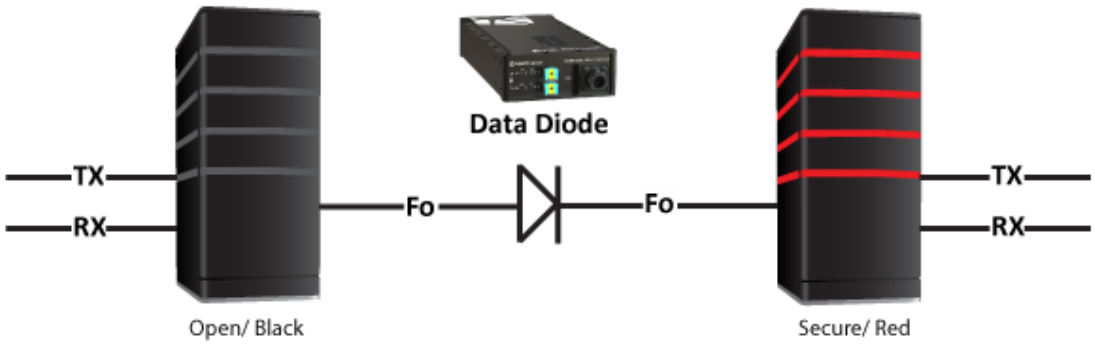
Defend Network Protocols

Data – Diode



Submitters: Zohar Simhon – 211871868
Lidor Keren Yeshuah – 213205230

תוכן עניינים

The main purpose of the code:

The main goal of this code is to simulate the idea of unidirectional communication.

A diode is a physical electric tool which can pass information (electrical signals) only in 1 direction.

Since we can't build our own physical diode, our code simulates a network communication between a client that located outside the organization, that wants to send files to another client, that's located inside the organization without any other way to send data from inside to outside.

Needed files:

The code files:

sender.py, proxy1.py, proxy2.py, receiver.py, file_to_send.jpeg.

Dockers:

this file explains how to work with dockers and how to run the code files.

Video:

A video that shows a full run of the code.

PDF:

This document that explains about the whole task.

Sender.py:

This code file represents someone outside of the organization that wants to send a file (in our case a picture) to someone inside the organization.

This code file has 3 functions:

1. Hash_md5() – reads all the file that we want to send and hashes it by the hash code named "md5" and prints the hash code that we got.
2. Size_file() – calculates the size of the file and returns the size.
3. Send_file_tcp() – our sender has an open socket, whenever someone connects to him and asks for the file, it sends the size of the file and then the file itself (via TCP connection).

Proxy1.py:

This code file represents a proxy server that asks the file from the sender (via TCP connection) and then it will send the file to proxy2 (via RUDP connection).

In this part we will explain the main code of the RUDP connection that we have established.

After receiving the whole file from the sender, we hash the file in order to check we have got the correct file in the correct order (of course we would get it correctly since we have used TCP for this part).

We divide the file by chunks and appending it to a list. Then we open a socket connection (UDP), and we will enter into an infinity loop, in order to handle the incoming requests.

When we get a "syn" message we reply with "syn-ack", sending the size of the file, and waiting for ack.
We start sending the file with "slow start" and the current window size equals to 1. In each packet we send, we start a timer and waiting for an ack message. If we get the correct ack number, then we enlarge the window's size and send the packets according to it. If we have triple duplicate ack then we cut the window size by 2. Also, if the timer runs out and we didn't get an ack message, we initialize the window into 1, and go back to slow start.

After sending all the packets and receiving all the acks we send a "FYN" message in order to close the connection.

NOTE:

We have a congestion control, sliding window, timeout for the socket and threshold. All those things combined together ensure us the **reliability** in our UDP.

Proxy2.py:

This code file represents a proxy server inside the organization that communicates with proxy1 that located outside the organization. The diode idea in the communication between the proxies implemented by RUDP.

In this part we will explain the main code of the RUDP connection that we have established.

We are opening an UDP connection with proxy1 and sending him "syn" message. After getting the "syn-ack" packet which contains the size of the file and sending the "ack", we are ready to ask for the file and get it.

First, we create a list with the size of the file and initialize it to null. Then, for each packet we receive from proxy1 we send "ack" and place the chunk that we have got into its space in the list according to his sequence number.

After receiving the whole file, we get a "fyn" message, sending a "fyn-ack" message, and combining all the parts of the file together.

We are hashing the file that we have got, and we can send it via TCP to the end user.

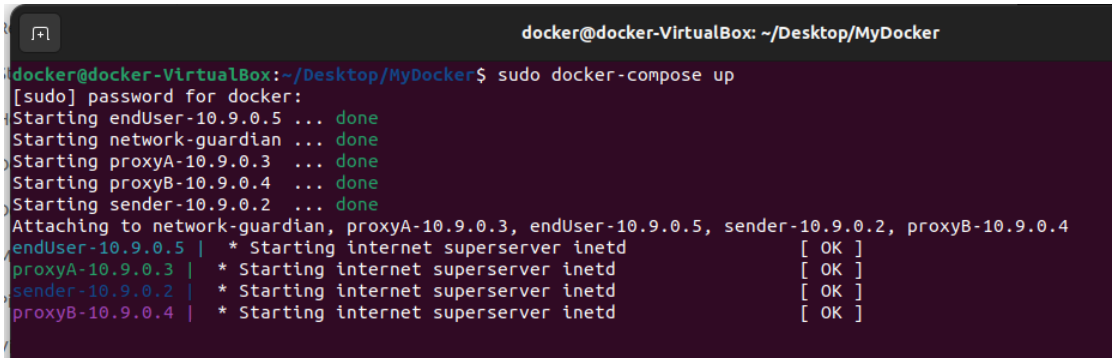
receiver.py:

this code file represents the person inside the organization that need to get the file.

He opens a TCP connection with proxy2 and asks the file from it. After receiving the whole file, he hashes it and then we can compare between all the hashes to check if there were no mistakes in the process.

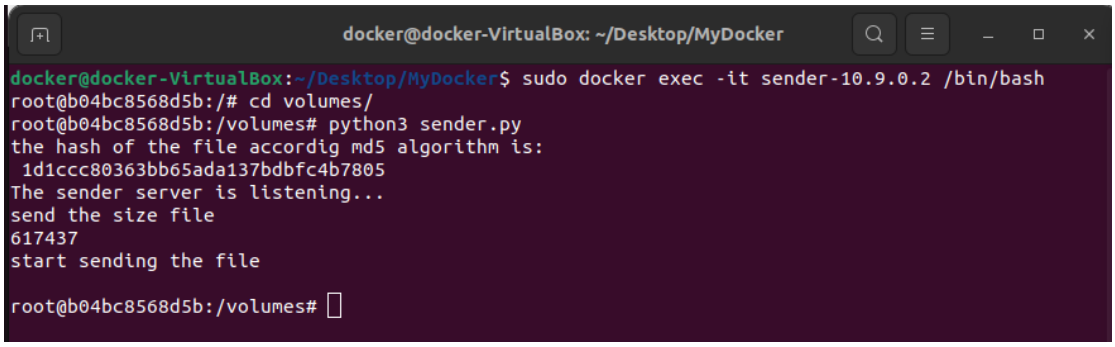
Terminal:

All the dockers are ready to be used.

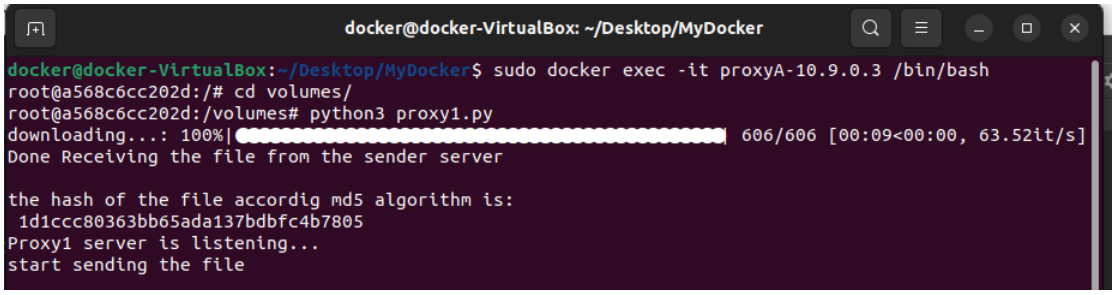


The dockers' information (ignore the first one it's apache2 and it is for the cyber lab).

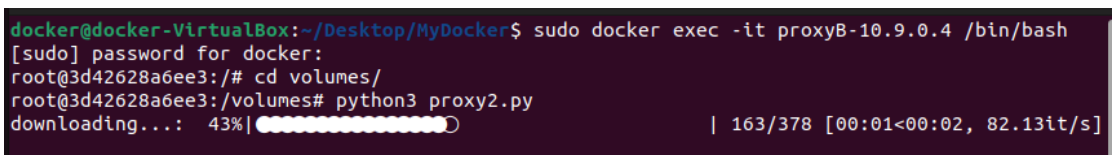
The senders docker.



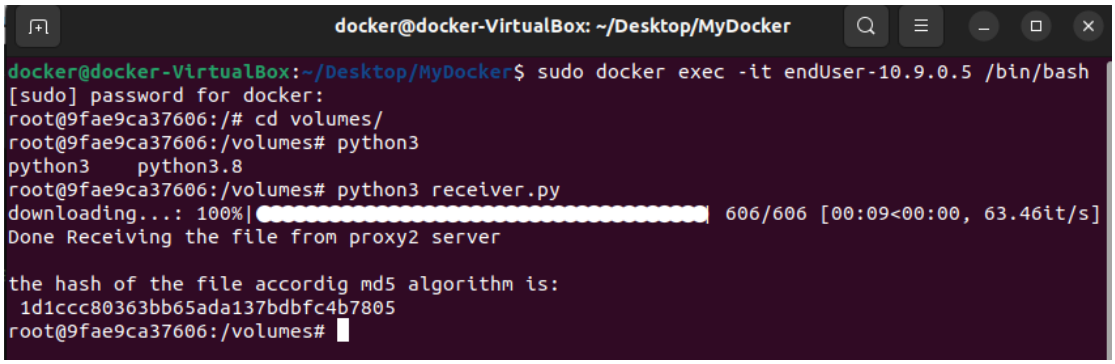
Proxy1's docker.



Proxy2's docker.



Receiver's docker.



Now we can compare between all the hashes and see that all of them are equal.

Answers:

1. The different diodes are:
    a. Light Emitting Diode (LED) - is a semiconductor device that emits light when current flows through it.
    b. Laser diode - is a semiconductor device in which a diode pumped directly with electrical current can create lasing conditions at the diode's junction.
    c. Avalanche diode - is a diode that is designed to experience avalanche breakdown at a specified reverse bias voltage.
    d. Zener diode - is a special type of diode designed to reliably allow current to flow "backwards" (inverted polarity) when a certain set reverse voltage is reached.
    e. Schottky diode - is a semiconductor diode formed by the junction of a semiconductor with a metal. It has a low forward voltage drop and a very fast switching action.
    f. Photodiode - is a light-sensitive semiconductor diode. It produces current when it absorbs photons.
    g. PN junction diode - is a type of semiconductor diode based upon the p–n junction. The diode conducts current in only one direction, and it is made by joining a p-type semiconducting layer to an n-type semiconducting layer.

       Semiconductor diodes have multiple uses including rectification of alternating current to direct current, in the detection of radio signals, and emitting and detecting light.

       (NOTE: there are about 30 different diodes and we wrote here the ones that looked important for us.)

   We have implemented our diode based on the idea of the PN junction diode which conducts current in only one direction.

   In the communication level our diode works on RUDP which allows us to send information (in our case 2mb picture) from proxyA to proxyB and we are only sending **signals** from proxyB to proxyA to ensure that we have got all the information (that what makes our UDP being reliable and it doesn't ruin the main idea of the diode that we can send info only in 1 direction).

2. The problems are:
    a. If we want to send information in the opposite direction we need to construct 1 more diode that will work in this direction which is more complicated rather than just a normal 2 way directional connection.
    b. We don't have a way to reasure that all the information we have send was fully arrived to its destination without any loses of information or in the correct order.

3. An air gap, is a network security measure employed on one or more computers to ensure that a secure computer network is physically isolated from unsecured networks, such as the public Internet or an unsecured local area network. It means a computer or network has no network interface controllers connected to other networks, with a physical or conceptual air gap.

4. In order to create a 2way communication we want to add a new diode that we can send data from inside to an outsider proxy. This yet doesn't assure that the communication will stay secured.
   We want to add some proxies inside the organization each one of them will have different classification.
   For example: main_proxy, proxy1, proxy2 and proxy3. The main proxy is the server that gets the information from outside, proxy 1 to 3 (present inside the organization) are classified and communicate with main proxy. (assume 1 isn't classified and 3 is the most classified).
   When someone wants to request some information from the organization he will send his request to the outsider_proxy and it will send the request to the main_proxy.
   The message will contain array with the next parameters:
   Array[0] – username.
   Array[1] – level of classification that matches the user.
   Array[2] – password that matches the classification level.
   Arrya[3] – the requested data.
   When the main_proxy gets the message he will compare the data that the message contains according to a list that it has if all the data of the message is correct and can be combined together then the main_proxy will pass the message to the correct proxy (1,2 or 3), the proxy will send back the data and then the main_proxy will send the data to the outsider_proxy and back to the sender.
   The idea of adding locks and classification should prevent leaks of data outside the organization and make the communication more secured.