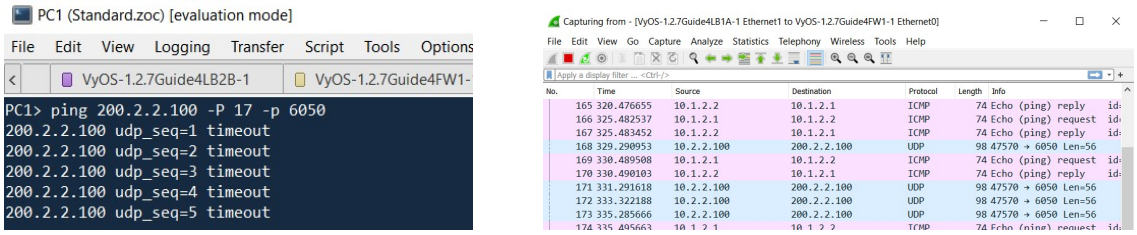

Figure 1.5: Barred traffic that don't match firewall rules

1.4 Policies Definition and Integrated Deployment

To test more Flow Control Rules, it was added a new zone to the project. The scenario now has a DMZ with a server protected by the two Firewalls. This server supports multiple services, such as HTTP (80), HTTPS (443), DNS (53) and SSH (22) and is created using a Linux Virtual Machine provided by the professor. Everything else was kept the same.

The setup has now the structure depicted in Figure 1.6.

To guarantee communication with the DMZ, new static routes were defined in the Firewalls.

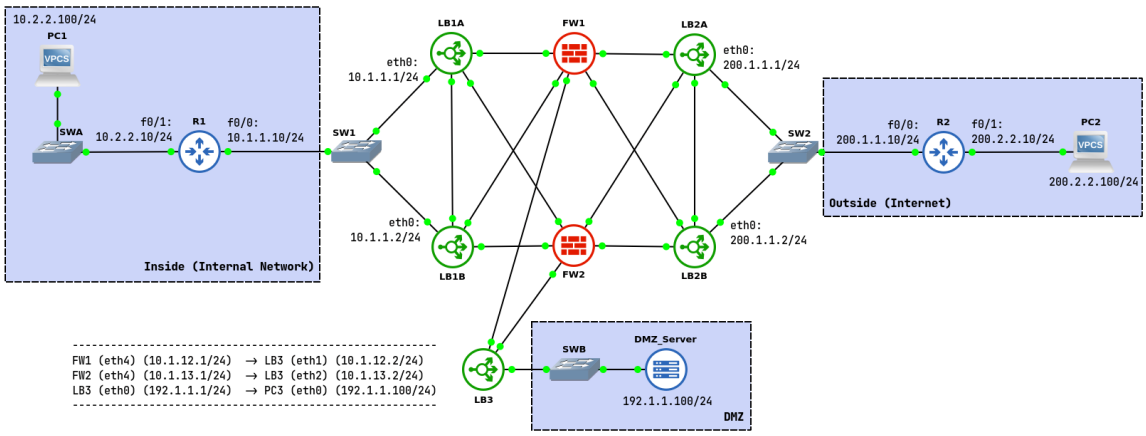

Figure 1.6: Diagram of the network with the DMZ

To limit the internal users and the public to only access the intended services from the DMZ (specified above), the rules *FROM-INSIDE-TO-DMZ* and *FROM-OUTSIDE-TO-DMZ* were created, both accepting TCP and UDP requests to the ports 22, 53, 80 and 443.

To test the connection to a HTTPS service, a dummy HTTP server running on port 443 was setup on the server inside the DMZ using Python's *http.server* library, through the following command:

```
$ sudo python3 -m http.server 443
```

The other services were already running on the machine.

After setting the rules mentioned above in both Firewalls, we got the results shown on Figure 1.7. The ping to the port 5001 represents a request to a service that doesn't exist in the DMZ, or exists but is not included in the accepted ports in the flow control rules.

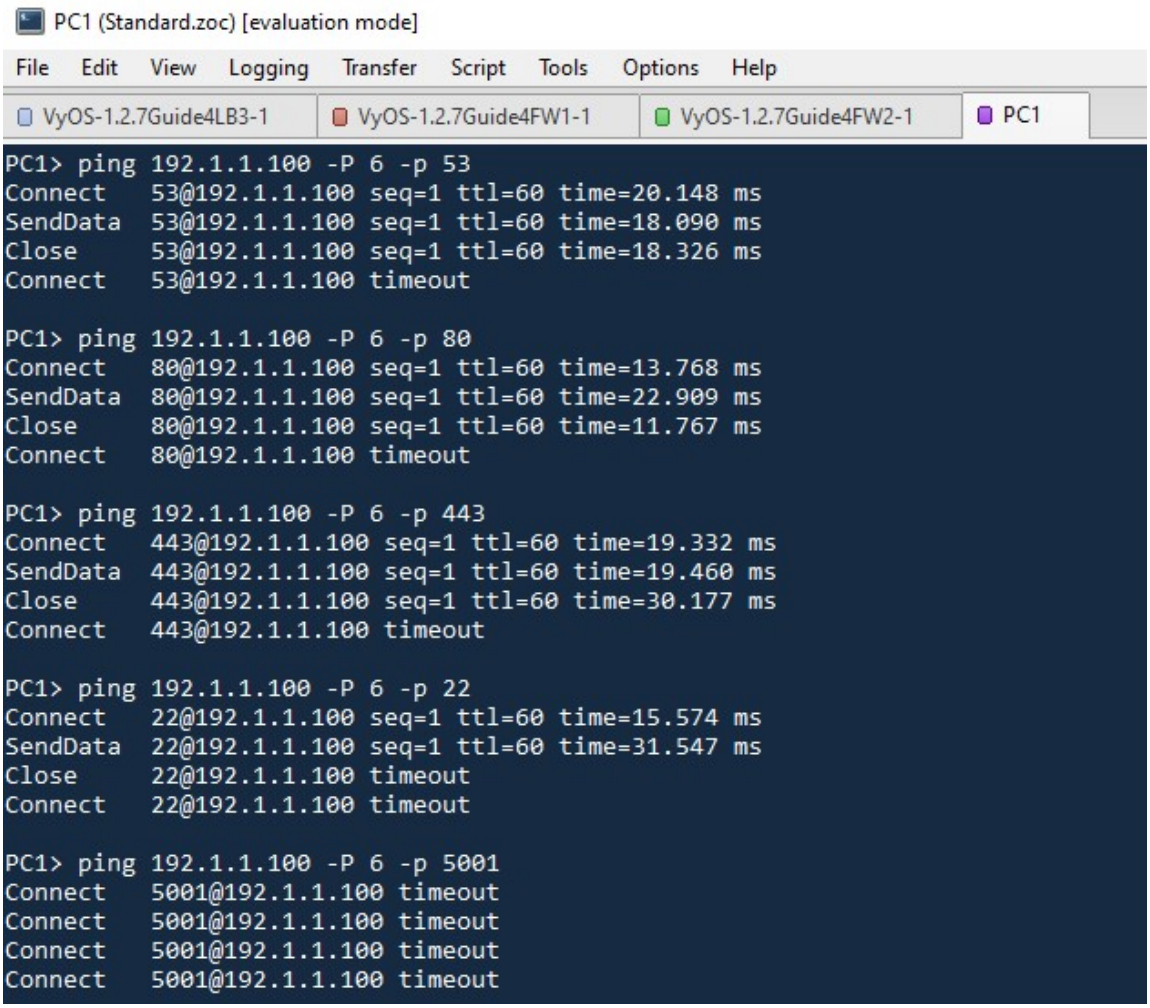

Figure 1.7: Pings to DMZ services

1.5 Extra: Script to create blocking rules

The goal was to create a script, running in a internal server directly connected to the firewalls, that automatically creates the blocking rules in the firewalls during a DDoS. In order to achieve this, a VM was placed in between the two Firewalls, as we can see in Figure 1.8

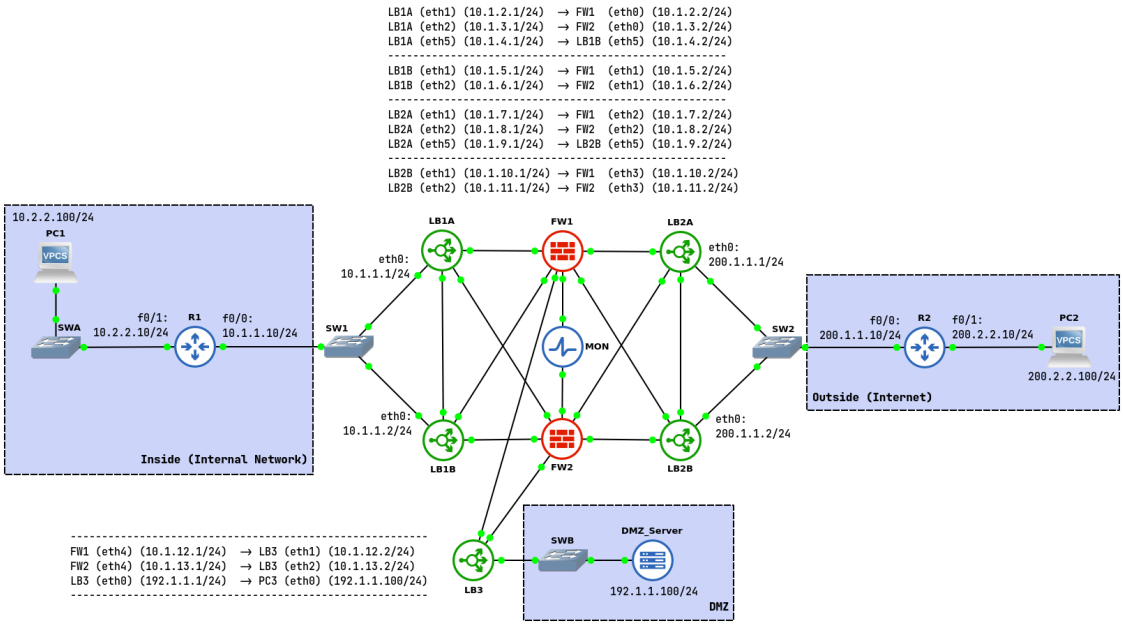

Figure 1.8: Final project overview with the monitoring system in between the Firewalls

The script takes a JSON file with the configuration of the Firewall devices in order to Netmiko library to establish a connection. Then it loads a text file containing the list of IP addresses to be blocked. Then the script checks if the IP address is already blocked and, if not, creates all the necessary rules in the corresponding Firewalls. Figure 1.9 illustrates the script behaviour.

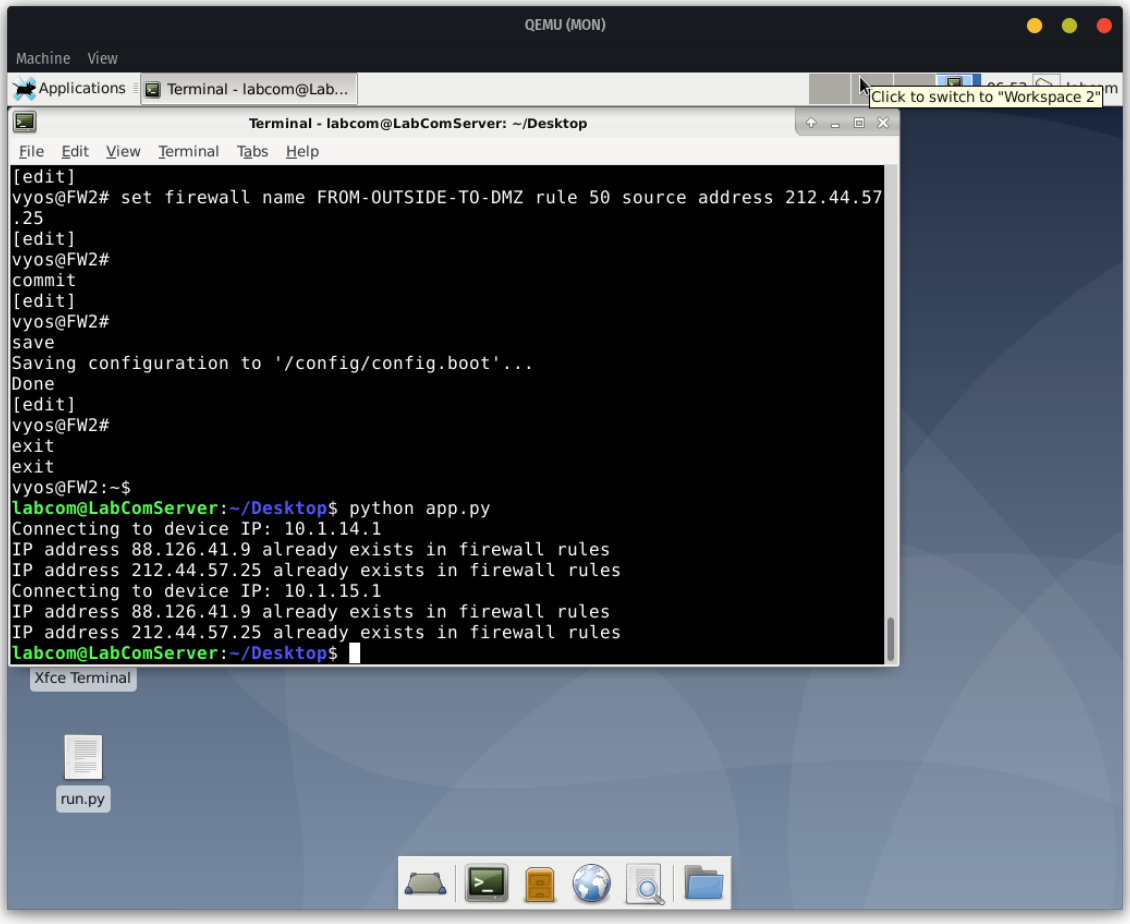

Figure 1.9: Screenshot of the DDOS prevention script in action

Attachments

This report should be accompanied by a text file named **assignment1_commands.md** with the relevant configuration commands for all Routers, Load Balancers and Firewalls used in this project. There will be provided the source code for the DDOS prevention script that creates the blocking rules, under the name **ddos_prevention.py**