PRELIMINARY PROJECT – ECOPR

PATRYCJA KRANKOWSKA 310692

MARCEL PIOTROWSKI 303108

ALEKSANDER SZWERYN

PRELIMINARY PROJECT

The aim of the project is to design a developer environment to conveniently investigate behavior of a protocol.

Before we designed the system, we decided that we should get some common knowledge about the well-known protocols to see what exactly is needed to be added to our project design.

We decided to take into consideration DHCP, TCP and ARP.

DHCP PROTOCOL DESCRIPTION

DHCP is a protocol which is of type client-server. Communication between the client and server is done by sending request and getting the response.

The client sends a broadcast, and the server sends unicast to client.

DHCP connection has 4 stages:

- DISCOVER – it is a request type message sent by client. It is sent to discover/identify DHCP servers
- OFFER – it is a message of respond type from server which consists of available IP addresses and options.
- REQUEST – it is a message of request type which consists of request the IP address from the server.
- ACKNOWLEDGE – server acknowledges the IP request and completes the initiation cycle.

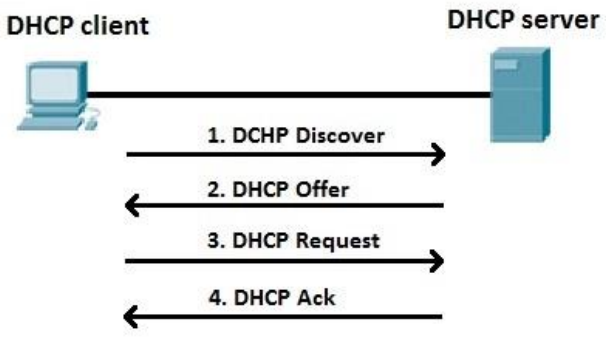

Fig 1. DHCP design schematic

TCP PROTOCOL DESCRIPTION

TCP enables application programs and computing devices to exchange messages over a network. It is. designed in such a way to send packets across the internet. TCP organizes data in a way that one device is a server and the other a client. There has to be a connection between a source and destination.

The connection of TCP is established by using 3-way handshaking. It means that we have 3 main stages to be followed:

- Client sends the segment with its sequence number
- Then the server as a respond sends its segment with its own sequence number and as a addition it adds acknowledgement sequence.
- When a client receives the acknowledgement of its segment, then it sends the acknowledgement to the server.

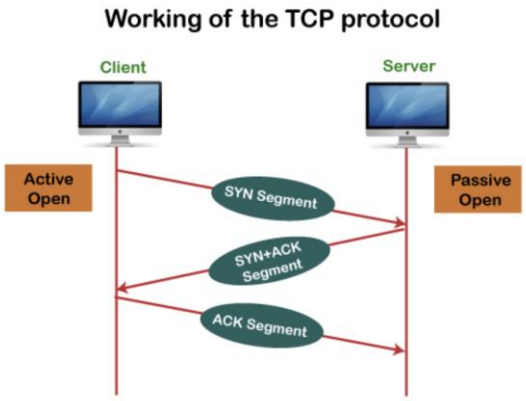

Fig 2. TCP design scheme

ARP PROTOCOL DESCRIPTION

ARP is a protocol that establishes the connection of ever-changing IP address to a fixed physical machine address – MAC address within a local-area network LAN.

MAC address can be described as data-link layer. It establishes and terminates a connection between 2 physically connected devices and that is why the data transfer can take place.

We know that when a new device joins the LAN, it gets its unique IP address.

Terminating ARP is really easy. When the source wants to find out the MAC address of the destination device, first look to see if the address is present in an ARP cache. If it is present, then it will use the MAC address from the cache.

When it is not present then the source device will generate ARP request which is a broadcast signal within LAN. The request contains MAC and IP address of the source, destination IP and MAC address of the source is left blank. Broadcast message will be received by all devices within a LAN. If the IP address does not match the device then it will be dropped. If the device whose IP matches the one from the request, then it will send an ARP reply message which will contain the destination MAC address.

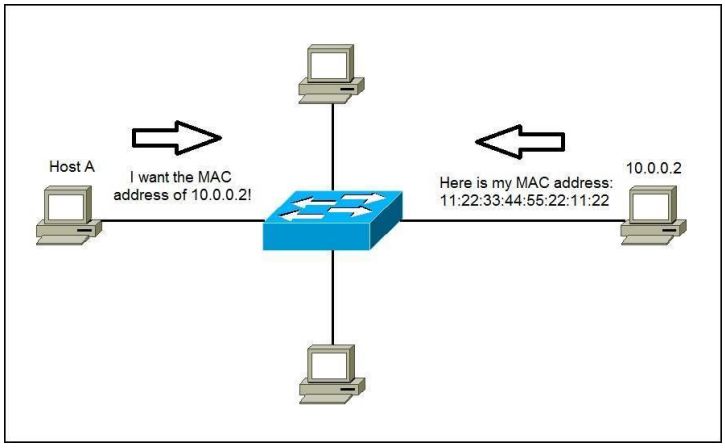

Fig 3. ARP design schematic

FINITE STATE MACHINE PROCEDURE AND DETAILS

FSM (Finite State Machine) describes the protocol by explaining all possible states in which protocol can ever possibly be in. Each state describes the events which can occur and what actions should be taken as a response and how the transitions should look like. FSM are commonly used in reactive systems and describes the behavior of the entity. Each entity has a particular state assigned at the given time.

Protocol usually starts in the beginning state when we run it. Then it moves to other states as a response to for example different output, actions taken by the user or other circumstances.

FSM allows to prove the properties of the machine using graph theory. First of all, we have to design the transition graph and then redraw it to, for example Mealy FSM format to have a more general view of the automaton.

From the task, we know that our FSM automata consists of individual entities. Each automaton is modelled as a separate object. The system should be able to deal with at least 3 protocol entities, but it can just run on any non-zero number of entities.

Considering all above protocols, we noticed that devices no matter if they were servers or clients can send a signal and receive signal. What is more, when one device transmits a signal then it waits for the response for some period of time so this leaded us to design this simple algorithm for individual entity.

The actions which can be taken by the entity are:

- Send a signal
- Receive the signal
- Wait for the signal
- Wait for the response

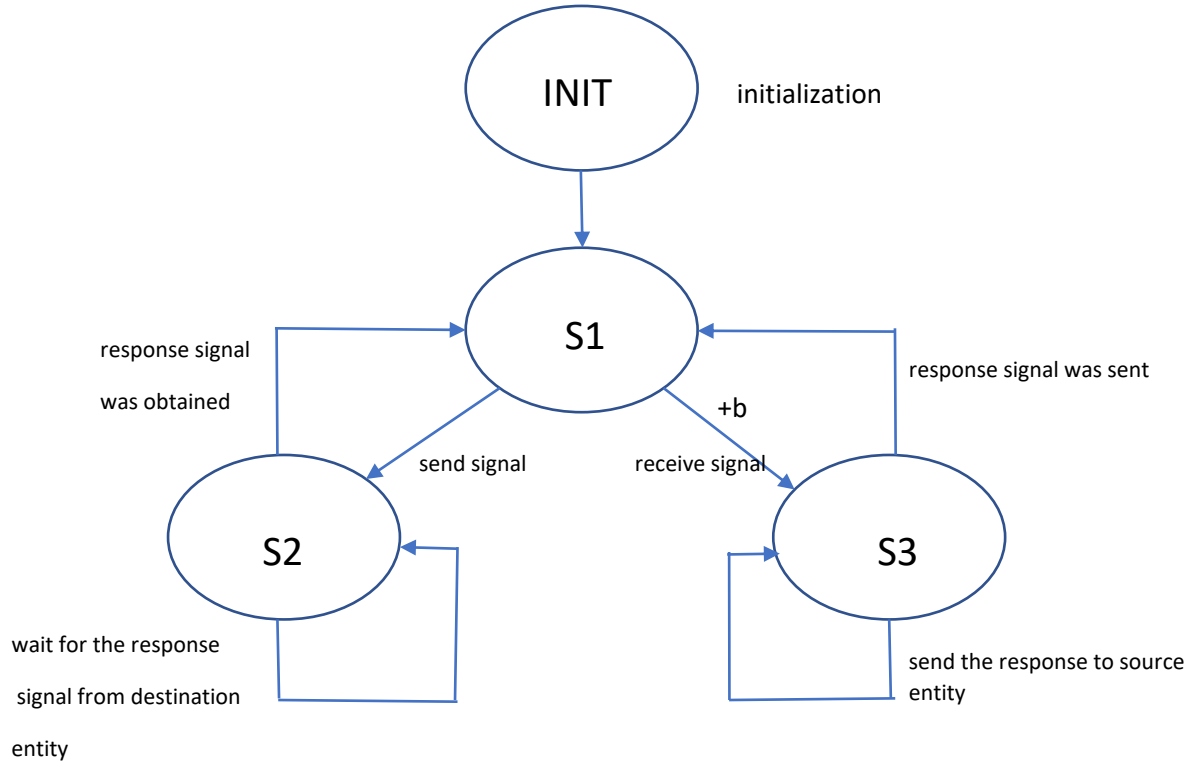

Fig 4. Algorithm how the individual entity works

The above figure shows the FSM for single entity. When we add more entities to the system then we will notice that when one entity wants to transmit the signal, then the other entity has to be in state s1 to be able to listen for the incoming signal. There are also 2 wait loops included for response messages if the signal was received correctly.

ADDITIONAL INFORMATION TAKEN FROM THE TASK DESCRIPTION

The protocol can run using one of two modes. Main differences will be seen when entity description will contain some logical choices using simple “if” commands:

- **Automatic** – all steps will be decided automatically, no commands will be omitted.
- **Step by step** - for every step, we can provide some decisions using simple UI. For example, if the entity is about to send some message, we can stop it, and omit the state.

The mode will be chosen before running the protocol.

In automatic mode, we also have to consider the following case, when no entity cannot do anything. Then we should get the message to the user in the console. This situation should be considered a fault.

PROTOCOL TEXTUAL NOTATION REFERENCES

We are aiming to provide a textual notation for writing protocol entities as finite state machines in an efficient and uncomplicated way. Good practices are taken from programming languages such as Structured Text and Python. Two essential commands are given: SEND and LISTEN. Both take one argument written in quotes. Simple example of DHCP Protocol client written in our notation is provided below:

```
0:
      WAIT 1s;
      goto 1;
1:
      SEND "DHCP DISCOVER" to 3;
      goto 2;
2:
      LISTEN "DHCP OFFER" from 2;
      goto 3;
3:
      SEND "DHCP REQUEST" to 1;
      goto 4;
```

```
4:
      LISTEN "DHCP ACK" from 2;
      goto 1;
```

The above notation shows the easiest case when there are no if statements, we just listen to the transmission from one entity and send the signal to another entity.

The notation “to 3” means that we send the signal to entity denoted by 3. Notation “from 2” means that we would like to listen to the transmission from entity 2.

Our notation will also include the the possibility to send and listen not only from one entity, but there will also be introduced broadcast transmission.

When we go further through the code, we also introduce the part of the code which is responsible for going into the next state and it is denoted by the syntax “goto 1” - go to state 1.

We also took into consideration the case within one state, we have multiple transitions. When we consider the step-by-step mode, then the movement to the next state is easy because automaton does what the user wants if the conditions for the transitions are met. In automatic mode the situation is more complicated. The decision about what to do should be made randomly.

```
1:
      SEND "DHCP DISCOVER" to 3
          goto 2;
      LISTEN "DHCP DISCOVER" from 3
     goto 3;
```

Notation considers messages as strings (text in quotes) which will be read after a command is written).

MEMORY MANAGEMENT

Because of the possibility that multiple messages can be sent to some entity, our application will take care of queuing for the signals from other entities. The queue has the for of the stack. When we would like to read from the queue, then we consider it in the following way. We can just read the message which is the first in a queue. We check if it is from the entity which we would like it to be and then we have to find out if the content of the message is the desired one. Content of the message in the queue has to be the same as in the statement.

We assume each connection of the entities has each queue and its size should be defined by the user. The size of the queue is infite and each user can say a different number.

When there is a situation when more signals want to be in the queue than we have space, then it should be considered as a protocol fault – unboundedness.

When there are no messages saved in the queue and we would like to read from them, then we can notice some faults connected to deadlocks.

Example queue notation (list):

[

1. {name: “entity1”, message: “DHCP REQUEST”}

2. {name: “entity2”, message: “DHCP REQUEST”}

3. {name: “entity3”, message: “DHCP REQUEST”}

]

//Because of the possibility that multiple messages can be sent to some entity, our application will take care of queuing for the signals from other entities. The queue of messages will be implemented using Python’s list, so that all entries will be numbered and taken care of in a proper order. Elements of the list will be dictionaries which will contain names and messages.

BIBLIOGRAPHY

https://networkacademy.pl/dhcp-czyli-o-automatyzacji-konfiguracji-sieci/

https://www.techtarget.com/searchnetworking/definition/DHCP

https://www.javatpoint.com/tcp

https://afteracademy.com/blog/what-is-arp-and-how-does-it-work/

http://tcpipguide.com/free/t_TCPOperationalOverviewandtheTCPFiniteStateMachineF.htm

http://ajbasweb.com/old/ajbas/2011/October-2011/956-960.pdf

https://www.ee.columbia.edu/~nick/EE6777/Chapter.04.Specification.pdf

https://ieeexplore.ieee.org/stamp/stamp.jsp?tp=&arnumber=342810

https://community.wvu.edu/~hhammar/rts/adv%20rts/statecharts%20patterns%20papers%20and%20%20examples/yacoub-ammar%20fsm%20ch%2010.pdf

mfrszpiotro/dhcprotocolism: Visualize DHCP Connection with our simulator (github.com)