L-Università ta' Malta
Faculty of Information & Communication Technology

Lab 1 Report

Manwel Bugeja

May 31, 2021

Contents

1 Introduction

The aim of this assignment was to replace the helper classes in an NS3 simulation program. Section 2 describes the code replacement of each helper class. On top of that, this said section also describes how the specified code correlates with the Open Systems Interconnection (OSI) Model. Section 3 investigates a packet captured.

2 Code changes

2.1 NodeContainer

The node container was removed by simply creating the nodes separately and not storing them in a container at all.

2.2 PointToPointHelper

Listing 1: Creating a P2P net device

```
        Ptr<PointToPointChannel> pointToPointChannel =
            CreateObjectWithAttributes<PointToPointChannel>(
            "Delay", StringValue("3ms"));


    Ptr<Node> nodeA = CreateObject<Node>();
    //nodeContainer.Add(nodeA);

    Ptr<PointToPointNetDevice> pointToPointNetDeviceA =
        CreateObjectWithAttributes<PointToPointNetDevice>("
        DataRate", StringValue("5Mbps"));
    nodeA->AddDevice(pointToPointNetDeviceA);

    Ptr<Queue<Packet> > queueA = CreateObject<DropTailQueue
        <Packet> > ();
    pointToPointNetDeviceA->SetQueue(queueA);

    pointToPointNetDeviceA->Attach(pointToPointChannel);

    netDeviceContainer.Add(pointToPointNetDeviceA);
```

Referring to listing 1, this part of the code starts by creating a point to point channel. This channel acts as a medium between the point to point devices for example a wire. This means that this corresponds to the first layer of the OSI model.

After that, the first node was created along with its corresponding point to point (P2P) net device. The same attributes as the original code were used for the net device. Then it was attached to the node. A drop tail queue was initiated to be used by the net device. Finally, the previously created channel was also attached to the net device and the device was added to its corresponding container. This process was repeated another time for a second node. The P2P protocol corresponds to the data link layer i.e. layer to of the OSI model.

2.3 InternetStackHelper

For the Network layer and Transport layer, several protocols where bundled with the nodes. These are `ArpL3Protocol, Ipv4L3Protocol, Icmpv4L4Protocol, UdpL4Protocol`. The first three being make part of the Network Layer while the `UdpL4Protocol` makes part of the Transport layer. Static routing was used for IPv4. A traffic control layer and packet socket factory where also aggregated to the nodes.

2.4 Ipv4AddressHelper

This part of the code also concerns the Internet layer of the nodes. Here, the devices are each given a unique IPv4 address. The nodes are given the IP addresses `192.169.1.1` and `192.169.1.2` with the default subnet mask for class C i.e. `255.255.255.0`. Here, like in `Ipv4AddressHelper`, addresses are only incremented (the code does not take into account a system of IPs.

2.5 UdpEchoServerHelper

As discussed in section 2.3 the nodes already have a transport layer. This section makes part of the application layer for the node. Here, a `UdpEchoServer` object is created and attached to one of the nodes as an application. Henceforth this node starts acting as the server. The start and stop time are specified.

2.6 UdpEchojClientHelper

Similarly to section 2.5, in this part of the code an object of type `UdpEchoClient` is created. The attributes were set with the `PacketSize` being set to 2048.

This is greater than the MTU which is 1500, to fit the required specification. This is then attached to the other node so it starts acting as a client.

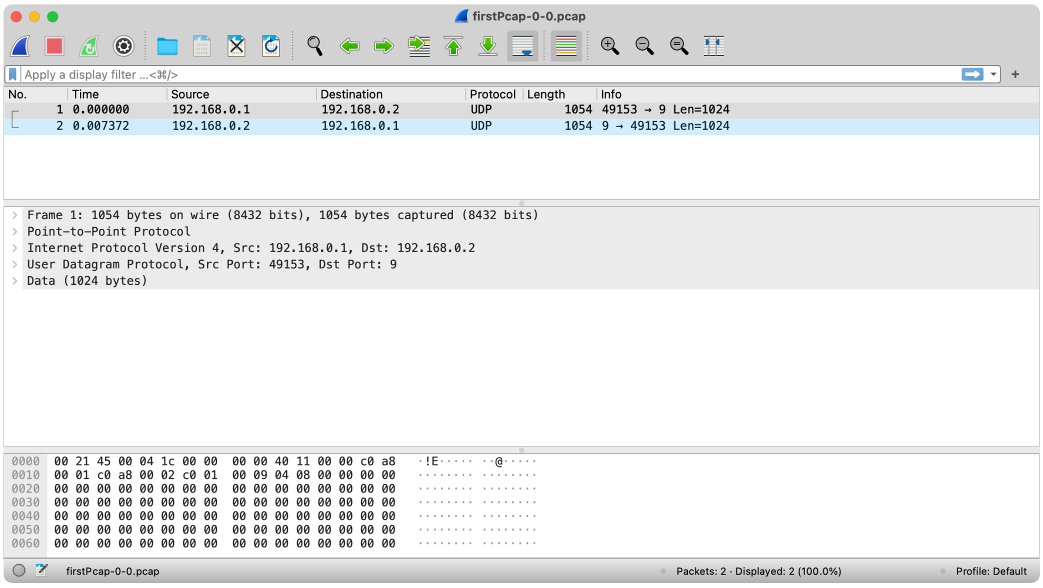
2.7 PcapHelper

When doing this section, it was necessary to learn how different classes communicate with each other in NS3. Communication within classes is done via callbacks. Here, function pointers are used to communicate the data. This also applies to communication between the different layers of the system, as most of them are represented by different instances of classes.

This call back feature was used to enable PCAP tracing without the use of the helper. First off, a PCAP file wrapper object was created. This was used to open a file called `firstPcapA`. This is the PCAP tracing file for the first node. Following that, the method `TraceConnectWithoutCallback` is used and a function pointer called `DefaultSink` is passed as one of the parameters as required by the method signature.

3 Investigation

3.1 Establishment



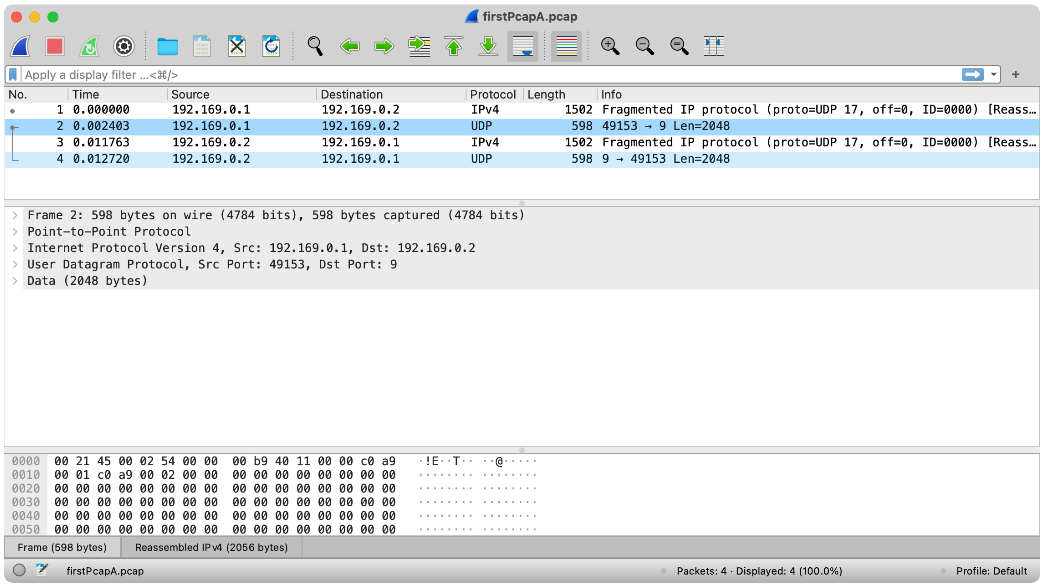
```
No.   Time       Source        Destination   Protocol Length Info
    1 0.000000   192.168.0.1   192.168.0.2   UDP        1054 49153 → 9 Len=1024
    2 0.007372   192.168.0.2   192.168.0.1   UDP        1054 9 → 49153 Len=1024
```

```
> Frame 1: 1054 bytes on wire (8432 bits), 1054 bytes captured (8432 bits)
> Point-to-Point Protocol
> Internet Protocol Version 4, Src: 192.168.0.1, Dst: 192.168.0.2
> User Datagram Protocol, Src Port: 49153, Dst Port: 9
> Data (1024 bytes)
```

```
0000  00 21 45 00 04 1c 00 00  00 00 40 11 00 00 c0 a8   ·!E·····  ··@·····
0010  00 01 c0 a8 00 02 c0 01  00 09 04 08 00 00 00 00   ········  ········
0020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········  ········
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········  ········
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········  ········
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········  ········
0060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········  ········
```

```
firstPcap-0-0.pcap          Packets: 2 · Displayed: 2 (100.0%)          Profile: Default
```

Figure 1: Original packet capture file



No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	192.169.0.1	192.169.0.2	IPv4	1502	Fragmented IP protocol (proto=UDP 17, off=0, ID=0000) [Reass…
2	0.002403	192.169.0.1	192.169.0.2	UDP	598	49153 → 9 Len=2048
3	0.011763	192.169.0.2	192.169.0.1	IPv4	1502	Fragmented IP protocol (proto=UDP 17, off=0, ID=0000) [Reass…
4	0.012720	192.169.0.2	192.169.0.1	UDP	598	9 → 49153 Len=2048

```
> Frame 2: 598 bytes on wire (4784 bits), 598 bytes captured (4784 bits)
> Point-to-Point Protocol
> Internet Protocol Version 4, Src: 192.169.0.1, Dst: 192.169.0.2
> User Datagram Protocol, Src Port: 49153, Dst Port: 9
> Data (2048 bytes)
```

```
0000  00 21 45 00 02 54 00 00  00 b9 40 11 00 00 c0 a9   ·!E··T·· ··@·····
0010  00 01 c0 a9 00 02 00 00  00 00 00 00 00 00 00 00   ········ ········
0020  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0030  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
0050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00   ········ ········
```

Frame (598 bytes)
Reassembled IPv4 (2056 bytes)
firstPcapA.pcap
Packets: 4 · Displayed: 4 (100.0%)
Profile: Default

Figure 2: Packet capture file without helper

3.2 Analysis

Figure 1 show the pcap file captured on the given simulation, while figure 2 shows the one capture without the use of any helpers. These figures confirm that the UDP protocol was used in the transport layer. The figures also show the IP addresses of each node. Moreover, one can see the bytes of data that were transferred; the data link protocol (P2P); the Internet protocol (version 4). These confirm that the situation is working as intended.

References