P4PI L2 Enhanced Switch

Final Report

Project in computer networks

Winter 2021-2022

Team Members: *Hussein Abu Jabal, Haneen Jeries*

Instructor: *Eran Tavor*

Table of Contents

Introduction

Background

With the expansion of the Internet, the security and privacy of data transportation throughout the internet became crucial.
For instance, if an employee wants to connect from his home to the network resided in the company he works in, the connection must be secure and only those inside the company can understand what goes within these communications.

Overview

In this project, we aspire to develop a private encrypted network (Tunneling) using 2 P4PI devices.
A tunneling protocol is a communications protocol that allows for the transferring of data from one network to another. It involves allowing private network communications to be sent across the Internet securely through a process called encapsulation. Specifically, in this project, we will use PPP (Point-to-Point Protocol) over UDP as a tunneling protocol, and the packets will be encrypted using AES algorithm.

A standard VPN works by establishing a secure tunnel between a **client** and a **private network.** Whereas in our project, the VPN is created by establishing a secure tunnel between a **local network** and a **private network,** without clients intervening. The 2 P4PI devices serve as the two endpoints of the tunnel.

Goals

- ❖ Learn and research PPP protocol
- ❖ Implement a simple LAN using 2 P4PIs switches
- ❖ Implement PPP using P4
- ❖ Implement and test sending PPP packets over UDP
- ❖ Implement AES algorithm and verify encrypting/decrypting PPP packets
- ❖ Verify the product correctness and weak points
- ❖ Define KPI and analyze performance

Theoretical Background

The tunneling between the two switches uses PPP protocol over UDP.

PPP

What is PPP and why do we use it?

PPP is a data link protocol, resides at layer 2 of OSI model.
PPP is used to encapsulate layer 3 protocols for transmission across serial links only between 2 devices connected on a link.

PPP is a layered protocol that has three components:

1. An **encapsulation** component that is used to transmit datagrams over the specified physical layer.

2. A **Link Control Protocol (LCP)** to establish, configure, and test the link as well as negotiate settings, options and the use of features.

3. One or more **Network Control Protocols (NCP)** used to negotiate optional configuration parameters and facilities for the network layer. There is one NCP for each higher-layer protocol supported by PPP.

- **States description:**
  - **Dead**

    The link always begins and ends with this phase.
    When an external event indicates that the physical layer is ready, PPP will proceed to the next phase: link establishment

  - **Establish**

    Relies on LCP packets to establish the connection.
    The packets that are exchanged contain configuration options and the nodes will negotiate to determine settings such as: maximum transmission unit, authentication protocol PAP or CHAP etc.
    Unspecified options are set to default.

  - **Authentication**

    Requiring a peer to authenticate itself before allowing network-layer protocol packets to be exchanged.
    Authentication is not mandatory, however if it's supported then it should take place as soon as possible after the link establishment phase.

- **Network**

  Determines what network layer protocol will be used and how will it be configured. Network control protocol (NCP) packets are sent to negotiate the necessary settings. Once the network layer protocol is configured then network layer datagrams can now be sent over the established session.

- **Terminate**

  PPP can terminate the links at any time.
  LCP is used to close the link through an exchange of Terminate packets. after the termination phase the PPP proceeds to the Link Dead phase.

- **Connection establishment:**

  The connection is held between two different sides (2 P4PI switches).
  The process of establishing a PPP connection:
  1. Side A sends a configure request to side B
  2. Side B sends a configure request to side A
  3. Side B sends a configure acknowledgement to side A
  4. Side A sends a configure acknowledgement to side B

- **Packet Header Layout**

flag	protocol	payload	FCS	flag

Flag: marks the beginning or the ending of the packet
Protocol: determines the type of data to be sent in payload (LCP/NCP)
Payload: actual data
FCS: used for error detection: CRC

PPP over L2TP

PPP over Layer two tunneling protocol

- **What is L2TP?**

  L2 tunneling protocol is used to carry almost any Layer 2 data across the network. These Layer two frames are PPP frames, the L2tp enables the PPP frames to be encapsulated within network packets and sent through a Layer 3 network.

  L2TP consists mainly of two components:
  1. LAC – L2TP Access concentrator.
  2. LNS – L2TP Network server

  L2TP is established between the LAC and the LNS.

- **LAC and LNS:**

  <u>LAC:</u> acts as one side of the tunnel endpoints, PPP packets are forwarded from the remote connection (the client) to the LAC, the PPP packets are encapsulated and then forwarded from the LAC to the LNS over the l2 tunnel protocol.

  <u>LNS:</u> acts as the other side of the tunnel endpoints, it acts as the termination point for all the L2TP sessions coming from various LACs.
  The LNS is also the termination point of the PPP session.

  The LAC and LNS have no awareness of the data contained in the PPP packets.

- **Types of messages:**
  L2TP mainly uses two types of messages, control messages and data messages.
  the messages are passed over separate channels.

  <u>Control messages:</u> are sent over the control channel, these types of messages control connection management, call management and error reporting.

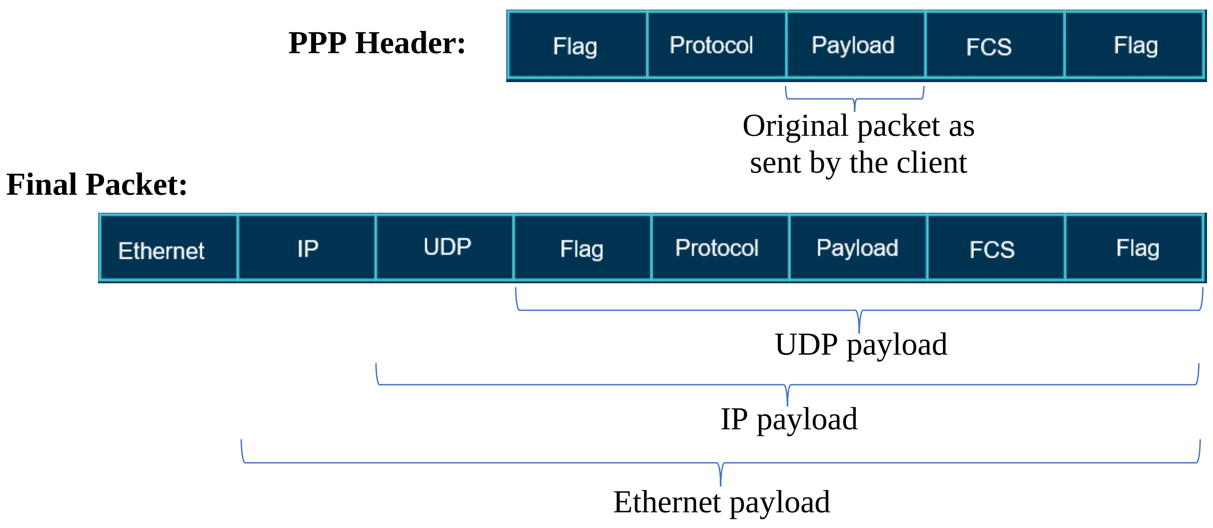
  <u>Data messages:</u> are sent over the data channel, these messages are used to encapsulate the PPP frames that are sent into the L2TP tunnel.
  L2TP uses UDP port 1701, the l2tp packets are encapsulated within the UDP datagram.

- **L2tp Session:**
  The control session <u>must</u> be established before sending the first PPP packet through the tunnel. After successfully establishing the control channel a session is established for each PPP connection between the LAC and LNS. For incoming packets, the LAC initiates the session, and for outgoing packets the LNS initiates the session.

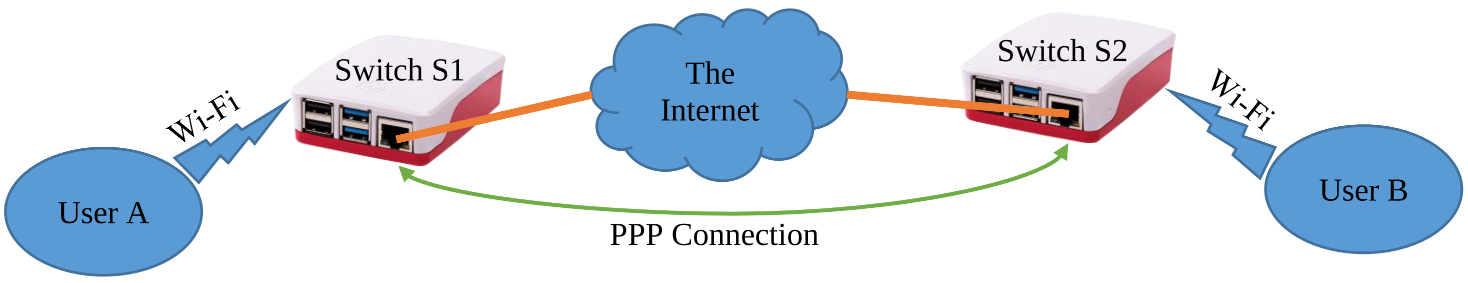
- **The flow:**
  The user wants to establish a PPP connection with the private network. Therefore, the user needs to establish a PPP session with the LAC, and the LAC requests the LNS to accept the session (creating the l2tp tunnel), and it will then encapsulate all the PPP packets and forward them to the LNS. LNS then decapsulates the packets and passes the traffic over to the private network.

Sending a PPP packet

Every PPP packet mentioned in the PPP section, in any PPP state, including data and control messages, must be sent throughout the internet.
A certain problem arises in the process of sending the packet. The routers throughout the routing path do not recognize the PPP protocol, and since it is implemented in Layer 2, the routers don’t know how to decode the headers of the packet and to get the packet destination.
Our solution is to wrap the PPP packet using UDP and to send it in the internet on port 1701, as shown in the figure below. This will allow the routers in the routing path to deliver the packet to the correct destination.



The journey of a packet



User A is connected to switch s1, and wants to send a packet to user B who is sitting in a private network which is connected to switch s2, assuming that this is the first packet sent between A and B.
If the PPP connection is already established skip step 2.

1. User A sends a packet to user B.
2. Switch S1 decodes the IP header of the packet, and notices that the packet is meant for User B. S1 establishes a PPP connection with switch S2, according to the process described in the PPP section.
3. S1 encapsulates the packet using PPP, and then encrypts the PPP packet using AES algorithm, and sends the packet to S2 using UDP protocol on port 1701.
4. Switch S2 receives the packet, decrypt it from using AES, and then decapsulates the PPP packet, and sends the packet (PPP payload) to User B according to his IP address which is found in the IP header of the original packet.
5. User B receives the packet.

P4Pi

P4Pi is a low cost, open-source platform for computer networks teaching and research.
The platform is based on the Raspberry Pi board and uses the P4 programming language and the open-source T4P4S P4 Compiler and open-source hardware.

P4 language

P4 is a programming language for controlling packet forwarding planes in networking devices, such as routers and switches. P4 is a domain-specific language with a number of constructs optimized for network data forwarding. P4 is distributed as open-source, permissively licensed code, and is maintained by the P4 Language Consortium, a not-for-profit organization hosted by the Open Networking Foundation.

Setting up the environment

Required Equipment:
We require 2 pieces of each of the following equipment.

- Raspberry Pi 4 Model B – 4GB.
- Raspberry Pi 4 USB-C Power Supply.
- Micro SD Card 16GB or 32GB
- Micro-HDMI to HDMI Cable
- Keyboard (USB)
- Monitor which supports HDMI
- Ethernet Cable

Powering on the Raspberry Pi:

To use the Pi you need to:

1. Install the required Operating System on the Micro SD Card (explained in the following section)
2. Plug the SD Card in the dedicated place in the Raspberry Pi
3. Plug the USB-C power cable
4. Connect the Raspberry Pi to the monitor using the HDMI cable
5. Connect the keyboard to the Pi through the USB port

Raspberry Pi Operating System:

The Operating System used by P4Pi is based on the Debian operating system. To download and install an OS for the Raspberry Pi follow the guide that appears in the P4Pi official GitHub repository in following link:

https://github.com/p4lang/p4pi/wiki/Installing-P4Pi

Login and configuring user credentials:

When installing the OS on the Raspberry Pi, a default user is set up with the following user credentials (case sensitive):

- Login: *pi*
- Password: *raspberry*

You can change the password using the ***passwd*** command.

Connecting devices to the P4Pi WiFi:

P4Pi will come up as a hotspot as soon as you turn it on, when using an unmodified image of P4Pi.

The wireless network name (SSID) is ***p4pi***.

The initial password is ***raspberry*** - You should change the password to avoid security vulnerabilities

It is possible to configure P4Pi using either:

1. SSH: ***ssh pi@192.168.4.1***
   when prompted, use the user password that you set up in the *Login and configuring user credentials section.*

2. Web Interface: ***http://192.168.4.1***

The access point configuration can be updated via **ssh** by editing ***/etc/hostapd/hostapd.conf*** and restarting the ***hostapd*** service using:

systemctl restart hostapd.service

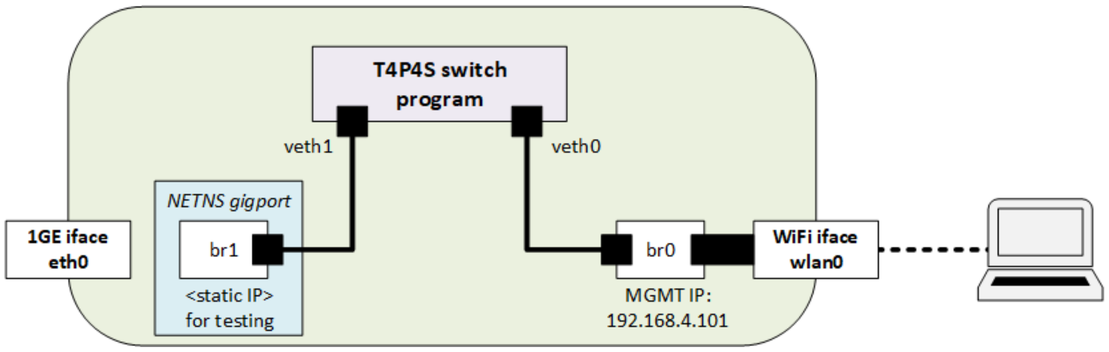
Connecting the Raspberry Pi to the Internet:

To connect the P4Pi to the internet, you need to run the following command:

```
sudo /root/setup_eth_wlan_bridge.sh
```

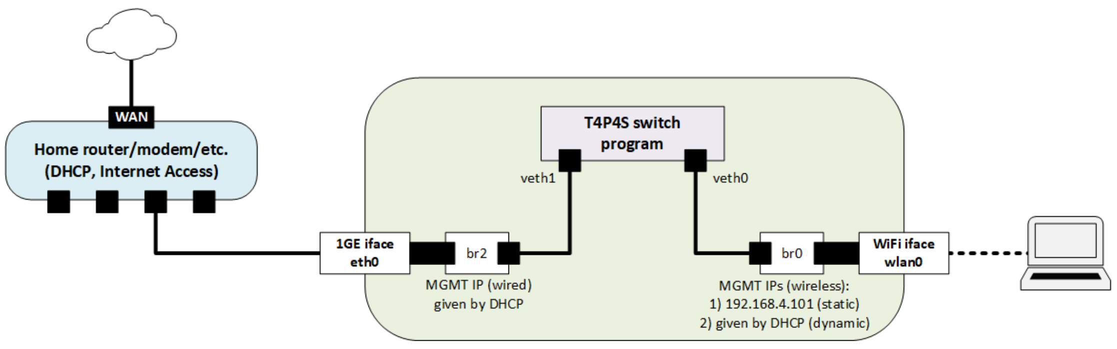
Explanation of the script setup_eth_wlan_bridge.sh:

The following figure depicts the setup used after starting the P4Pi:



P4Pi runs a DHCP daemon (dnsmasq) to assign IP addresses to devices connected through Wi-Fi. The generated T4P4S switch interconnects the wireless interface wlan0 and the Linux bridge br1. However, the 1GE Ethernet port eth0 is not used. The default management IP address is assigned to br0. Through this IP address the P4Pi node is accessible (e.g., via the web interface or SSH).

The following figure depicts the setup after running the command above, where P4Pi node acts as a low-level relay or proxy between your laptop and your home router (or a private network) connected to the 1GE wired Ethernet port.



So, the script setup_eth_wlan_bridge.sh connects the P4Pi to the internet by Bridging eth0 and wlan0 through the T4P4S switch:

- # Check if the P4PI is connected to the internet through the ethernet port, and there exists an external DHCP server. If not, stop the script.
- # Stop DHCP service (dnsmasq)
- # Redirecting veth1 endpoint:
  - o set the namespace id of the veth1-1 interface in the gigport network namespace to 1
  - o set br2 to active status (online)
  - o create a new ethernet bridge instance: "br2"
  - o set the ethernet (MAC) address aging time to 0 seconds (no aging I guess), the learned MAC address will be dropped after this time if not accessed.
  - o set br2 to active status (online)

#	Connecting eth0 to br2:
- make the interface eth0 a port of the bridge br2. This means that all frames received on eth0 will be processed as if destined for the bridge br2.
  When sending frames on br2, eth0 will be considered as a potential output interface.
- make the interface veth1-1 a port of the bridge br2.

#	Request IP for br2 from the external DHCP server
#	Request IP for br0 from the external DHCP server

P4 programs

As mentioned before, the P4Pi uses P4 for controlling packet forwarding planes in the Raspberry Pi. There are several examples of these P4 programs found in the directory /root/t4p4s/examples.

The default P4 example is “l2switch”, which is loaded automatically when powering on the Pi, which implements two core functionalities of a simple Layer 2 switch:

1. MAC learning (control plane support is needed)
2. forwarding of Ethernet frames.

To run any of the other examples, let’s say “stateful-firewall”, you should execute the following commands:

```
echo 'stateful-firewall' > /root/t4p4s-switch
systemctl restart t4p4s.service
```

Compiling and running custom P4 programs

To compile and run a P4 program on P4Pi, you can either use T4p4s automatically or manually:

- Automatically:

    1. Copy your P4 program file (let’s say *myprog.p4*) to the directory ***/root/t4p4s/examples***
    2. Define the configuration options for your program by adding the following line to the file ***/root/t4p4s/examples.cfg***:

    ```
    myprog              arch=dpdk hugepages=1024 model=v1model smem vethmode pieal piports
    ```

    3. Launch your new program by executing the following commands:

    ```
    ➢ sudo -i
    ➢ echo 'myprog' > /root/t4p4s-switch
    ➢ systemctl restart t4p4s.service
    ```

    4. If you want to run the P4Runtime shell for your program, use this command:

    ```
    sudo t4p4s-p4rtshell myprog
    ```

- Manually (with verbose output):
  For testing purposes, it is often helpful if you can see what happens during the execution of the P4 pipeline. To this end, you should launch T4P4S manually:

  1. Stop the T4P4S service:
     This is required since the service holds a lock that the T4P4S itself needs when running manually.

     ```
     sudo systemctl stop t4p4s.service
     ```

     Note that when you turn off the T4P4S service, the P4Pi will stop working as a hot-spot or an access point, and the devices connected through the WiFi will be disconnected.

  2. Compile and run your program by running the following commands:

     ```
     ➢ sudo -i
     ➢ cd /root/t4p4s
     ➢ ./t4p4s.sh :myprog p4rt verbose dbg
     ```

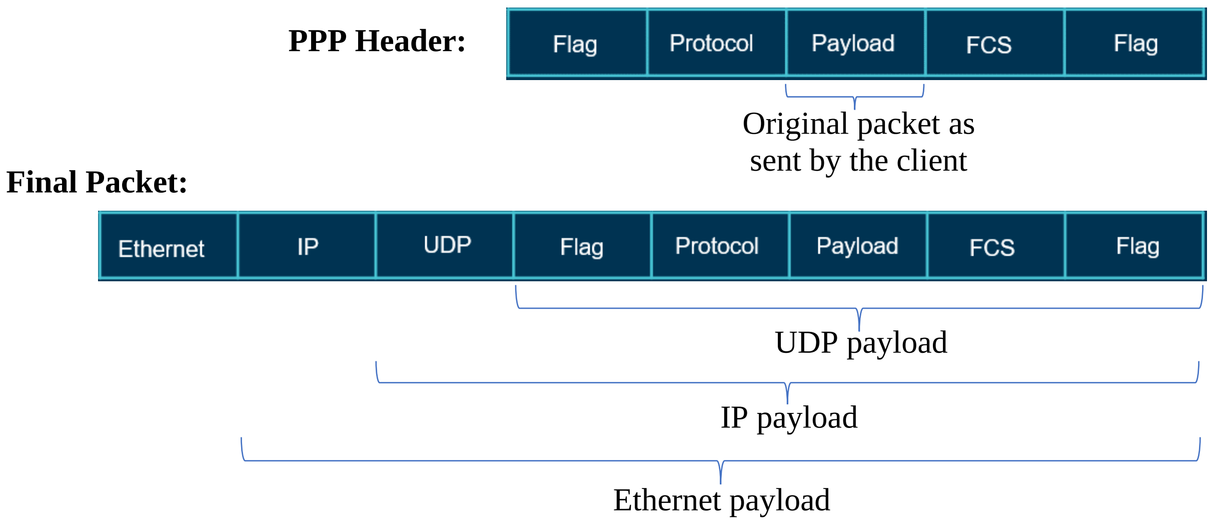
     The verbose output shows all the performed actions for each packet, including parsing steps, table lookups, action executions, deparsing, etc.

Implementation

When a packet is received in a P4Pi switch, it goes through several phases until it is retransmitted or dropped. These phases are:

1. Parsing
   Parsing the packet and extracting the headers in the right format.
2. Verifying Checksum
   Verifying that the packet headers are valid.
3. Ingress
   Specifying the rules of incoming packets and the way of dealing with them.
4. Egress
   Specifying the rules of outgoing packets (the source of the packet is the P4Pi switch itself) and the way of dealing with them.
5. Compute Checksum
   Computing the checksum of the packet after modifying its headers.
6. De-parsing
   Reconstructing the packet with the desired headers, and preparing it to be retransmitted, by encapsulating the packet with new headers.

Let's define headers and constants that we will use in the implementation.

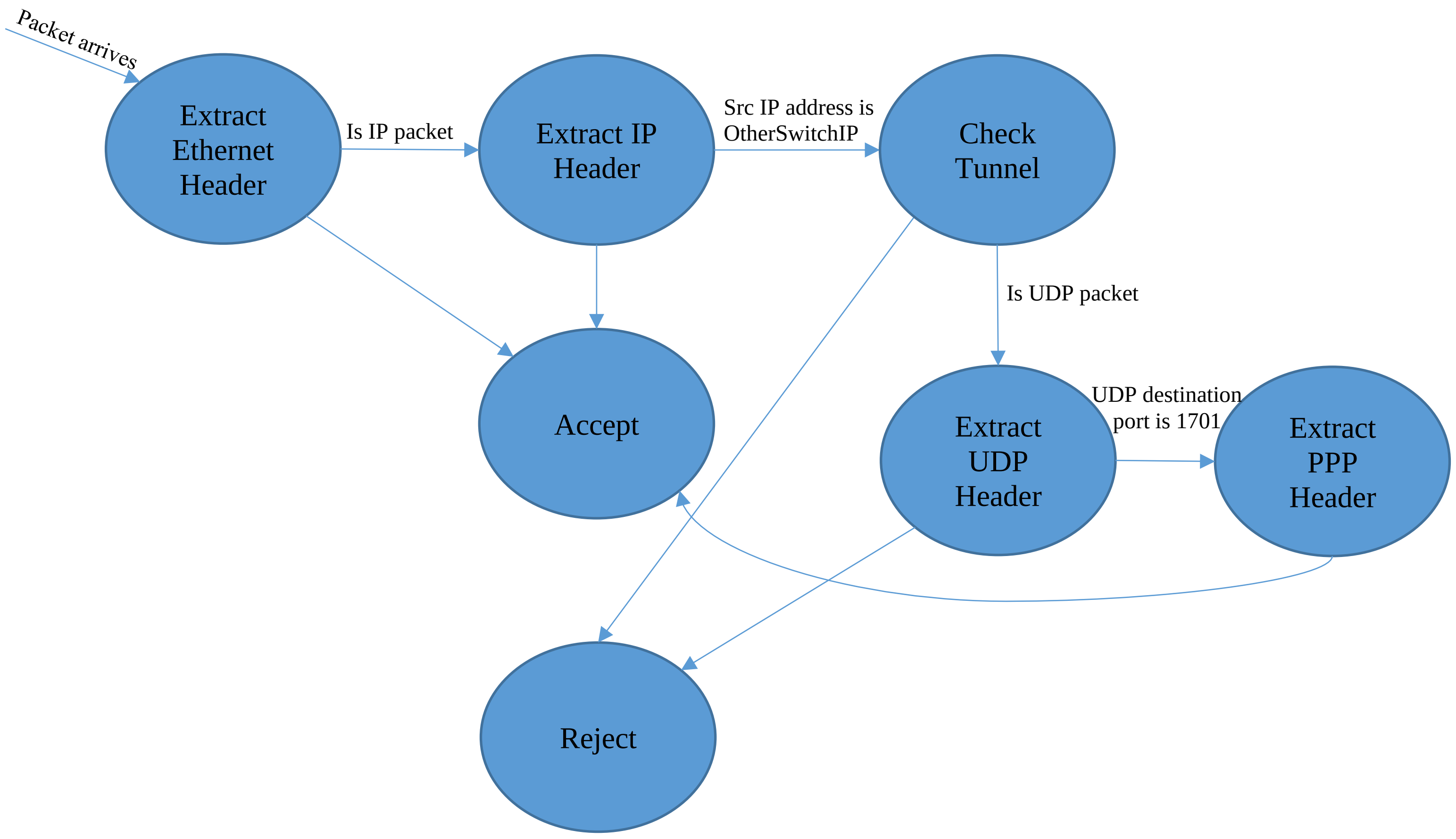


In the implementation, we need to use the IP addresses of the 2 P4Pi switches, and they are hardcoded in the switch program.
We refer to them as: *CurrentSwitchIP* and *OtherSwitchIP*.
We also use a subnet mask to recognize the source and destination of a packet. We refer to it as: *SubnetMask*.

In this subsection, we will describe the process of manipulating a packet the moment it arrives to the P4Pi switch. Note that we will skip describing the phases of Verifying Checksum, Egress and Compute Checksum, since they are intuitive or do not apply to our use.
Moreover, we will not talk about the packet encryption and decryption in this subsection since it is covered in the next subsection.

Parsing Phase

A packet arrives and enters the Parsing Phase:
We describe the parsing phase as the following automaton:



Ingress Phase

If the packet was not rejected in the Parsing Phase, it enters the Ingress Phase.
In the beginning of the phase, we have access to all the information we need, about the packet and its headers, in order to decide what to do with the packet: create a tunnel, decode a tunnel, process PPP data and control packets, or just forward the packet.
In the end of the phase, the right headers should be prepared with valid tunnel values and ready to be transmitted over the right egress port.

Given a parsed packet, we need to decide what to do with it based on its source IP address and its destination IP address.

- If the masked source IP address (XORed with SubnetMask) equals to the masked CurrentSwitchIP and the destination IP address is OtherSwitchIP, then we must change the egress_port to the Ethernet port of the P4Pi, and then, build a tunnel by encapsulating the packet with PPP header, and after that, encapsulating the PPP packet with UDP, IP and Ethernet headers, as explained in the PPP part in the background in this report.

- If the source IP address is OtherSwitchIP and the destination IP address is CurrentSwitchIP. If the packet is a PPP control packet, then the P4Pi switch must deal with it according to the PPP protocol, since it doesn't involve the clients in the local network. Otherwise, the packet is a PPP data packet, then we only have to change the egress_port to the WiFi port of the P4Pi, since we already extracted the tunnel in the Parsing Phase, and we will deal with reconstructing the packet, if necessary, in the De-Parsing Phase.

- Otherwise, one of the endpoints of the packet does not involve a P4Pi switch and their subnets, then all we need to do is to forward the packet as we got it, therefore, we need to change the egress_port of the P4Pi to the opposite of the ingress_port (the port which the packet came from).

De-Parsing Phase

Once the packet and all the required headers are ready, all is left to do is to retransmit the packet with the right headers.
We decide which headers to transmit based on the (new) source IP address and its destination IP address.

- ❖ If the packet is a IPv4 packet:
    - If the destination IP address is OtherSwitchIP, this means that we need to build a tunnel. Therefore, we must emit the headers in the following order: Ethernet ➔ IPV4 ➔ UDP ➔ PPP

    - If the masked source IP address (XORed with SubnetMask) equals to the masked OtherSwitchIP, this means that there was a tunnel, and it has been decoded. Therefore, we don't need to emit any headers, since the headers are already attached to the payload as it was transmitted from the client of the remote network.

    - Otherwise, the packet is a normal IPv4 packet that does not involve the other switch. Therefore, only emit the following headers:
      Ethernet ➔ IPV4

- ❖ Otherwise, only emit the Ethernet header (return the packet into its original form).

Encryption

In order for a PPP packet to be secure throughout, it must be encrypted before it’s transmitted from the P4Pi switch, otherwise the encapsulation and the tunneling is meaningless since all the packets and their headers will be exposed to everyone.
Therefore, the PPP packets between the two P4PIs must be encrypted with a powerful encryption algorithm.
We’ve chosen the AES encryption algorithm, which uses a symmetric key of length 128 bit, to encrypt our packets since it’s a powerful and reliable algorithm and it’s hard to break.
However, due to the lack of power of P4Pi, it is recommended to use a light-weighted AES algorithm, which operates almost the same as AES but merges two operations into one operation, which allows for a less complex algorithm and optimized running time, but with a down-side of a little less security. This version of AES can be found in this link.

As explained in the “Challenges” section bellow, there is no intuitive and straightforward way to manipulate the packet payload. Therefore, we can’t read the payload, encrypt it, and replace the payload with the encrypted one in a straightforward way.

We suggest two ways to overcome this challenge:

1. Encrypt only the PPP header packet and the network headers (Ethernet and IP headers) of the original packet. This way, it’s possible to read and update the headers in a normal way, however, the downside is that the rest of the packet will be exposed to the internet if it wasn’t encrypted by the client that sent it.

2. Define the payload as part of the packet headers. This way we can read it and encrypted in a normal way. However, since there is no loops or recursions in P4, the payload size must be constant and every operation on every chunk must be repeated in a hard-coded way for every chunk. This results in a long and bug-prone code and increase the overhead for packets which are originally small.

Challenges

P4 limitations:

- The P4 programming language is not Turing Complete, it doesn’t even have loops and. Therefore, it is hard to implement complex functions and protocols, and some of them are impossible to implement. We suppose that it has the same power as an automaton since it works in the same way.

- The P4 language does not support manipulating data payload in straight forward way, so we need to overcome this limitation by manipulating data in indirect ways. For instance, defining the payload to be as a part of the header allows us to perform arithmetical operations on the payload.
  Manipulating the payload is important for encrypting and decrypting the packets using algorithms like AES.
  This work-around still have its limitations when it comes to AES, since we need to perform arithmetical operations (like XOR) on chunks of the payload, therefore, since we don’t know the actual size of the packet and there are no loops in the P4 language, we need to limit the payload to a constant number of small chunks, or divide the whole payload space into small chunks and work on each of them manually in a hard-coded way.

Project External interface description

Since the PPP connection is done between the 2 switches, the user does not need to interfere with it. Thus, there is no special interface for the user, and all he has to do is connect to the switch through Wi-Fi and send packets as usual (e.g., using browser, command line, network applications, etc.).

Dependencies to external modules

The project depends on several hardware and software modules:

- P4PI language (P4 over Raspberry PI)
- Raspberry PI devices
- Existing routers in the Internet
- Internet protocols: UDP and IP

The relevance is explained explicitly in the relevant sections of this report.

References

- https://datatracker.ietf.org/doc/html/rfc1661
- https://github.com/p4lang/p4pi
- https://en.wikipedia.org/wiki/Tunneling_protocol
- https://en.wikipedia.org/wiki/Advanced_Encryption_Standard
- https://www.cisco.com/c/en/us/support/docs/dial-access/virtual-private-dialup-network-vpdn/23980-l2tp-23980.html
- https://dl.acm.org/doi/10.1145/3405669.3405819