Performance evaluation of IEEE 802.11 MAC protocol

Markus Søvik Gunnarsson

A report submitted for the course
MobWat - Mobile Wireless Access Technology
Eurecom

May 2021

Abstract

This lab introduced hands-on experience about the IEEE 802.11 MAC protocol. The lab explored some features in the IEEE 802.11 MAC protocol, specifically goodput, throughput and the hidden terminal terminal problem. Practical experience in Virtual machines, Ubuntu, Wireshark, bash programming and web technology in general is obtained by doing this lab.

Contents

Chapter 1

Introduction

1.1 Project Scope

This lab is an exprimental/practical exercise set given to students of theMobWat course at Eurecom to get more familiar with the IEEE 802.11 MAC protocol, especially from a practical point of view. The content of the report is divided into two parts.

The first portion of the lab focuses on the understanding of the mmpact of network load on the performance of IEEE 802.11 MAC protocol, while the second portion tackles the hidden nodes problem

1.2 Report Outline

The report consist of 3 main chapters.

The report consist of an introduction (1) and a chapter for each part of the lab (2)(3). The last two chapters includes everything from implementation, results and discussion.

Chapter 2

Part 1 Impact of network load on the performance of IEEE 802.11 MAC protocol

An aspect that is worth investigating is the behavior of the IEEE 802.11 MAC protocol when varying the network load. It depends basically on the number of nodes connected to the network and on the bit rate produced by each node.

For this the goodput (the number of useful bits per unit of time forwarded by the network to a certain destination) is measured. It is calculated as the number of bits received at a destination node over the time of simulation.[1]

We consider 2 scenarios:

1. The channel reaches saturation at 2 Mbps.
2. the channel reaches saturation at 54 Mpbs. Also the MAC adapts modulation rate depending on the link quality.

2.1 Scenario 1

A plot of the goodput as a function of bit rate is shown in the figure below (2.1).

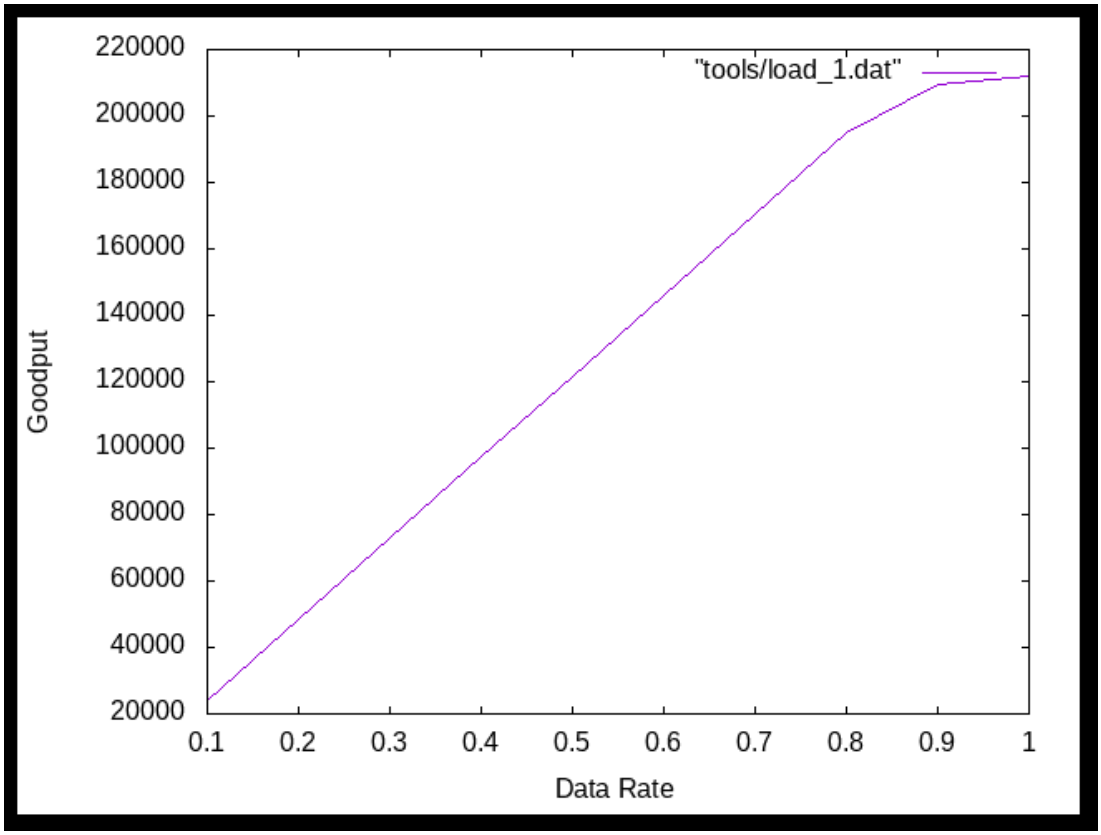

Figure 2.1: A plot of the goodput versus bit rate. The bit rate have a stepsize of 0.1 Mbps.

From the plot above (2.1) its possible to see that goodput increases linearly for the first part of the plot. When the data rate is 0.8 the increase in goodput starts to decrease. For a data rate of 1 the goodput is about 200 000 and starts to saturate.

2.2 Scenario 2

Similar to the section above a new plot of the goodput as a function of bit rate is shown in the figure below (2.2).

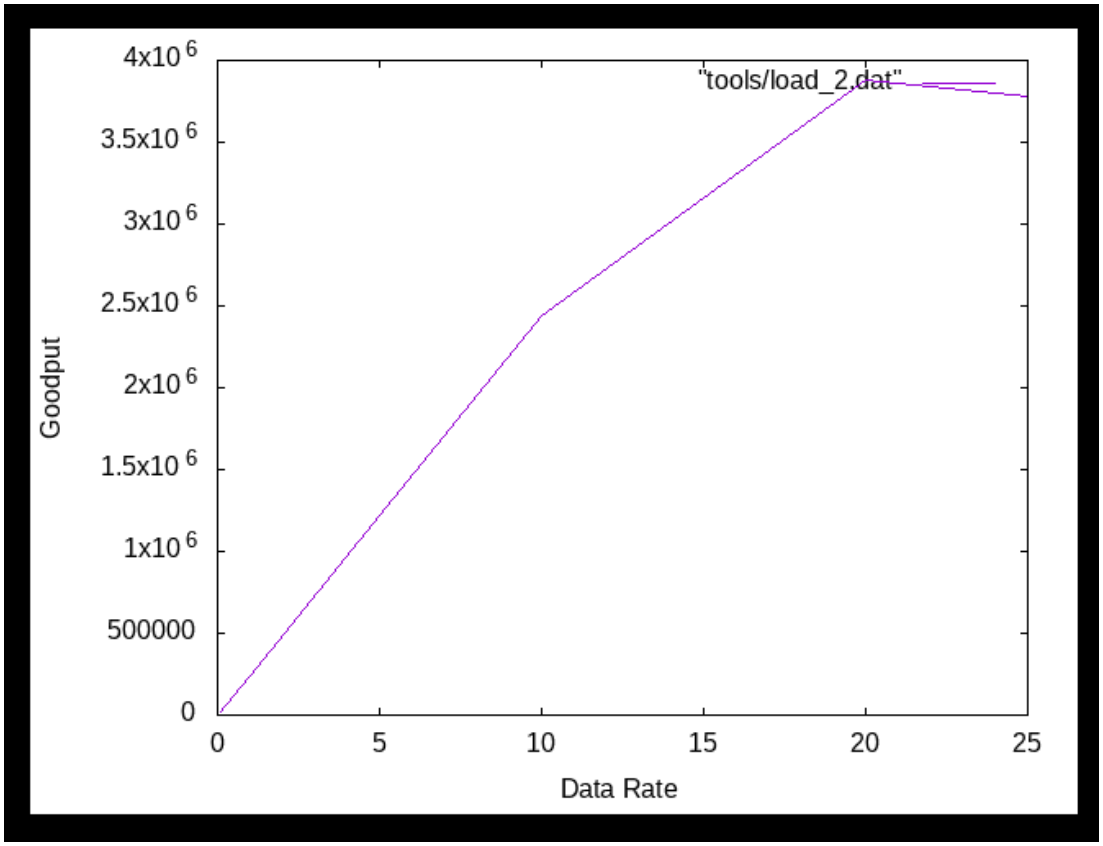

Figure 2.2: A plot of the goodput versus bit rate. The bit rate have a varying stepsize.

From the plot above (2.2) its possible to see that goodput increases linearly for the first part of the plot. When the data rate is 10 the increase in goodput starts to decrease. For a data rate of 20 the goodput is at its highest and it is about 3 875 600. Beyond that point the goodput starts to decrease for further increased data rates.

Chapter 3

Part 2 Hidden nodes problem

The hidden terminal problem occurs when there are two nodes that are outside the transmission range of each other but will each transmit to a node that is shared between them. In Figure 3.1 below, nodes 2 and 0 cannot sense each other's transmissions. If both nodes 2 and 0 were to transmit to node 1 at the same time a collision would occur. The 802.11 protocol addresses this problem by adding an optional Request to Send/Clear to Send (RTS/CTS) transmission before the actual data is transferred.[1]

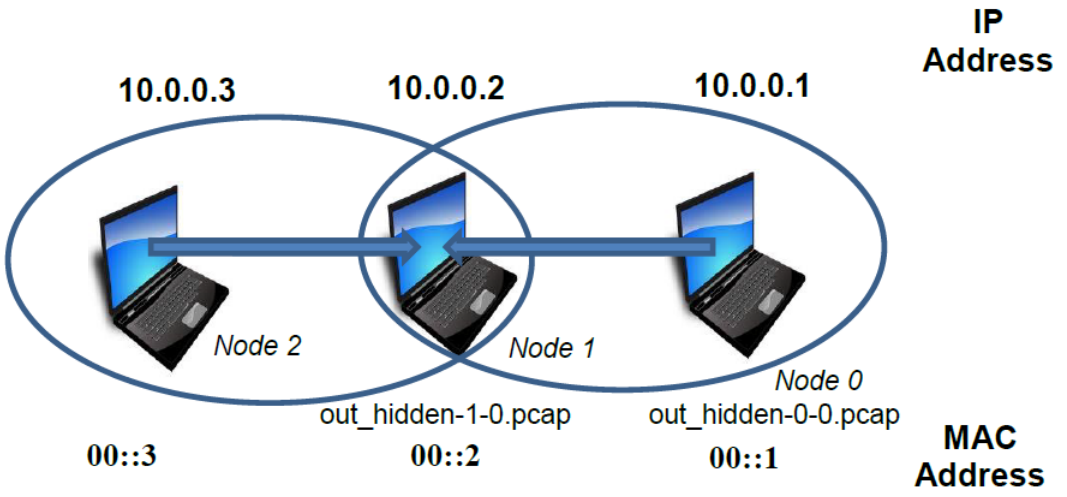

Figure 3.1: An illustration of how the network layout is structured. Picture taken from - [1]

Information which is featured in figure 3.1: IP addresses, MAC addresses, Node number and PCAP-trace name.

3.1 RTS/CTS handhshake disabled

By using wireshark, it is possible to look at frames that is sent in transmission. Figure 3.2 is a screenshot of a wireshark capture between Node 2 and Node 1.

No.	Time	Source	Destination	Info
1	0.00…	00:00:00_0…	Broadcast	Who has 10.0.0.2? Tell 10.0.0.1
2	0.00…	00:00:00_0…	00:00:00_00:00:01	10.0.0.2 is at 00:00:00:00:00:02
3	0.00…		00:00:00_00:00:02 (…	Acknowledgement, Flags=……..
4	0.00…	10.0.0.1	10.0.0.2	49153 → 9 Len=10
5	0.00…		00:00:00_00:00:01 (…	Acknowledgement, Flags=……..
6	0.00…	00:00:00_0…	Broadcast	Who has 10.0.0.2? Tell 10.0.0.3
7	0.00…	00:00:00_0…	00:00:00_00:00:03	10.0.0.2 is at 00:00:00:00:00:02
8	0.00…		00:00:00_00:00:02 (…	Acknowledgement, Flags=……..
9	0.00…	10.0.0.3	10.0.0.2	49153 → 9 Len=10
10	0.00…		00:00:00_00:00:03 (…	Acknowledgement, Flags=……..
11	0.00…	00:00:00_0…	Broadcast	Who has 10.0.0.1? Tell 10.0.0.2
12	0.00…	00:00:00_0…	00:00:00_00:00:02	10.0.0.1 is at 00:00:00:00:00:01
13	0.00…		00:00:00_00:00:01 (…	Acknowledgement, Flags=……..
14	0.00…	10.0.0.2	10.0.0.1	Destination unreachable (Port unr
15	0.01…		00:00:00_00:00:02 (…	Acknowledgement, Flags=……..
16	0.01…	00:00:00_0…	Broadcast	Who has 10.0.0.3? Tell 10.0.0.2
17	0.01…	00:00:00_0…	00:00:00_00:00:02	10.0.0.3 is at 00:00:00:00:00:03
18	0.01…		00:00:00_00:00:03 (…	Acknowledgement, Flags=……..
19	0.01…	10.0.0.2	10.0.0.3	Destination unreachable (Port unr

Figure 3.2: a screenshot of a wireshark capture between Node 2 and Node 1

Broadcasting frames, Acknowledgement frames and data frames are examples of frames that is sent in this wireshark capture (3.2).

Figure 3.3 is a screenshot of a wireshark capture between Node 2 and Node 1.

No.	Time	Source	Destination	Info
1	0.00…	00:00:00_0…	Broadcast	Who has 10.0.0.2? Tell 10.0.0.1
2	0.00…	00:00:00_0…	00:00:00_00:00:01	10.0.0.2 is at 00:00:00:00:00:02
3	0.00…		00:00:00_00:00:02 (…	Acknowledgement, Flags=……..
4	0.00…	10.0.0.1	10.0.0.2	49153 → 9 Len=10
5	0.00…		00:00:00_00:00:01 (…	Acknowledgement, Flags=……..
6	0.00…	00:00:00_0…	00:00:00_00:00:03	10.0.0.2 is at 00:00:00:00:00:02
7	0.00…		00:00:00_00:00:03 (…	Acknowledgement, Flags=……..
8	0.00…	00:00:00_0…	Broadcast	Who has 10.0.0.1? Tell 10.0.0.2
9	0.00…	00:00:00_0…	00:00:00_00:00:02	10.0.0.1 is at 00:00:00:00:00:01
10	0.00…		00:00:00_00:00:01 (…	Acknowledgement, Flags=……..
11	0.01…	10.0.0.2	10.0.0.1	Destination unreachable (Port unr
12	0.01…		00:00:00_00:00:02 (…	Acknowledgement, Flags=……..
13	0.01…	00:00:00_0…	Broadcast	Who has 10.0.0.3? Tell 10.0.0.2
14	0.01…		00:00:00_00:00:03 (…	Acknowledgement, Flags=……..
15	0.01…	10.0.0.2	10.0.0.3	Destination unreachable (Port unr
16	1.00…	10.0.0.1	10.0.0.2	49154 → 12345 Len=1400
17	1.01…	10.0.0.1	10.0.0.2	49154 → 12345 Len=1400
18	1.02…	10.0.0.1	10.0.0.2	49154 → 12345 Len=1400
19	1.02…	10.0.0.1	10.0.0.2	49154 → 12345 Len=1400

Figure 3.3: a screenshot of a wireshark capture between Node 1 and Node 0

By comparing figure 3.2 and 3.3 it is possible to see that the main difference here is that there is one less broadcast.

The broadcast that is missing from 3.3 is "Who has 10.0.0.2? Tell 10.0.0.3". This creates a chance for collision. From figure 3.3 it can be seen that frame 6 is the reply from 10.0.0.2 to 10.0.0.3, but 10.0.0.1 can not see the Broadcast from 10.0.0.3 which requested this information. The next frames is where the collision happens, when

both Node 0 and Node 2 tries to talk to Node 1 at the same time.

3.2 RTS/CTS handhshake enabled

Similar to the section above figure 3.2 is a screenshot of a wireshark capture between Node 2 and Node 1. This time the RTS/CTS handshake flag is enabled.

No.	Time	Source	Destination	Info
1	0.00…	00:00:00_0…	Broadcast	Who has 10.0.0.2? Tell 10.0.0.1
2	0.00…	00:00:00_0…	00:00:00_00:00…	10.0.0.2 is at 00:00:00:00:00:02
3	0.00…		00:00:00_00:00…	Acknowledgement, Flags=........
4	0.00…	10.0.0.1	10.0.0.2	49153 → 9 Len=10
5	0.00…		00:00:00_00:00…	Acknowledgement, Flags=........
6	0.00…	00:00:00_0…	Broadcast	Who has 10.0.0.2? Tell 10.0.0.3
7	0.00…	00:00:00_0…	00:00:00_00:00…	10.0.0.2 is at 00:00:00:00:00:02
8	0.00…		00:00:00_00:00…	Acknowledgement, Flags=........
9	0.00…	10.0.0.3	10.0.0.2	49153 → 9 Len=10
10	0.00…		00:00:00_00:00…	Acknowledgement, Flags=........
11	0.00…	00:00:00_0…	Broadcast	Who has 10.0.0.1? Tell 10.0.0.2
12	0.00…	00:00:00_0…	00:00:00_00:00…	10.0.0.1 is at 00:00:00:00:00:01
13	0.00…		00:00:00_00:00…	Acknowledgement, Flags=........
14	0.00…	10.0.0.2	10.0.0.1	Destination unreachable (Port unreach
15	0.01…		00:00:00_00:00…	Acknowledgement, Flags=........
16	0.01…	00:00:00_0…	Broadcast	Who has 10.0.0.3? Tell 10.0.0.2
17	0.01…	00:00:00_0…	00:00:00_00:00…	10.0.0.3 is at 00:00:00:00:00:03
18	0.01…		00:00:00_00:00…	Acknowledgement, Flags=........
19	0.01…	10.0.0.2	10.0.0.3	Destination unreachable (Port unreach

Figure 3.4: a screenshot of a wireshark capture between Node 2 and Node 1

Figure3.4 shows the same selection of frames as in the wireshark screenshots from the previous section. Here two new frames are introduced as well; Request-to-send and

Figure 3.3 is a screenshot of a wireshark capture between Node 2 and Node 1.

No.	Time	Source	Destination	Info
1	0.00…	00:00:00_0…	Broadcast	Who has 10.0.0.2? Tell 10.0.0.1
2	0.00…	00:00:00_0…	00:00:00_00:00…	10.0.0.2 is at 00:00:00:00:00:02
3	0.00…		00:00:00_00:00…	Acknowledgement, Flags=........
4	0.00…	10.0.0.1	10.0.0.2	49153 → 9 Len=10
5	0.00…		00:00:00_00:00…	Acknowledgement, Flags=........
6	0.00…	00:00:00_0…	00:00:00_00:00…	10.0.0.2 is at 00:00:00:00:00:02
7	0.00…		00:00:00_00:00…	Acknowledgement, Flags=........
8	0.00…	00:00:00_0…	Broadcast	Who has 10.0.0.1? Tell 10.0.0.2
9	0.00…	00:00:00_0…	00:00:00_00:00…	10.0.0.1 is at 00:00:00:00:00:01
10	0.00…		00:00:00_00:00…	Acknowledgement, Flags=........
11	0.01…	10.0.0.2	10.0.0.1	Destination unreachable (Port unreach
12	0.01…		00:00:00_00:00…	Acknowledgement, Flags=........
13	0.01…	00:00:00_0…	Broadcast	Who has 10.0.0.3? Tell 10.0.0.2
14	0.01…		00:00:00_00:00…	Acknowledgement, Flags=........
15	0.01…	10.0.0.2	10.0.0.3	Destination unreachable (Port unreach
16	1.00…	00:00:00_0…	00:00:00_00:00…	Request-to-send, Flags=........
17	1.00…		00:00:00_00:00…	Clear-to-send, Flags=........
18	1.00…	10.0.0.1	10.0.0.2	49154 → 12345 Len=1400
19	1.01…		00:00:00_00:00…	Acknowledgement, Flags=........

Figure 3.5: a screenshot of a wireshark capture between Node 1 and Node 0

By comparing figure 3.4 and 3.5 it is still possible to see that the main difference here is that the same broadcast is missing as from the previous section. In frame 11 Node 0 get a reply from Node 1 that the destination is unreachable. A request-to-send flag is raised from Node 0 in frame 16 and a clear-to-send flag is replied from Node 1 frame 17.

3.3 Throughput

Figure 3.6 and 3.7 shows the throughput in from both Nodes(0 and 2) to Node 1 in both cases(RTS/CTS handhshake enabled/disabled)

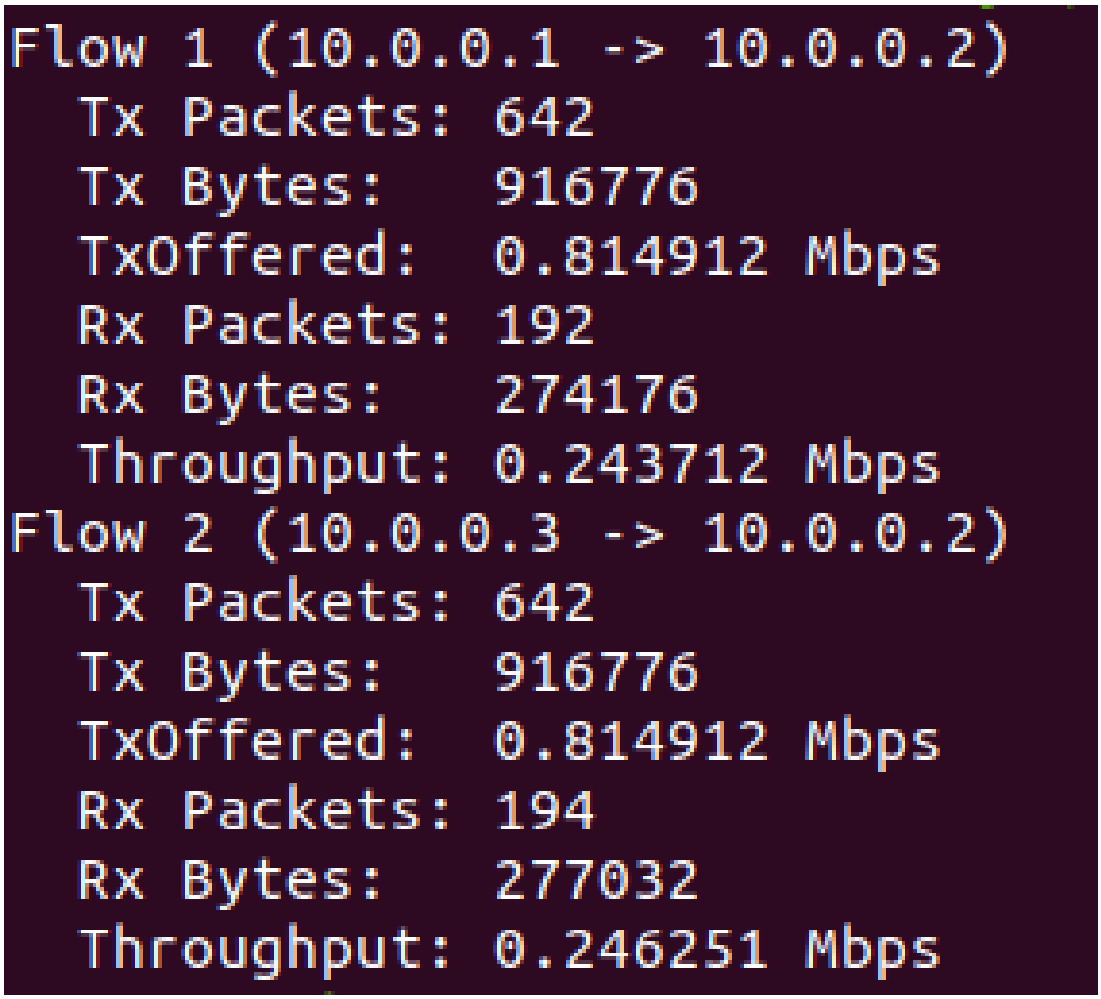

Figure 3.6: Throughput in from both Nodes(0 and 2) to Node 1 with RTS/CTS handhshake disabled.

Figure 3.6 shows that the throughput of flow 1 (From Node 0 to 1) is 0.243 Mbps and the throughput of flow 2 (From Node 2 to 1) is 0.246 Mbps

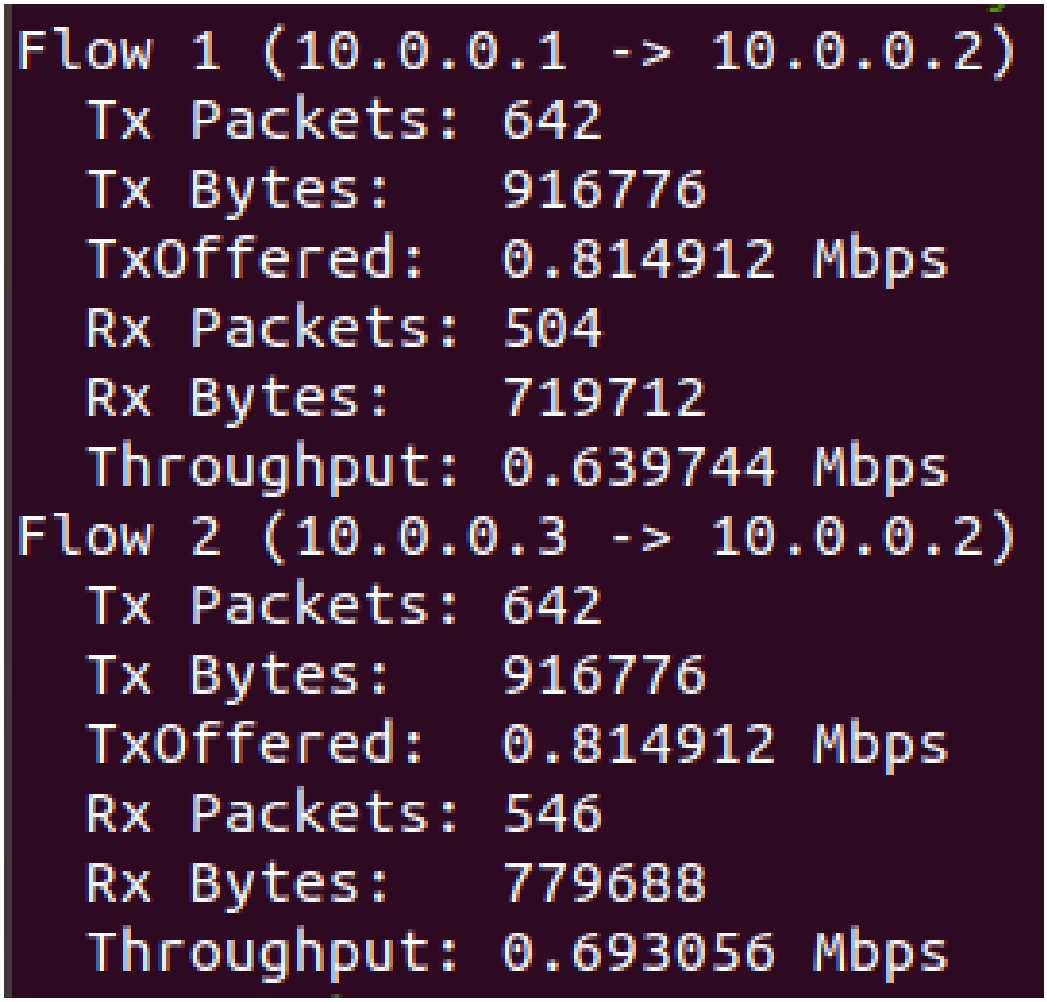

Figure 3.7: Throughput in from both Nodes(0 and 2) to Node 1 with RTS/CTS handhshake enabled.

Figure 3.7 shows that the throughput of flow 1 (From Node 0 to 1) is 0.639 Mbps and the throughput of flow 2 (From Node 2 to 1) is 0.693 Mbps.

By comparing figure 3.6 and 3.7 it is clear that RTS/CTS handhshake enabled improves the throughput of both communication channels. A reason for this is that the RTS/CTS handhshake creates less collitions.

Bibliography

[1] Prof. Jérôme Härri, Jin Yang. *Performance evaluation of IEEE 802.11 MAC protocol 2021*. Eurecom, 2021.
(cited on pages 2 and 5)