Overlay Networks

João Pedro da Silva Ferreira, Júlio Ismael Barroso de Oliveira, Rui Filipe Ribeiro Freitas, and Tiago João Pereira Ferreira

Universidade do Minho, Guimarães, Portugal
www.eng.uminho.pt

Abstract. Nowadays, a very useful platform is becoming widely used to deploy new services and applications in the Internet: the overlay networks. These networks create a virtual topology on top of physical ones with the goal of providing several different services such as content delivery or gaming. The following article describes our journey to build an overlay network that focus on resolving one of the impairments of today's communications, how can we get the minimum delay possible between 2 users.

Keywords: Overlay Networks · Overlay topologies · Delay .

1 Introduction

As part of the Course on Emulation and Simulation of Telecommunication Networks, we were proposed to develop an overlay application network that allows us to minimize the end-to-end delay between the nodes that compose it.

To carry out the proposal we divided our work in 4 main tasks. The first task was to build a physical topology that would be used to emulate a local area network so we could exchange data between several computers in the same environment. After that, we started to build our overlay network that would run over the physical network previously created. This overlay network was made in Python because was the programming language the group was most used to. After the overlay network was finished we built a game with the goal to test the network environment created by us. With everything working as planned we began testing the network by inputting traffic and running the program on multiple computers to see the adaptability of our ambient.

In this article we will enumerate the steps we took to implement our solution starting with the network architecture where we talk about how we designed our environment where we chose a centralized system with a main server and peers connected to him. After the architecture we will specify the protocol we created by demonstrating our PDU format and the communication primitives. With the solution already thought out, we moved on to the implementation where is shown how the server and the peer programs were built and then made some tests in order to verify the correct functioning of our network.

2 Network Architecture

Firstly, a topology was performed in a CORE environment in order to simulate a physical network with the routing between routers working efficiently. This routing was made with the help of the OSPF protocol which is responsible to find the shortest path between terminals [3]. In order to test the correct functionality of the overlay network, a game was developed in order to control the end-to-end delay in 2 different scenarios, with the overlay network running and without. Regarding the game, it consists of a multiplayer game with the main goal of simulating the popular game Agar.io, in which each player is a ball and aims to grow by eating the opponents. This game was the chosen because in the making of the tests it would be possible to observe the delay visually [1].

A diagram (figure 1) was designed to explain the system architecture. As for the operation of the overlay network, we opted for a central server that controls the entire network with TCP communications with the peers while peers communicate with each other using the UDP protocol. There is a parallelism between this system and an SDN as both are based on a central platform that is responsible for the correct functioning of the network.

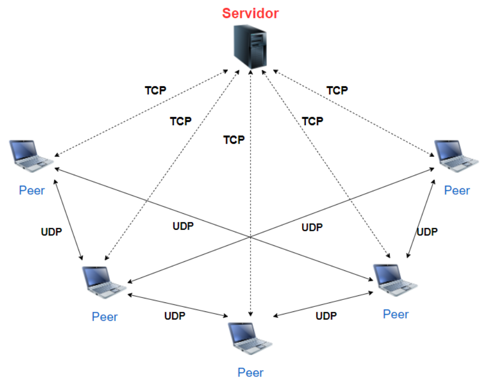

Fig. 1. Architecture diagram.

The system architecture consists of a set of peers and a main server making this a centralized system. In order to implement the proposed architecture, a Python language program was developed for the functioning of peers as well as a program for the operation of the central system. Regarding communication between the peers and the server, communication protocols were developed so that they could communicate efficiently with each other.

3 Protocol Specification

In order to establish an efficient communication between the project entities, several packets were created. The design of these packets had as main objective to respond to the needs of exchanging information between server and peers, allowing the best interpretation possible on both sides. For this purpose, a field corresponding to the type of packet was created, which is identified with only a number, thus occupying only the first byte of the packet. It could occupy less than that but because we chose a byte oriented communication this was the way to go. With 1 byte to represent the type of packet it is only necessary a simple reading of the packet header to know which actions both parts have to perform depending on the type of the message. The remaining of the packet contains the additional information needed as shown in the following sections.

3.1 PDU Format

In order to establish an efficient communication 11 packets were created, being 4 for peer-to-server communication, 4 for server-to-peer communication and 3 for peer-to-peer communication. In peer-server communication (figure 2) there are 2 packets that only differ in the type, which are the connect and disconnect packets, their type being 0 and 1 respectively. These consist on the message type, the IP address of the peer trying to connect/disconnect and the timestamp when the request was made. When the message type is 3, we have the send costs packet in which the peer sends to the server the number of neighbours, its IP and the IP of the neighbors, as well as the cost between them. Finally we have the costs changed packet, which is composed of its type 5 and the peer's IP to warn the server that this peer's costs with his neighbors have changed and it is necessary to recalculate the routing tables.

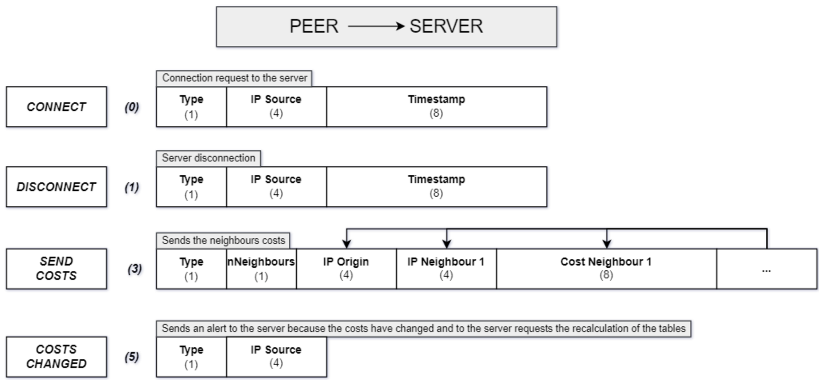

Fig. 2. Peer to server PDU.

As for server-peer communication (figure 3) there is the send neighbor packet with a message type 10 that sends the neighbors to the peer, consisting in the number of neighbors and their respective IPs. The send graph packet with type 11 is sent to all peers connected to inform them about the current state of the graph and this packet contains the size of the graph and the destination IP, next hop and cost for all connections in the network. There is also a ask costs packet with type 12 and the IP of the intended peer and the disconnected peer packet with type 13 and the IP of the peer that was disconnected.

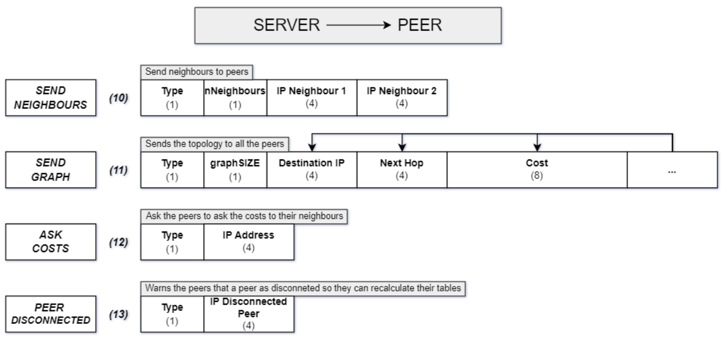

Fig. 3. Server to peer PDU.

Finally in peer-to-peer communication (figure 4) there is a get costs packet with type 20 that contains the IP source of the peer and the timestamp to later be compared. There is also a send data packet constituted by the source IP of the peer, the destination IP as well as the data to be sent. As a response to the type 20 packet, the peer neighbor sends the send cost packet with type 30, its IP and the latency between it and the source peer.

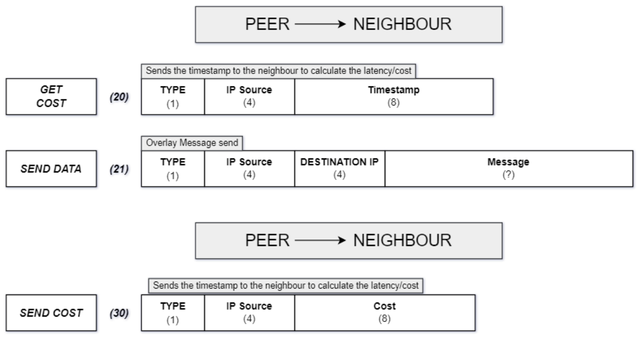

Fig. 4. Peer to peer PDU.

4 Implementation

4.1 Server

The server aims to be the overlay network manager, i.e., it has access to all available users on the network and allows the connection/disconnection of peers to the overlay network. In its implementation, TCP sockets were created so that the peers of the underlay network could have a secure and reliable communication with the server. After a connection is established, the server stores in a database information related to each peer, the id, the peer IP and the status, whether or not this peer is connected to the overlay network.

After this process is finalized, it is then possible to accept connections to the overlay network. When a peer wish to connect to the overlay network, the server assigns it 2 randomly chosen neighbors, changes its status in the database and notifies all the other peers that there has been a new connection established and therefore they must recalculate their end-to-end times so it is possible to build a new topology with that peer taken into account. Hereupon, the server awaits for the costs form all peers, builds the new topology and sends it to each peer so that they can calculate the respective routing table. When a peer disconnects from the network or just shuts down, the server warns every peer in the network so they can remove this one from their topology and build a new routing table, also, if one of the remaining peers in the network, after the removal of the previous peer, sees himself without neighbors its servers responsibility to send new neighbors and the process of building a new topology begins once again. When the server is informed that the costs of a connection between peers have changed, it goes through the same process again, warns all peers and waits for the costs. After receiving the costs from all the connected peers, it build the updated graph and sends it to all peers in order to recalculate their routing tables.

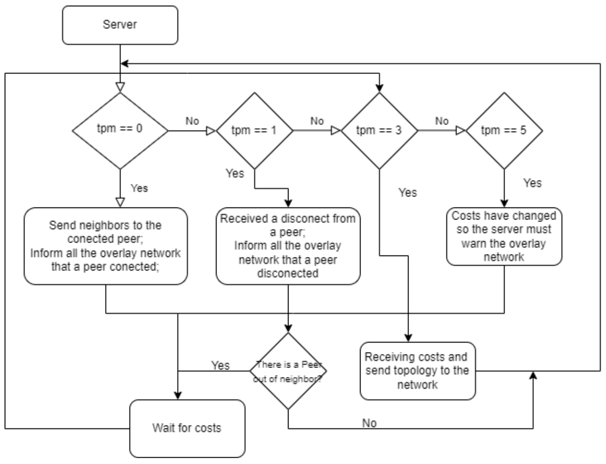
Server
tpm == 0
No
tpm == 1
No
tpm == 3
No
tpm == 5
Yes
Yes
Yes
Yes
Send neighbors to the conected peer; Inform all the overlay network that a peer conected;
Received a disconect from a peer; Inform all the overlay network that a peer disconected
Costs have changed so the server must warn the overlay network
Yes
There is a Peer out of neighbor?
Receiving costs and send topology to the network
Wait for costs
No

Fig. 5. Server flowchart.

The figure 5 show us a flowchart about the communication between the server and a peer and how he would act after receiving a PDU from the peer, like the way explained before.

4.2 Peer

The peer's main goals are to forward the received packets and inform the server of its connections with its neighbors. if a connection gets better or worse, it informs the server so this one can share this update to the rest of the network.

As the cost factor of the network connections is the end-to-end latency, it was necessary to synchronize the clocks of the peers so that they can calculate the latency with the smallest associated error, for this purpose we used the Ntplib library [4]. The Ntp library allows to obtain the time through a request to an online server and with this it is guaranteed that all peers remain synchronized. In order to use this tool in our code we had to implement in our network the ability to access an external DNS. This was done by changing the default DNS route in our machine.

After starting the peer program, two different threads are created, one of them to communicate with the server and the other to listen to the different packets received from the network. In the thread that communicates with the server, a TCP connection is initiated in order to receive and send important information for the efficient performance of the network. In the second thread, a UDP socket is created to allow listening to received packets and be able to carry out the necessary forwarding.

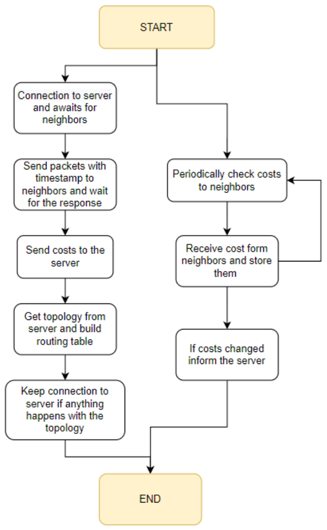
START
Connection to server and awaits for neighbors
Send packets with timestamp to neighbors and wait for the response
Send costs to the server
Get topology from server and build routing table
Keep connection to server if anything happens with the topology
Periodically check costs to neighbors
Receive cost form neighbors and store them
If costs changed inform the server
END

Fig. 6. Peer flowchart.

The figure show us a flowchart about the communication between the peer and the server and with other peers. When a peer tries to connect to the overlay network, it sends a packet to the server with the request to make the connection. After that, the peer awaits to receive further information like the number of neighboring peers and their IPs. From the moment the peer receives his neighbors, it starts to communicate with them periodically in order to obtain the costs associated with each connection. After having these costs stored sends them to the server. Whenever a cost of a connection changes either for the worse or the better, the peer sends its new costs to the server. The peer then receives the complete topology from the server and calculates its routing table. This table is calculated using the dijkstra algorithm [5]. If a peer disconnects from the network, it removes it from the topology previously received and calculates a new routing table.

5 Tests and Results

In order to verify the developed solution was working efficiently it was important to perform some control tests.

5.1 Simple latency tests

First, it was tested the latency between the underlay and the overlay network by sending packets with a timestamp from one computer A to a computer B and the computer B would subtract his time to the time received and would have the latency between the 2 computers since they were synchronized with the same clock. The realization of this test was made with the assistance of a software that was used to emulate a real environment where was injected some traffic in the network so the results could be as close to reality as possible. In the next table we show the average latencies from the tests we performed.

Table 1. Average latencies.

Underlay	0.00792 s
Overlay with 2 neighboring peers	0.00526 s
Overlay with 6 neighboring peers	0.01623 s
Overlay with 6 non-neighbor peers	0.0221 s

In the previous table we can see 4 values for 4 different situations. In the first situation the test was made in the underlay network between 2 peers in both edges of the topology and by comparing that value with the second situation where we used the overlay network between the same peers we can actually see an improvement of about 50%. By adding more peers to the network as tested in situation 3 and 4 it is possible to observe that both average latencies increased substantially. This happened because in the third situation with the presence of 6 peers the traffic was higher and that contributed to the higher latencies. In situation 4 the test was made between 2 peers in a 6 peer network where the 2 peers being tested were not neighbors. The result indicates an increase in latency is proportional to the increase in the number of peers and the connections between them. To be noted that this tests were made with the use of iperf technology [2] where it was inserted a certain amout of traffic onto the network in order to emulate a real life situation.

5.2 Network Adaptability

The second test was performed to see which network would have a better adaptability to some outside problems like connection delays or even a connection break. In this test some delay was added to a connection between 2 peers far apart from each and not neighbors in the overlay network so it could be possible to see the adaptability of our solution. The results are shown in the next figure.

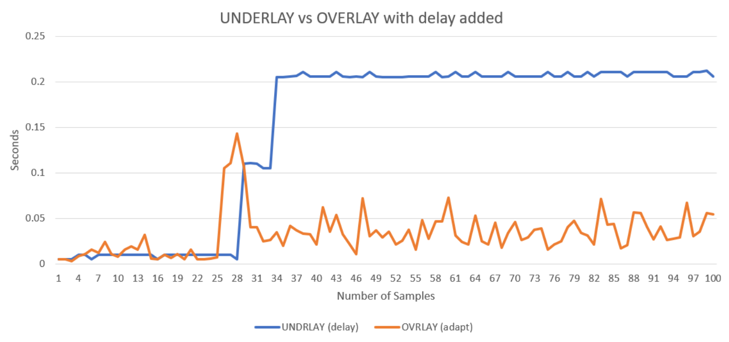

Fig. 7. Underlay vs Overlay with delay added.

With the observation of the previous figure we can easily see that the underlay network does not adapt to the delay added which indicates that the traffic is sent through the same path. On the other hand, the overlay network, despite the initial spike upon the delay being added, it easily adapts by changing to better paths and still having a decent latency.

5.3 Overload test

In this third and final test the goal was to see how would the overlay network react when the network was flooded with peers. It started with only 2 peers connected and every 30 seconds 1 more was added until the total of 11 peers was reached. This test was performed in every single computer connected to the network and each one was sending packets with timestamp to each other and the latency was being stored in a text file. The overlay network was designed to have a maximum of 2 neighbors per peer and a threshold of 4 connections. This test was also made so it would be possible to observe that when the number of hops between the peers increase the latency increase with it. With the addition of peers in the test, the peers received more connections and to give a visual demonstration of the test, the topology is shown in the next figure with all the links created by the overlay network procedure.

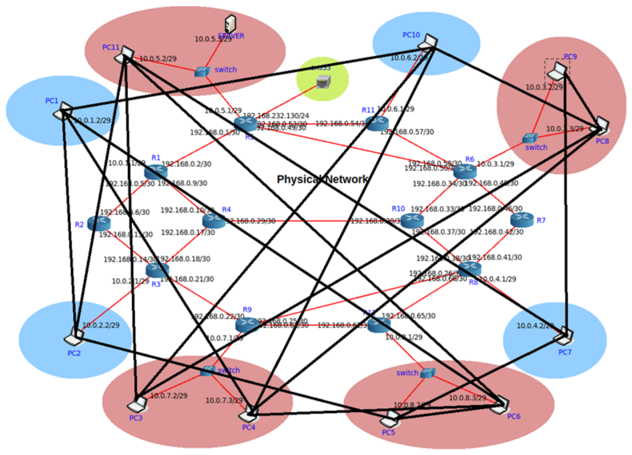

Fig. 8. Topology.

A graph was made to each computer to see the correlation between the latency and the increase in the number of peers as well as the hops between them. In the next figure we show the timeline of one of the first peer connected to the network.

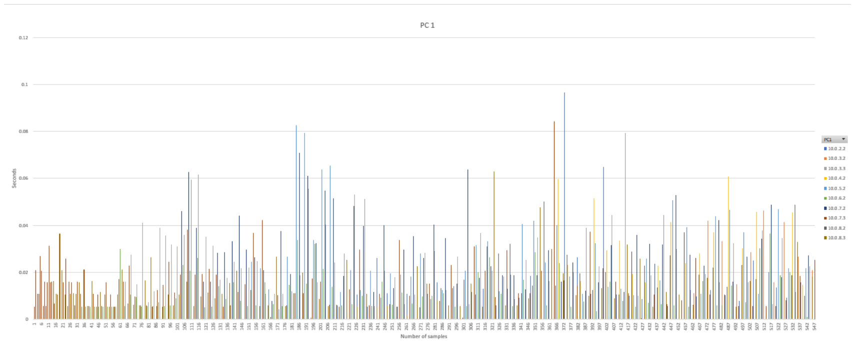

Fig. 9. Overload test - First peer connected.

With the visualization of the previous graph it is possible to conclude that in the solution implemented when the peers are only a few and they are directly connected the latencies are very low but with the addition of peers and hops between them the time that it takes to send a packet from one computer to another starts to increase.

In the next graph it is shown the last computer added to the network where we can see different values from different computers. This test was made to see the difference in latency between 2 peers directly connected and peers connected via other peers.

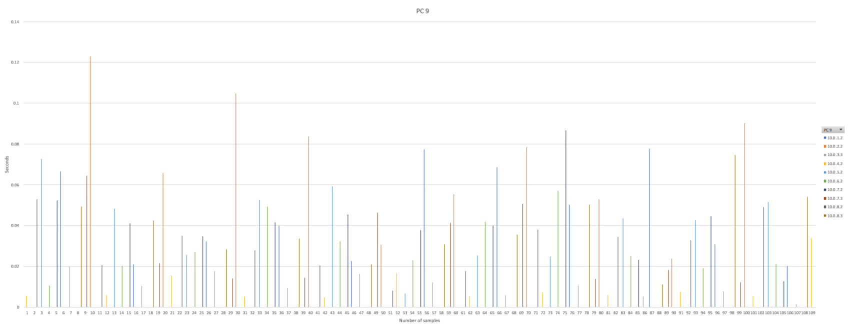

Fig. 10. Overload test - Last peer connected.

If we take a closer look at the values on the graph and compare for example the PC 7(10.0.4.2) which is directly connected to PC 9(PC this test refers to) and PC 2(10.0.2.2) which is minimum 3 hops away from PC 9 we can clearly see that the latency between the last scenario is substantially higher compared to the first scenario where they are directly connected. This indicates that the latency becomes higher with the increase in the number of hops between peers.

6 Conclusion

Upon finishing the development of the project we can say that we are satisfied with the final result and with our performance since we met the proposed objectives. Regarding the problems we encountered along our journey, those were surpassed despite the initial unknowing of the used technologies. In the making of this project the task it took most of our time was the development of the server and peer program since these had some level of complexity. It should also be noted the initial problems the group had with the CORE program when trying to secure an exterior connection with the internet as well as search ways to start outside programs in this environment. After a semester of hard work and dedication in the search of what would be a better solution for the implementation of the overlay network, the group concluded that all goals were accomplished. Since the emulation of the physical network all the way to the development of the overlay network and the implementation of an application in both environments, everything was successfully achieved.

References

1. Agario Implementation, https://www.techwithtim.net/tutorials/python-online-game-tutorial/client/
2. Iperf technology, https://chanind.github.io/2020/05/15/network-bandwidth-stress-testing-iperf.html
3. OSPF Fundamentals, https://www.geeksforgeeks.org/open-shortest-path-first-ospf-protocol-fundamentals/
4. Ntplib Library, https://pypi.org/project/ntplib/
5. Dijkstra Algorithm Implementation, https://www.bogotobogo.com/python/python-Dijkstras$_{S}hortest_{P}ath_{A}lgorithm.php$