Internet of Things - Laboratory exercise 01

Department of Information Technology, Uppsala University

Overview

This laboratory exercise consists of two mandatory parts.

Administration

Student 1

Name: Anderson Leong Ke Sheng
UpUnet-S ID: anle5400

Student 2

Name: Qsai Alyounes
UpUnet-S ID: qsal1429

Student 3

Name:
UpUnet-S ID:

Agreement

I/we have independently worked on the following assignment solution. In the case of two or more people, we have both taken part in creating the solution, according to the assignment specification.

Sign 1: Anderson

Sign 2:

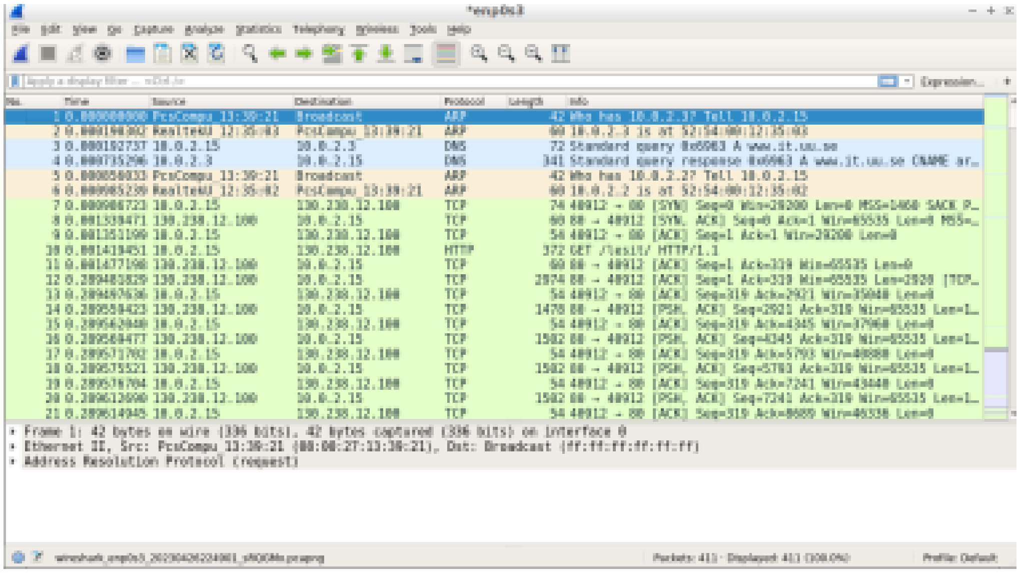
Sign 3:

Part 1

Part 1.1: See the Packets Sent From and To Your Computer

Getting Started

In this part of the lab, you will become acquainted to the tool *Wireshark*, a popular packet analyzer, that lets you view the packets sent from and to your computer. In particular, you will see the packets that are sent and received to retrieve a site from a web server. The recorded packet trace will be used to review concepts you have learned in the course. You will also learn about other protocols that are used when accessing a web page with a browser.


*enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Expression…

```
No.  Time        Source             Destination        Protocol Length Info
   1 0.000000000 PcsCompu_13:39:21  Broadcast          ARP        42 Who has 10.0.2.3? Tell 10.0.2.15
   2 0.000198302 RealtekU_12:35:03  PcsCompu_13:39:21  ARP        60 10.0.2.3 is at 52:54:00:12:35:03
   3 0.000192737 10.0.2.15          10.0.2.3           DNS        72 Standard query 0x6963 A www.it.uu.se
   4 0.000735296 10.0.2.3           10.0.2.15          DNS       341 Standard query response 0x6963 A www.it.uu.se CNAME ar…
   5 0.000850033 PcsCompu_13:39:21  Broadcast          ARP        42 Who has 10.0.2.2? Tell 10.0.2.15
   6 0.000985239 RealtekU_12:35:02  PcsCompu_13:39:21  ARP        60 10.0.2.2 is at 52:54:00:12:35:02
   7 0.000906723 10.0.2.15          130.238.12.100     TCP        74 40912 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_P…
   8 0.001339471 130.238.12.100     10.0.2.15          TCP        60 80 → 40912 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=…
   9 0.001351199 10.0.2.15          130.238.12.100     TCP        54 40912 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
  10 0.001419451 10.0.2.15          130.238.12.100     HTTP      372 GET /lesit/ HTTP/1.1
  11 0.001477198 130.238.12.100     10.0.2.15          TCP        60 80 → 40912 [ACK] Seq=1 Ack=319 Win=65535 Len=0
  12 0.289481829 130.238.12.100     10.0.2.15          TCP      2974 80 → 40912 [ACK] Seq=1 Ack=319 Win=65535 Len=2920 [TCP…
  13 0.289497636 10.0.2.15          130.238.12.100     TCP        54 40912 → 80 [ACK] Seq=319 Ack=2921 Win=35040 Len=0
  14 0.289559423 130.238.12.100     10.0.2.15          TCP      1478 80 → 40912 [PSH, ACK] Seq=2921 Ack=319 Win=65535 Len=1…
  15 0.289562040 10.0.2.15          130.238.12.100     TCP        54 40912 → 80 [ACK] Seq=319 Ack=4345 Win=37960 Len=0
  16 0.289569477 130.238.12.100     10.0.2.15          TCP      1502 80 → 40912 [PSH, ACK] Seq=4345 Ack=319 Win=65535 Len=1…
  17 0.289571702 10.0.2.15          130.238.12.100     TCP        54 40912 → 80 [ACK] Seq=319 Ack=5793 Win=40880 Len=0
  18 0.289575521 130.238.12.100     10.0.2.15          TCP      1502 80 → 40912 [PSH, ACK] Seq=5793 Ack=319 Win=65535 Len=1…
  19 0.289578704 10.0.2.15          130.238.12.100     TCP        54 40912 → 80 [ACK] Seq=319 Ack=7241 Win=43440 Len=0
  20 0.289613690 130.238.12.100     10.0.2.15          TCP      1502 80 → 40912 [PSH, ACK] Seq=7241 Ack=319 Win=65535 Len=1…
  21 0.289614945 10.0.2.15          130.238.12.100     TCP        54 40912 → 80 [ACK] Seq=319 Ack=8689 Win=46336 Len=0

▸ Frame 1: 42 bytes on wire (336 bits), 42 bytes captured (336 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_13:39:21 (08:00:27:13:39:21), Dst: Broadcast (ff:ff:ff:ff:ff:ff)
▸ Address Resolution Protocol (request)

wireshark_enp0s3_20230426224001_sNQ0Mn.pcapng    Packets: 411 · Displayed: 411 (100.0%)    Profile: Default
```

Fig. 1. A screenshot of Wireshark

Carefully read through the instructions before carrying them out. Otherwise, your packet traces may not have sufficient data to answer the questions. Some questions may require you to do some research. The course book is always a good place to start.

Start Wireshark by double clicking the icon on the desktop. You will be asked for a password, which is "`user`" (without the quotes). Select `enp0s3` from the list of network devices on the computer, and then click *Start* to start recording packets. With Wireshark still running, open the Firefox browser, and visit the website `http://www.it.uu.se/lesit/` by entering this URL in the address bar. Wait until the page has finished loading and then go back to the Wireshark window and stop capturing packets by clicking *Capture* → *Stop*.

You should see a number of lines in Wireshark now. Each line represents one packet that was either received or sent. Take some time to get familiar with Wireshark by looking at the packets. Click on a packet and have a look at the information in the lower half of the window. Click on the small arrows to expand the information.

Address and Domain Name Resolution

Now, consider the first two packets that you captured, which are used by the Address Resolution Protocol (ARP).

What is the purpose of this protocol? Describes why and how it is used here.

This protocol aims to help the senders to identify which device to send information to for an IP address. The sender first sends a request to establish a connection and requests to find out which MAC address it should be connecting with. It then receives a packet reply with providing the sender with the target MAC address given by the source MAC address in the ARP reply. This will be kept in a cache for faster routing purposes.

The next packets in the trace are related to DNS request(s) sent by the browser in order to determine the IP address associated with the `www.it.uu.se` domain name. The browser, or more generally, the application performing name resolution, may send the same request to one or more DNS servers. The IP addresses of these servers, and that of the gateway router, are often discovered by your host automatically when you connect to the network (see DHCP).
What transport protocol does the DNS protocol use?

User Datagram Protocol is used in the DNS protocol

What is/are the IP address(-es) of the DNS server(s)?

130.238.4.133;
130.238.164.6;
130.238.20.50;

What is the IP address of `www.it.uu.se`?

130.238.4.133

What is the IP address of the first node/hop your DNS request(s) has/have traversed enroute to the DNS server(s)? (hint: this can be found in the ARP packets captured earlier)

10.0.2.2

The Transport Layer

After the first DNS response, and possibly ARP packets for resolving the MAC address of the gateway router, you should see three TCP packets.
What is the purpose of these three packets?

These packages come from the three way handshake used for establishing a connection for the first time

Which port did your browser use?

54602

Apply a filter in Wireshark to only view packets of this connection.
For this enter "ip.addr == ... and tcp.port == ..." with the IP address of it.uu.se and your own port out of the first three TCP packets after the DNS request and response into the "Filter" field and confirm the input by pressing enter.
Consider the last three packets in the trace. What is their purpose? Explain why all three packets are necessary.

To terminate the connection gracefully. Both parties send [FIN, ACK] to tell each other they have no packages to send over to the other side. Both know that the connection can be closed. The last packet acknowledges receiving their [FIN,ACK] packages as a failsafe in case the package is lost and the other party waits not knowing it has finished. This frees the connection for others to use.

Which new packets are acknowledged by the last TCP packet in the trace? Justify your answer.
('New packets' means packets that were not acknowledged by previous packets.)

The [FIN, ACK] packet from host 130.238.12.100 to local at 10.0.2.15 as seen by the seq numbers and ack numbers shows that it was a reply acknowledging the FIN,ACK packet from the local host.

Retrieving a Web Page

We have so far looked at packets for address resolution and some TCP packets. Let's now inspect the packets necessary to retrieve the web page from the server.

What is the application-layer protocol used for web requests?

HTTP protocol is used for web requests

On what port does the web server listen for requests?

80

The protocol uses multiple TCP connections to deliver web resources (e.g., HTML pages, images). Identify one such connection by the tuple (server port, server address, client port, client address) plus the web resource delivered over it.

(src port: 36362, Src Addr: 10.0.2.15, Dest Port:80, Dest Addr:130.238.12.100) information delivered is a GET request from the source for an image.

Why are resources delivered over multiple TCP connections instead of a single one? Wouldn't that create unnecessary overhead?

It allows for more parallelism in sending the data over the connections as compared to congesting a single connection and leaving the other connections idle. It utilizes the bandwidth better and the overhead costs may be less than congesting the network.

Design a filter applying which reveals **only** packets showing the browser's requests for **PNG image** resources. Briefly explain how the filter works.

Http.request.method == "GET" and http.request.uri contains ".png"

This filter select only http requests that are trying to GET information from the server, and also specifically states for png format information in the request.

Adding it all up

Explain in a concise manner how the four protocols discussed are used when you use your browser to access a web page over a wired network.

The four protocols here are the ARP > DNS > TCP > HTTP.

ARP requests were made to establish a mapping of the IP address of the web server to its MAC address. This is done by the browser to the host. DNS is then used to resolve the IP address of the web page by making hops until it eventually arrives at the correct IP address of the Web Server. The TCP establishes a reliable connection between the browser and the web server, by undergoing a 3 way handshake started using a SYN packet to server, whereby a SYN-ACK is received from server, followed by browser acknowledging the SYN ACK after that. The HTTP communicates data between browser and server, where browser makes request to server in the http query and receives response from the server together with the requested data. The browser then unpacks the packet and render it accordingly for users.

Part 1.2: Looking Up Information About IP Addresses

The Internet is hierarchical, composed of a number of *autonomous systems* (AS). The `whois` command lets you look up information about an IP address, including which AS it belongs to.

Now consider this scenario: You are the network administrator for a small company. While inspecting the logs of one of your servers, you see that a host with the IP address 91.55.12.46 is trying to login into one of your servers by repeatedly trying out different passwords. The attacker is generating so many login requests that your server is overwhelmed.

Whom do you contact about the misbehaving host?
The police and your boss

What's the probable physical location of the host?
Germany

Part 2

Description

In this part of the lab, you will be using an IoT network simulator/emulator to perform data collection from nodes of a network. Moreover, you will use Wireshark to investigate the underlying protocols used in the application. This part of the lab will teach you about: ● Simulating/emulating IoT multi-hop networks

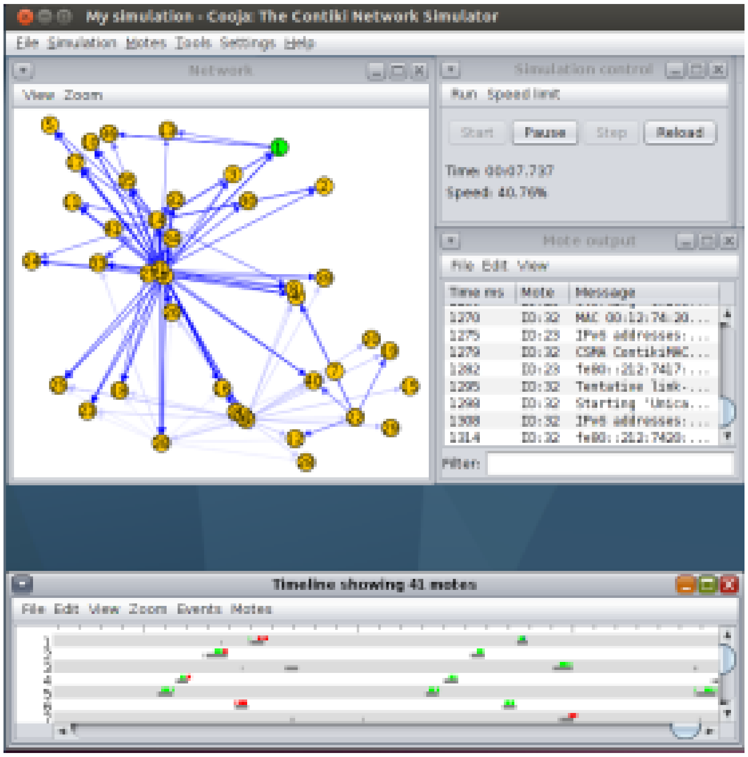
- Standard protocols used for networking IoT
- How IPv6 addressing is used in IoT

Background

In this section, you are given necessary background information related to the tools and protocols used in the task.

Cooja

Cooja is a wireless network simulator which is mainly used for developing protocols for the Internet of Things. In addition to simulation of network nodes, Cooja also supports emulation where actual hardware is emulated and deployment-level firmware images can be used. In this task, you will be using the emulation of the popular *Tmote Sky* node. A screenshot of Cooja's interface is shown below.


My simulation - Cooja: The Contiki Network Simulator
File Simulation Motes Tools Settings Help
Network
View Zoom
Simulation control
Run Speed limit
Start
Pause
Step
Reload
Mote output
File Edit View
Filter:
Timeline showing 41 motes
File Edit View Zoom Events Motes

```
Time: 00:07.737
Speed: 40.76%
```

```
Time ms  Mote   Message
1270     ID:32  MAC 00:12:74:20...
1275     ID:23  IPv6 addresses:...
1279     ID:32  CSMA ContikiMAC...
1282     ID:23  fe80::212:7417:...
1285     ID:32  Tentative link-...
1288     ID:32  Starting 'Unica...
1308     ID:32  IPv6 addresses:...
1314     ID:32  fe80::212:7420:...
```

Though *Cooja* emulates almost the entire hardware including the radio, it does not simulate all the properties of a real wireless environment. In other words, applications that rely on properties of a real wireless environment might behave differently on Cooja and vice versa.

IPv6

As the number of devices connected to the Internet has started to increase rapidly, availability of usable IP addresses (IPv4) have started to deplete rapidly. As a solution, the Internet Engineering Task Force (IETF) formalized a successor protocol called Internet Protocol version 6 (IPv6). IPv6 addresses use an 128-bit addressing scheme which allows 2^{128}, or approximately 3.4×10^{38} addresses, or more than 7.9×10^{28} times as many as what IPv4 addressing provides.

IPv6 addressing

IPv6 addresses are written in eight groups of four hexadecimal digits separated by colons (:) such as the following.

```
2001:0db8:85a3:0000:0000:8a2e:0370:7334
```

For convenience, an IPv6 address may be written in shorthand notation by following rules listed below.

- Leading zeros of any group are omitted.
- Consecutive sections of zeroes are replaced with a double colon (::). (This is known as zero compression). In order to avoid ambiguities, the double colon replacement may only be applied once in an address.

Using these rules, we can write the IPv6 address above as:

```
2001:db8:85a3::8a2e:370:7334
```

IPv6 prefixes

IPv6 uses prefixes for subnetting and routing. The high-order bits of an IPv6 address (on the left as the address is written on paper) specify the network, and the remaining bits specify the particular addresses in that network.

For example, the address `2001:db8:85a3::8a2e:370:7334` belongs to the prefix `2001:db8::/32`. Here, the high-order 32 bits represents the network. When it comes to complete IPv6 addresses, the prefix specifier (`/32` in this case) is often omitted to avoid confusion.

Special IPv6 prefixes

Some IPv6 prefixes have special use cases. The prefixes that are important to carry out this part of the lab are listed below. The addresses having these prefixes are not globally routable over the Internet.

- `fe80::/10` - This address prefix indicates that the address is valid only for communicating between two nodes. Addresses with this prefix cannot be used to communicate over multiple hops.
- `ff00::/8` - Any address with this prefix is automatically understood to be a multicast address. A multicast address is used to represent a group of destination nodes simultaneously.

RPL

RPL is the standard routing protocol for IPv6-based low-power and lossy IoT networks. RPL constructs tree-like structures where nodes of the network maintain parent/child relationships. A network running RPL can have multiple tree-like structures, and roots of the structures typically act as data sinks. These tree-like structures are called Destination Oriented Directed Acyclic Graphs (DODAGs). However, in your task, you will be using a network that has only one *DODAG*, i.e., one root node.

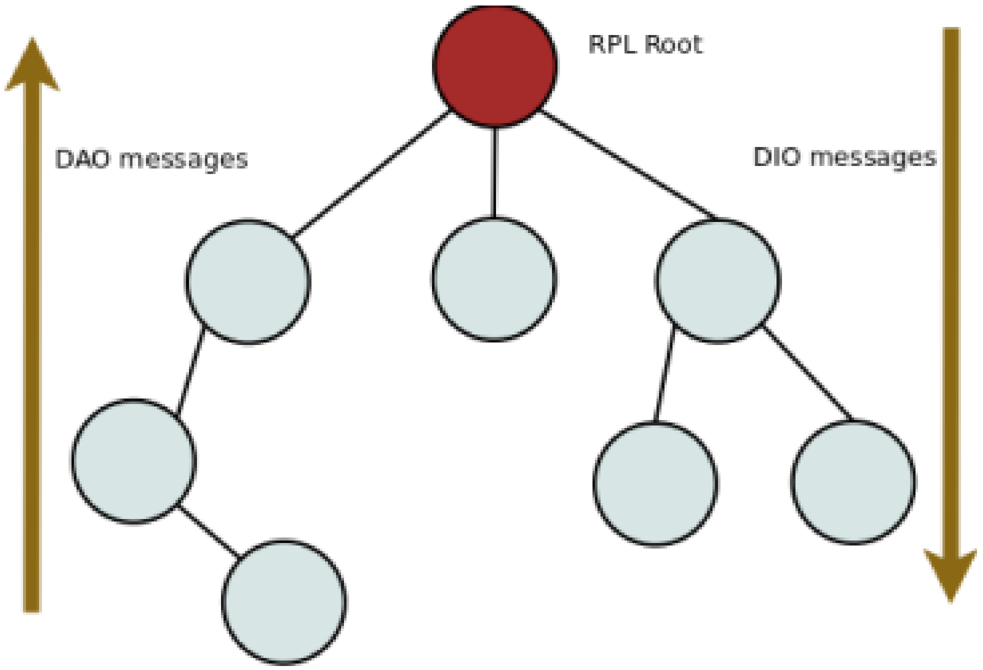
RPL uses ICMPv6 (Internet Control Message Protocol version 6) to exchange *DODAG* information among nodes. There are three main control messages used:

- *DODAG* Information Object (DIO)

  The root of the graph initiates the construction of the *DODAG* by sending this message. This message carries information that allow nodes to discover a RPL instance and learn its configuration parameters.
- *DODAG* Information Solicitation (DIS)

  Nodes may use this message to actively probe existing *DODAG* Information Object from a RPL node. Otherwise, nodes have to wait until they receive DIO messages.
- Destination Advertisement Object (DAO)

  This message is used by children nodes to propagate destination information towards root of the *DODAG*. Typically, DAO messages are sent by children to their parents. The following figure shows how DIO and DAO messages propagate in the tree-like structure.



As with any other routing protocol, the purpose of RPL is to populate the forwarding table of the nodes. The TCP/IP stack of the nodes uses the forwarding table to decide which node should be selected as the next hop when forwarding a packet. Note that you are not required to learn about RPL in detail in order to carry out this task.

Data collection application

In this task, you will be running a data collection application in Cooja with 25 emulated *Tmote Sky* nodes. Node 1 acts as the data sink as well as the RPL root node. All nodes send several statistics, such as sensor values and information parent node, to the sink node (node 1) every 60~62 seconds.

Getting started

- Start a terminal window by double clicking the icon "Terminal" on the Desktop. ● Use the following command to change the directory to RPL data collection example application directory

```
# cd contiki-github/examples/ipv6/rpl-collect
```

- Start Cooja with a pre-saved simulation file for RPL based data collection.

```
# make TARGET=cooja collect-tree-dense-noloss.csc
```

- Cooja will start and take a short time to load firmware images to all nodes.
    - ○ Two windows will pop up:
        - ■ Cooja simulator (with several sub windows)
        - ■ Sensor Data Collect with Contiki (Data collection control)
    - ○ Here, you can see all 25 nodes in the "Network" subwindow of Cooja. ● In Cooja, click "Tools → Radio messages" to open radio messages window. ○ In order to log all radio messages into a `.pcap` file, click "Analyzer → 6LowPAN Analyzer with PCAP". The `.pcap` file can be opened from *Wireshark* later for analyzing packets.
    - ○ This will create a file similar to the pattern `radiolog-`

`***************.pcap` (e.g., `radiolog-1424111165911.pcap`) in the current directory .

- In order to see the radio communication visually, click on the “View” menu item of the “Network” window of Cooja and select “Radio traffic” option.
- Now, start the simulation by clicking the “Start” button in the “Simulation control” window of Cooja. Let the simulation run for about ***120 (simulated) seconds*** in order to have enough packets saved in the `.pcap` file.
- Verify that a file similar to the pattern `radiolog-***************.pcap` is in the directory `/home/user/contiki-github/examples/ipv6/rpl-collect` by opening another terminal.
- Open “Wireshark” by clicking the icon on the Desktop.
  - Then, open the file `radiolog-***************.pcap` by “File → Open” menu.
  - A file chooser dialog will appear and you need to select `radiolog-***************.pcap` file located in `/home/user/contiki github/examples/ipv6/rpl-collect` directory.

Protocols

- As you can notice, IPv6 is used as the network layer protocol. Give **two** reasons for using IPv6 as the network protocol in IoT, providing also the implications for IoT.

IPv6 allows for much more connections as it provides 1028 times more addresses than IPv4 given how it uses 128-bit addresses compared to the 32-bit addresses, more than 7.9×10^{28} times of IPv4, providing more than enough unique addresses for the large amounts of sensors IoT systems may need without ever running out in the foreseeable future.

IPv6 also allows for some special prefixes that are not globally routable, meaning there is an added security to it since it would mean the devices are not directly linked to the internet, making it less susceptible to be compromised. These prefixes also allow better communication between the IoT devices such as multicasting prefixes to broadcast data to a group of devices more efficiently.

- IEEE 802.15.4 standard specifies 4 MAC layer frame types. The values for these frame types are shown below.

Frame type	**Value**
Beacon	0x0000
Data	0x0001
Acknowledgment	0x0002
MAC command	0x0003

Use *Wireshark* to find out which MAC layer frames are recorded in the `radiolog-***************.pcap` file. (Hint: You can use the filter `wpan.frame_type == <value>` to filter the particular frame type from others.)

The layer frames found are Data and Acknowledgment packets are found.

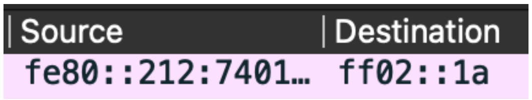
- As you can see from the Wireshark traces, ICMPv6 is used to carry RPL-related information. Why do you think ICMPv6 has been chosen over popular transport layer protocols such as TCP and UDP? Note that the operational purpose of ICMPv6 is similar to ICMPv4.

ICMPv6 allows for other features such as multicast, providing better efficiency and simplicity.

- Which transport layer protocol is used in the data collection application?

UDP (search "data" wireshark)

IPv6 addressing and RPL



Source	Destination
fe80::212:7401…	ff02::1a

- What is the destination IP address of ICMPv6 packets that carry RPL DIO messages? (Hint: ICMPv6 packets that carry DIO messages have `type` and `code` fields set to `155` and `1` respectively)

ff02::1a

- Why do you think that the destination address of ICMPv6 packets that carry RPL DIO messages is always same? (Hint: Nodes broadcast DIO messages)

Because the nodes are broadcasting DIO messages, and the multicast address has the prefix ff00:: and thus this prefix is used for multicasting (broadcast) to all nodes in the group thus always the same. Also, since this group needs to be identified, they have the same address as a whole after the prefix. The prefix allows for identification that it is a multicast and the address after that determines which group, in this case there is only 1 group thus all of them are the same.

- Write the 16-bit prefix of the destination address of UDP packets in shorthand notation of IPv6 addresses.

::/16

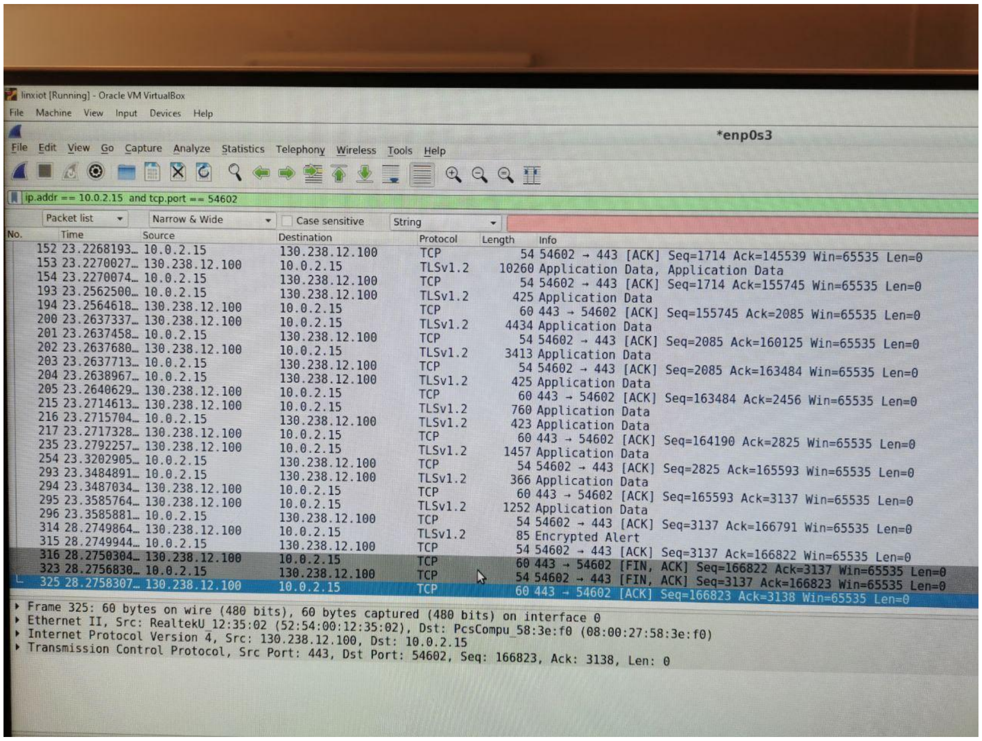
- Write the 16-bit prefix of the destination address of ICMPv6 packets that carry DAO messages in shorthand notation of IPv6 addresses. (Hint: ICMPv6 packets that carry DAO messages have `type` and `code` fields set to `155` and `2` respectively)

fe80::

- Why do you think that the prefixes you have written for the two previous questions are different? (Hint: DAO messages are sent by children to their parents while UDP packets are sent by all nodes to the sink node which is the RPL root of the tree.)

The UDP packets are all sent to a common destination, the sink node RPL root of the tree, which is identified as ::1. However, DAO messages are send by children to their immediate parents and hence have the prefix fe80 since fe80 prefix only does a single hop while the other prefix would allow more hops to arrive at the root if the source was further down the tree and fe80 prefix would not be able to hop multiple times to reach the root.

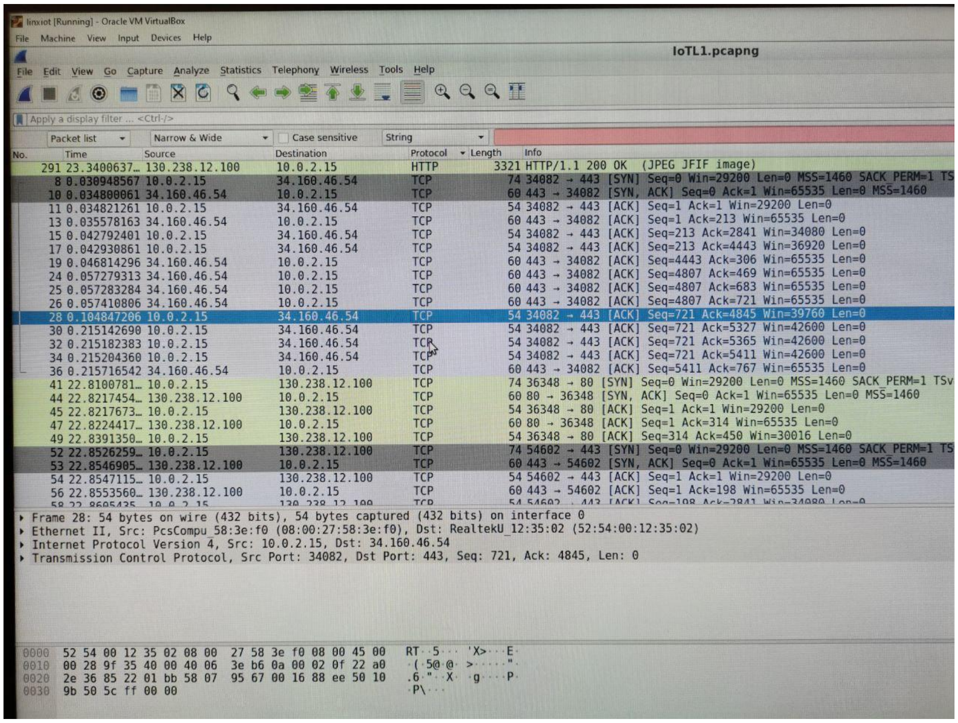
Appendix


linxiot [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
*enp0s3
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Packet list
Narrow & Wide
Case sensitive
String

```
ip.addr == 10.0.2.15  and tcp.port == 54602
```

```
No.  Time          Source          Destination     Protocol  Length Info
152 23.2268193… 10.0.2.15       130.238.12.100  TCP          54 54602 → 443 [ACK] Seq=1714 Ack=145539 Win=65535 Len=0
153 23.2270027… 130.238.12.100  10.0.2.15       TLSv1.2   10260 Application Data, Application Data
154 23.2270074… 10.0.2.15       130.238.12.100  TCP          54 54602 → 443 [ACK] Seq=1714 Ack=155745 Win=65535 Len=0
193 23.2562500… 10.0.2.15       130.238.12.100  TLSv1.2     425 Application Data
194 23.2564618… 130.238.12.100  10.0.2.15       TCP          60 443 → 54602 [ACK] Seq=155745 Ack=2085 Win=65535 Len=0
200 23.2637337… 130.238.12.100  10.0.2.15       TLSv1.2    4434 Application Data
201 23.2637458… 10.0.2.15       130.238.12.100  TCP          54 54602 → 443 [ACK] Seq=2085 Ack=160125 Win=65535 Len=0
202 23.2637680… 130.238.12.100  10.0.2.15       TLSv1.2    3413 Application Data
203 23.2637713… 10.0.2.15       130.238.12.100  TCP          54 54602 → 443 [ACK] Seq=2085 Ack=163484 Win=65535 Len=0
204 23.2638967… 10.0.2.15       130.238.12.100  TLSv1.2     425 Application Data
205 23.2640629… 130.238.12.100  10.0.2.15       TCP          60 443 → 54602 [ACK] Seq=163484 Ack=2456 Win=65535 Len=0
215 23.2714613… 130.238.12.100  10.0.2.15       TLSv1.2     760 Application Data
216 23.2715704… 10.0.2.15       130.238.12.100  TLSv1.2     423 Application Data
217 23.2717328… 130.238.12.100  10.0.2.15       TCP          60 443 → 54602 [ACK] Seq=164190 Ack=2825 Win=65535 Len=0
235 23.2792257… 130.238.12.100  10.0.2.15       TLSv1.2    1457 Application Data
254 23.3202905… 10.0.2.15       130.238.12.100  TCP          54 54602 → 443 [ACK] Seq=2825 Ack=165593 Win=65535 Len=0
293 23.3484891… 10.0.2.15       130.238.12.100  TLSv1.2     366 Application Data
294 23.3487034… 130.238.12.100  10.0.2.15       TCP          60 443 → 54602 [ACK] Seq=165593 Ack=3137 Win=65535 Len=0
295 23.3585764… 130.238.12.100  10.0.2.15       TLSv1.2    1252 Application Data
296 23.3585881… 10.0.2.15       130.238.12.100  TCP          54 54602 → 443 [ACK] Seq=3137 Ack=166791 Win=65535 Len=0
314 28.2749864… 130.238.12.100  10.0.2.15       TLSv1.2      85 Encrypted Alert
315 28.2749944… 10.0.2.15       130.238.12.100  TCP          54 54602 → 443 [ACK] Seq=3137 Ack=166822 Win=65535 Len=0
316 28.2750304… 130.238.12.100  10.0.2.15       TCP          60 443 → 54602 [FIN, ACK] Seq=166822 Ack=3137 Win=65535 Len=0
323 28.2756830… 10.0.2.15       130.238.12.100  TCP          54 54602 → 443 [FIN, ACK] Seq=3137 Ack=166823 Win=65535 Len=0
325 28.2758307… 130.238.12.100  10.0.2.15       TCP          60 443 → 54602 [ACK] Seq=166823 Ack=3138 Win=65535 Len=0
```

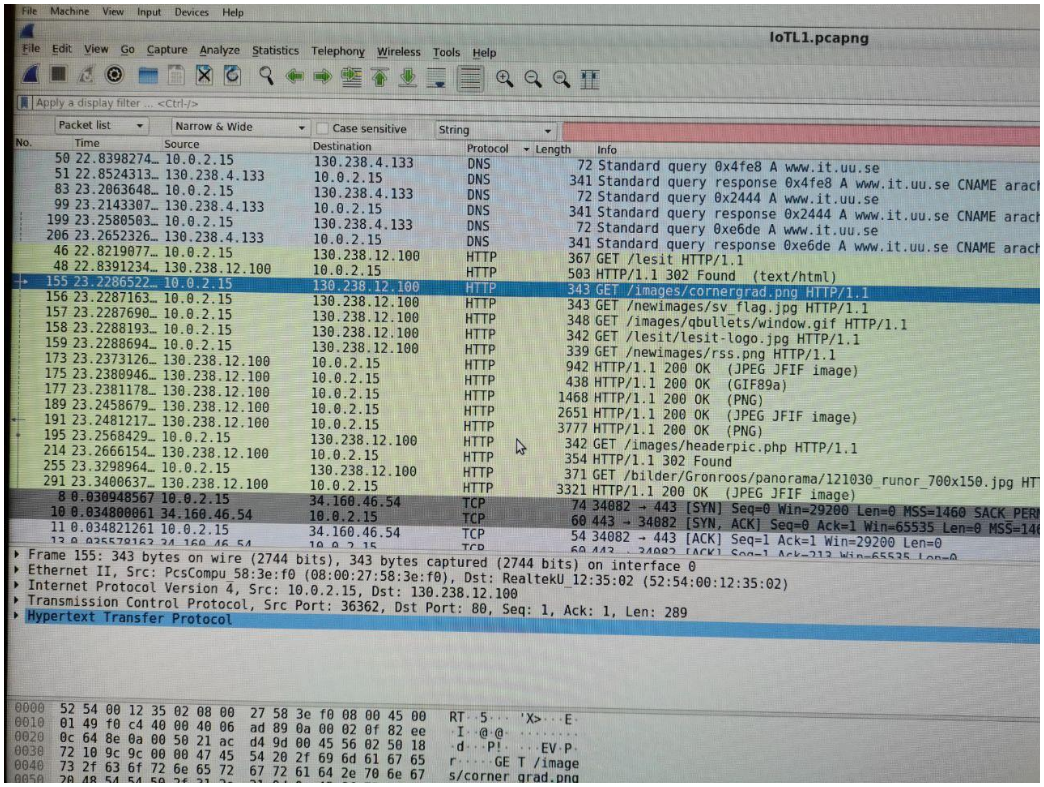
```
▸ Frame 325: 60 bytes on wire (480 bits), 60 bytes captured (480 bits) on interface 0
▸ Ethernet II, Src: RealtekU_12:35:02 (52:54:00:12:35:02), Dst: PcsCompu_58:3e:f0 (08:00:27:58:3e:f0)
▸ Internet Protocol Version 4, Src: 130.238.12.100, Dst: 10.0.2.15
▸ Transmission Control Protocol, Src Port: 443, Dst Port: 54602, Seq: 166823, Ack: 3138, Len: 0
```


linxiot [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
IoTL1.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter … <Ctrl-/>
Packet list
Narrow & Wide
Case sensitive
String

```
No.  Time          Source          Destination     Protocol  Length Info
291 23.3400637… 130.238.12.100  10.0.2.15       HTTP       3321 HTTP/1.1 200 OK  (JPEG JFIF image)
  8 0.030948567 10.0.2.15       34.160.46.54    TCP          74 34082 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TS
 10 0.034800061 34.160.46.54    10.0.2.15       TCP          60 443 → 34082 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
 11 0.034821261 10.0.2.15       34.160.46.54    TCP          54 34082 → 443 [ACK] Seq=1 Ack=1 Win=29200 Len=0
 13 0.035578163 34.160.46.54    10.0.2.15       TCP          60 443 → 34082 [ACK] Seq=1 Ack=213 Win=65535 Len=0
 15 0.042792401 10.0.2.15       34.160.46.54    TCP          54 34082 → 443 [ACK] Seq=213 Ack=2841 Win=34080 Len=0
 17 0.042930861 10.0.2.15       34.160.46.54    TCP          54 34082 → 443 [ACK] Seq=213 Ack=4443 Win=36920 Len=0
 19 0.046814296 34.160.46.54    10.0.2.15       TCP          60 443 → 34082 [ACK] Seq=4443 Ack=306 Win=65535 Len=0
 24 0.057279313 34.160.46.54    10.0.2.15       TCP          60 443 → 34082 [ACK] Seq=4807 Ack=469 Win=65535 Len=0
 25 0.057283284 34.160.46.54    10.0.2.15       TCP          60 443 → 34082 [ACK] Seq=4807 Ack=683 Win=65535 Len=0
 26 0.057410806 34.160.46.54    10.0.2.15       TCP          60 443 → 34082 [ACK] Seq=4807 Ack=721 Win=65535 Len=0
 28 0.104847206 10.0.2.15       34.160.46.54    TCP          54 34082 → 443 [ACK] Seq=721 Ack=4845 Win=39760 Len=0
 30 0.215142690 10.0.2.15       34.160.46.54    TCP          54 34082 → 443 [ACK] Seq=721 Ack=5327 Win=42600 Len=0
 32 0.215182383 10.0.2.15       34.160.46.54    TCP          54 34082 → 443 [ACK] Seq=721 Ack=5365 Win=42600 Len=0
 34 0.215204360 10.0.2.15       34.160.46.54    TCP          54 34082 → 443 [ACK] Seq=721 Ack=5411 Win=42600 Len=0
 36 0.215716542 34.160.46.54    10.0.2.15       TCP          60 443 → 34082 [ACK] Seq=5411 Ack=767 Win=65535 Len=0
 41 22.8100781… 10.0.2.15       130.238.12.100  TCP          74 36348 → 80 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TSv
 44 22.8217454… 130.238.12.100  10.0.2.15       TCP          60 80 → 36348 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
 45 22.8217673… 10.0.2.15       130.238.12.100  TCP          54 36348 → 80 [ACK] Seq=1 Ack=1 Win=29200 Len=0
 47 22.8224417… 130.238.12.100  10.0.2.15       TCP          60 80 → 36348 [ACK] Seq=1 Ack=314 Win=65535 Len=0
 49 22.8391350… 10.0.2.15       130.238.12.100  TCP          54 36348 → 80 [ACK] Seq=314 Ack=450 Win=30016 Len=0
 52 22.8526259… 10.0.2.15       130.238.12.100  TCP          74 54602 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TS
 53 22.8546905… 130.238.12.100  10.0.2.15       TCP          60 443 → 54602 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
 54 22.8547115… 10.0.2.15       130.238.12.100  TCP          54 54602 → 443 [ACK] Seq=1 Ack=1 Win=29200 Len=0
 56 22.8553560… 130.238.12.100  10.0.2.15       TCP          60 443 → 54602 [ACK] Seq=1 Ack=198 Win=65535 Len=0
 58 22.8605435… 10.0.2.15       130.238.12.100  TCP          54 54602 → 443 [ACK] Seq=198 Ack=2841 Win=34080 Len=0
```

```
▸ Frame 28: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_58:3e:f0 (08:00:27:58:3e:f0), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 34.160.46.54
▸ Transmission Control Protocol, Src Port: 34082, Dst Port: 443, Seq: 721, Ack: 4845, Len: 0
```

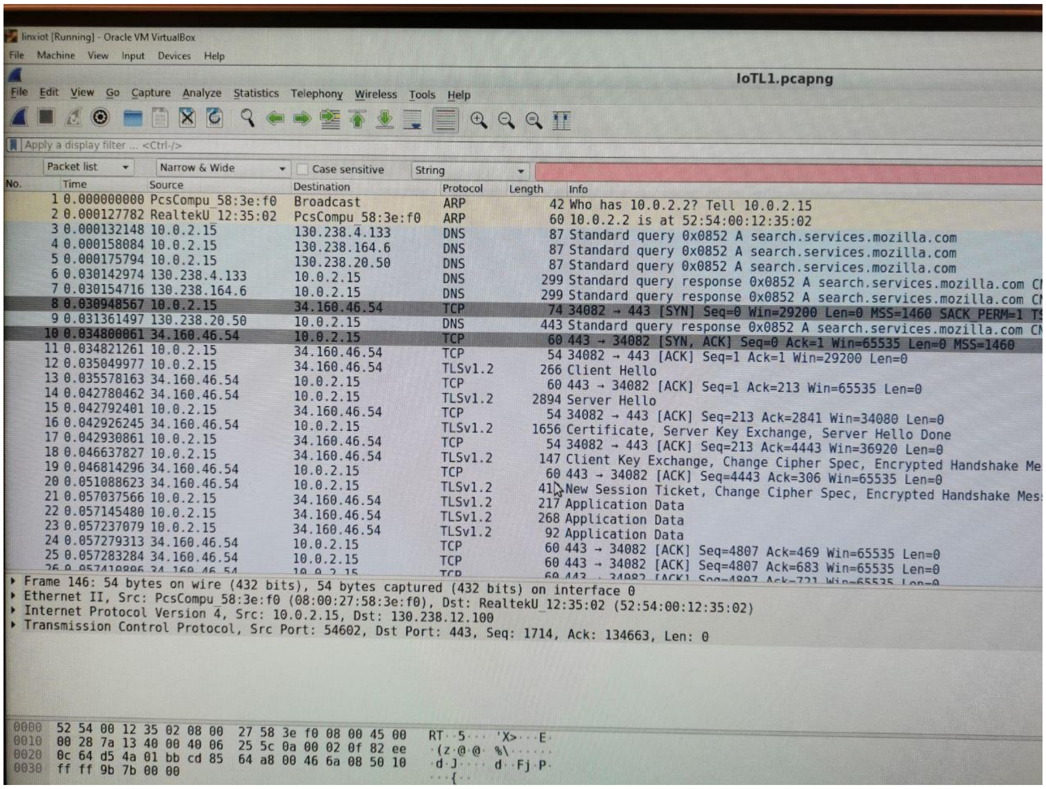
```
0000  52 54 00 12 35 02 08 00  27 58 3e f0 08 00 45 00   RT··5···'X>···E·
0010  00 28 9f 35 40 00 40 06  3e b6 0a 00 02 0f 22 a0   ·(·5@·@·>·····"·
0020  2e 36 85 22 01 bb 58 07  95 67 00 16 88 ee 50 10   .6·"··X··g····P·
0030  9b 50 5c ff 00 00                                  ·P\···
```


File Machine View Input Devices Help
IoTL1.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Packet list
Narrow & Wide
Case sensitive
String

```
No.   Time          Source          Destination     Protocol  Length Info
   50 22.8398274…  10.0.2.15       130.238.4.133   DNS          72 Standard query 0x4fe8 A www.it.uu.se
   51 22.8524313…  130.238.4.133   10.0.2.15       DNS         341 Standard query response 0x4fe8 A www.it.uu.se CNAME arach
   83 23.2063648…  10.0.2.15       130.238.4.133   DNS          72 Standard query 0x2444 A www.it.uu.se
   99 23.2143307…  130.238.4.133   10.0.2.15       DNS         341 Standard query response 0x2444 A www.it.uu.se CNAME arach
  199 23.2580503…  10.0.2.15       130.238.4.133   DNS          72 Standard query 0xe6de A www.it.uu.se
  206 23.2652326…  130.238.4.133   10.0.2.15       DNS         341 Standard query response 0xe6de A www.it.uu.se CNAME arach
   46 22.8219077…  10.0.2.15       130.238.12.100  HTTP        367 GET /lesit HTTP/1.1
   48 22.8391234…  130.238.12.100  10.0.2.15       HTTP        503 HTTP/1.1 302 Found  (text/html)
  155 23.2286522…  10.0.2.15       130.238.12.100  HTTP        343 GET /images/cornergrad.png HTTP/1.1
  156 23.2287163…  10.0.2.15       130.238.12.100  HTTP        343 GET /newimages/sv_flag.jpg HTTP/1.1
  157 23.2287690…  10.0.2.15       130.238.12.100  HTTP        348 GET /images/qbullets/window.gif HTTP/1.1
  158 23.2288193…  10.0.2.15       130.238.12.100  HTTP        342 GET /lesit/lesit-logo.jpg HTTP/1.1
  159 23.2288694…  10.0.2.15       130.238.12.100  HTTP        339 GET /newimages/rss.png HTTP/1.1
  173 23.2373126…  130.238.12.100  10.0.2.15       HTTP        942 HTTP/1.1 200 OK  (JPEG JFIF image)
  175 23.2380946…  130.238.12.100  10.0.2.15       HTTP        438 HTTP/1.1 200 OK  (GIF89a)
  177 23.2381178…  130.238.12.100  10.0.2.15       HTTP       1468 HTTP/1.1 200 OK  (PNG)
  189 23.2458679…  130.238.12.100  10.0.2.15       HTTP       2651 HTTP/1.1 200 OK  (JPEG JFIF image)
  191 23.2481217…  130.238.12.100  10.0.2.15       HTTP       3777 HTTP/1.1 200 OK  (PNG)
  195 23.2568429…  10.0.2.15       130.238.12.100  HTTP        342 GET /images/headerpic.php HTTP/1.1
  214 23.2666154…  130.238.12.100  10.0.2.15       HTTP        354 HTTP/1.1 302 Found
  255 23.3298964…  10.0.2.15       130.238.12.100  HTTP        371 GET /bilder/Gronroos/panorama/121030_runor_700x150.jpg HT
  291 23.3400637…  130.238.12.100  10.0.2.15       HTTP       3321 HTTP/1.1 200 OK  (JPEG JFIF image)
    8 0.030948567  10.0.2.15       34.160.46.54    TCP          74 34082 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM
   10 0.034800061  34.160.46.54    10.0.2.15       TCP          60 443 → 34082 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=146
   11 0.034821261  10.0.2.15       34.160.46.54    TCP          54 34082 → 443 [ACK] Seq=1 Ack=1 Win=29200 Len=0
   13 0.035578163  34.160.46.54    10.0.2.15       TCP          60 443 → 34082 [ACK] Seq=1 Ack=213 Win=65535 Len=0

▸ Frame 155: 343 bytes on wire (2744 bits), 343 bytes captured (2744 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_58:3e:f0 (08:00:27:58:3e:f0), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 130.238.12.100
▸ Transmission Control Protocol, Src Port: 36362, Dst Port: 80, Seq: 1, Ack: 1, Len: 289
▸ Hypertext Transfer Protocol

0000  52 54 00 12 35 02 08 00  27 58 3e f0 08 00 45 00   RT··5··· 'X>···E·
0010  01 49 f0 c4 40 00 40 06  ad 89 0a 00 02 0f 82 ee   ·I··@·@· ········
0020  0c 64 8e 0a 00 50 21 ac  d4 9d 00 45 56 02 50 18   ·d···P!· ···EV·P·
0030  72 10 9c 9c 00 00 47 45  54 20 2f 69 6d 61 67 65   r·····GE T /image
0040  73 2f 63 6f 72 6e 65 72  67 72 61 64 2e 70 6e 67   s/corner grad.png
```


linxiot [Running] - Oracle VM VirtualBox
File Machine View Input Devices Help
IoTL1.pcapng
File Edit View Go Capture Analyze Statistics Telephony Wireless Tools Help
Apply a display filter ... <Ctrl-/>
Packet list
Narrow & Wide
Case sensitive
String

```
No.  Time         Source             Destination        Protocol  Length Info
   1 0.000000000  PcsCompu_58:3e:f0  Broadcast          ARP          42 Who has 10.0.2.2? Tell 10.0.2.15
   2 0.000127782  RealtekU_12:35:02  PcsCompu_58:3e:f0  ARP          60 10.0.2.2 is at 52:54:00:12:35:02
   3 0.000132148  10.0.2.15          130.238.4.133      DNS          87 Standard query 0x0852 A search.services.mozilla.com
   4 0.000158084  10.0.2.15          130.238.164.6      DNS          87 Standard query 0x0852 A search.services.mozilla.com
   5 0.000175794  10.0.2.15          130.238.20.50      DNS          87 Standard query 0x0852 A search.services.mozilla.com
   6 0.030142974  130.238.4.133      10.0.2.15          DNS         299 Standard query response 0x0852 A search.services.mozilla.com CN
   7 0.030154716  130.238.164.6      10.0.2.15          DNS         299 Standard query response 0x0852 A search.services.mozilla.com CN
   8 0.030948567  10.0.2.15          34.160.46.54       TCP          74 34082 → 443 [SYN] Seq=0 Win=29200 Len=0 MSS=1460 SACK_PERM=1 TS
   9 0.031361497  130.238.20.50      10.0.2.15          DNS         443 Standard query response 0x0852 A search.services.mozilla.com CN
  10 0.034800061  34.160.46.54       10.0.2.15          TCP          60 443 → 34082 [SYN, ACK] Seq=0 Ack=1 Win=65535 Len=0 MSS=1460
  11 0.034821261  10.0.2.15          34.160.46.54       TCP          54 34082 → 443 [ACK] Seq=1 Ack=1 Win=29200 Len=0
  12 0.035049977  10.0.2.15          34.160.46.54       TLSv1.2     266 Client Hello
  13 0.035578163  34.160.46.54       10.0.2.15          TCP          60 443 → 34082 [ACK] Seq=1 Ack=213 Win=65535 Len=0
  14 0.042780462  34.160.46.54       10.0.2.15          TLSv1.2    2894 Server Hello
  15 0.042792401  10.0.2.15          34.160.46.54       TCP          54 34082 → 443 [ACK] Seq=213 Ack=2841 Win=34080 Len=0
  16 0.042926245  34.160.46.54       10.0.2.15          TLSv1.2    1656 Certificate, Server Key Exchange, Server Hello Done
  17 0.042930861  10.0.2.15          34.160.46.54       TCP          54 34082 → 443 [ACK] Seq=213 Ack=4443 Win=36920 Len=0
  18 0.046637827  10.0.2.15          34.160.46.54       TLSv1.2     147 Client Key Exchange, Change Cipher Spec, Encrypted Handshake Me
  19 0.046814296  34.160.46.54       10.0.2.15          TCP          60 443 → 34082 [ACK] Seq=4443 Ack=306 Win=65535 Len=0
  20 0.051088623  34.160.46.54       10.0.2.15          TLSv1.2     41  New Session Ticket, Change Cipher Spec, Encrypted Handshake Mes
  21 0.057037566  10.0.2.15          34.160.46.54       TLSv1.2     217 Application Data
  22 0.057145480  10.0.2.15          34.160.46.54       TLSv1.2     268 Application Data
  23 0.057237079  10.0.2.15          34.160.46.54       TLSv1.2      92 Application Data
  24 0.057279313  34.160.46.54       10.0.2.15          TCP          60 443 → 34082 [ACK] Seq=4807 Ack=469 Win=65535 Len=0
  25 0.057283284  34.160.46.54       10.0.2.15          TCP          60 443 → 34082 [ACK] Seq=4807 Ack=683 Win=65535 Len=0
  26 0.057410806  34.160.46.54       10.0.2.15          TCP          60 443 → 34082 [ACK] Seq=4807 Ack=721 Win=65535 Len=0

▸ Frame 146: 54 bytes on wire (432 bits), 54 bytes captured (432 bits) on interface 0
▸ Ethernet II, Src: PcsCompu_58:3e:f0 (08:00:27:58:3e:f0), Dst: RealtekU_12:35:02 (52:54:00:12:35:02)
▸ Internet Protocol Version 4, Src: 10.0.2.15, Dst: 130.238.12.100
▸ Transmission Control Protocol, Src Port: 54602, Dst Port: 443, Seq: 1714, Ack: 134663, Len: 0

0000  52 54 00 12 35 02 08 00  27 58 3e f0 08 00 45 00   RT··5··· 'X>···E·
0010  00 28 7a 13 40 00 40 06  25 5c 0a 00 02 0f 82 ee   ·(z·@·@· %\······
0020  0c 64 d5 4a 01 bb cd 85  64 a8 00 46 6a 08 50 10   ·d·J···· d··Fj·P·
0030  ff ff 9b 7b 00 00                                  ···{··
```