# Demonstration and Study of Bufferbloat in 5G and solution by implementing AQM algorithm CODEL

**Team Members:**
Sree Prathyush Chinta (CS19BTECH11043)
Medha Rachel Panna (CS22MTECH11003)
Manisha Mahapatra (CS22MTECH14009)
SOHAN D P (CS19BTECH11028 )

## Problem Statement:

Bufferbloat is an unfavorable condition that frequently occurs in the network and causes end-to-end latency, jitter and performance reduction. It occurs as a result of lengthy waiting time that absorbs a significant quantity of traffic in a clogged path. Network nodes use big buffers to stop packet losses in such cases. Even very high-speed networks can become virtually unusable for many interactive activities, such as voice over IP (VoIP), audio streaming, online gaming, and even basic web browsing, when a router or switch is set to use very large buffers.

## Project Description:

There are various scheduling algorithms that are trying to tackle the bufferbloat problem. We aim to showcase the bufferbloat problem in the 5G scenario with the help of ns-3. Taking the base paper on 'Controlled Delay Active Queue Management', AQM algorithm that is CoDel (Controlled Delay) is used to solve the bufferbloat problem.

There are two scenarios, first will be to demonstrate the bufferbloat problem in ns3. There is a single UE and a gNB with two downlink flows, one flow is mice flow and the other is elephant flow. In the second scenario, the AQM algorithm that is CoDel is used to solve this problem of bufferbloat.

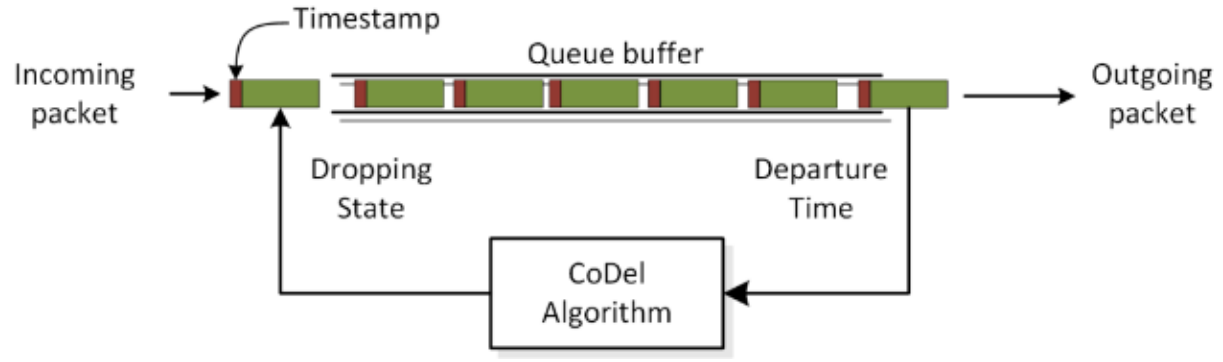



Fig: CoDel Algorithm

## CoDel:

CoDel is a packet layover time based algorithm. It discards the packets to let the sender's congestion control algorithm know that there is too much buffering going on. It is one of the most well-known AQM and a frequently used fix for the bufferbloat issue. It is based on two variables: interval time and target time. Every packet that is enqueued includes a timestamp(through CoDel). Every packet's layover duration is measured during the interval and the minimum time value is kept. The following packet is deleted as a means to alert the sender that excessive queuing is occurring if, after the interval time, the minimum saved time value is more than the desired time.

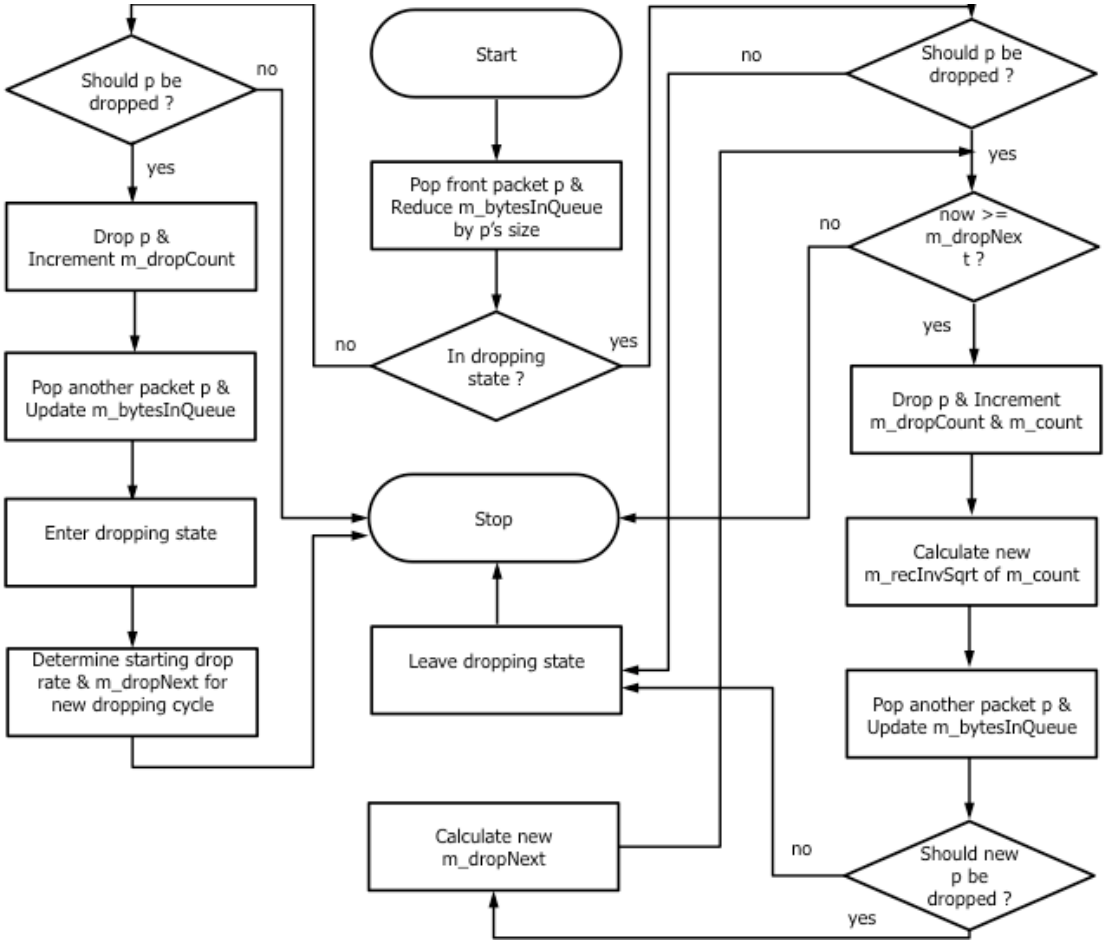



Fig : CoDel Algorithm Flowchart

## Project Implementation Details:

## Modification:

The implementation of the scenario involves creation of a single Gnb and single UE connection establishment with the respective parameters described in the simulation parameters part of the report.
We have enabled the RlcUm log component in the beginning of the code to collect the required RLC logs to verify the bufferbloat problem.

```
LogComponentEnable ("LteRlcUm", LOG_LEVEL_ALL);
```

We then create two downlink clients to define the characteristics of the two downlink flows. The destination for both of these flows is set to be the same UE i.e. both flows have the same source and destination. We then modify the respective lambda values for both the downlink clients to ensure that the elephant flow( i.e. flow 2 in our case ) always has a greater lambda than the mice flow ( flow 1 in our case) . We have set the mice flow to have lambda 1.0 ( 1 packet per second ) while varying the lambda value of flow 2 to get our readings.

```
lambda1 = 1.0
lambda2 = 30000.0
dlClient.SetAttribute ("Interval", TimeValue (Seconds (1.0 / lambda1)));
dlClient2.SetAttribute ("Interval", TimeValue (Seconds (1.0 / lambda2)));
```

We have then established the flow monitor to monitor the respective flows and give their respective stats in the output file which we use for our data analysis. We have included a code line to monitor the packet to the UE and generate the respective pcaps to analyze the packet flow.

```
internet.EnablePcapIpv4("results.pcap", ueNodes.Get (0)->GetId (), 1, true);
```

Implementation of the CoDel algorithm involves making changes in the lte-rlc-um.cc file where we directly manipulate the RLC transmission buffer.
The beginning of the code involves declaring the global variables that we are going to be using during the codel process with them referring to their respective counterparts in the CoDel algorithm description in the Simulation setup of the report.

```
Time cur = Time(0);
int count_new = 0;
int next_drop_time =0;
Time cur_qdelay =Time(0);
Time interval = Time(100000000);
Time threshold = Time(100000000);
```

In this code we have to apply CoDel on the RLC buffer which is a vector object with the name m_txBuffer.

```
std::vector < TxPdu > m_txBuffer;
```

We then make changes in the **DoNotifyTxOpportunity** method of the LteRlcUm class which deals with the transmission of packets from the RLC buffer so that before a packet moves ahead to the transmission phase ( i.e either transmission of segmentation ) we calculate the queue delay experienced by the packet and store it in the cur_qdelay variable by subtracting the timestamp stored in the packet from the current simulation time.

```
cur_qdelay = Simulator::Now() - firstSegmentTime;
```

This is followed by making changes in the **DoTransmitPdcpPdu** method of the LteRlcUm class which deals with the incoming PDCP PDU packets and stores them in the RLC transmission buffer. Here we use the  cur_qdelay that is being calculated at every transmission to verify if it crosses the threshold and apply the CoDel steps respectively.  Here we check if the queue delay is greater than the threshold and if it is we enter the next part where we store the time when the first packet with a queue delay greater than the threshold was noted and in the case that we see a continuous flow of packets which have a queue delay of greater than threshold for time period greater than the interval time we start to drop the packets appearing at the respective drop times calculated by the algorithm. In the case that we see a packet having a queue delay less than the threshold before the interval limit we reset the respective values. Here the dropping of packets is simply done by returning from the function without adding the packet to the transmission queue.

```
if (cur_qdelay > threshold)
{
    if (cur.GetMilliSeconds() == 0)
    {
        cur = Simulator::Now();
```

```
    }
    else if( Simulator::Now() - cur > interval )
    {
        if(count_new == 0)
        {
            count_new+=1;
            next_drop_time = Simulator::Now().GetMilliSeconds() +
interval.GetMilliSeconds()/sqrt(count_new);
            return;
        }
        else if( Simulator::Now().GetMilliSeconds()>=next_drop_time )
        {
             count_new+=1;
             next_drop_time = Simulator::Now().GetMilliSeconds() +
interval.GetMilliSeconds()/sqrt(count_new);
             return;
        }

}
}
else
{
        count_new = 0;
        next_drop_time=0;
        cur = Time(0);
}
```

## Challenges faced:

- Using the grep command to search through numerous files was a challenge. Additionally, make sure that any modifications made to one file do not have an impact on the other file.
- Examining the RLC logs to confirm that the same RLC buffer is used for both downlink flows.
- Locating the RLC buffer that has to be modified and verifying that any modifications made don't interfere with the packet-receiving procedure.
- Initially our idea was to implement a bufferbloat on the router that will be connected from the gNB to the internet through UPF. We ultimately dropped this strategy after understanding through our research that NS3 has no UPF implementation.
- Verification of logs at every step to ensure that the code was functioning properly.

# Simulation/Experimental Setup:

# Test Scenario:

### Phase 1: Simulation of bufferbloat problem

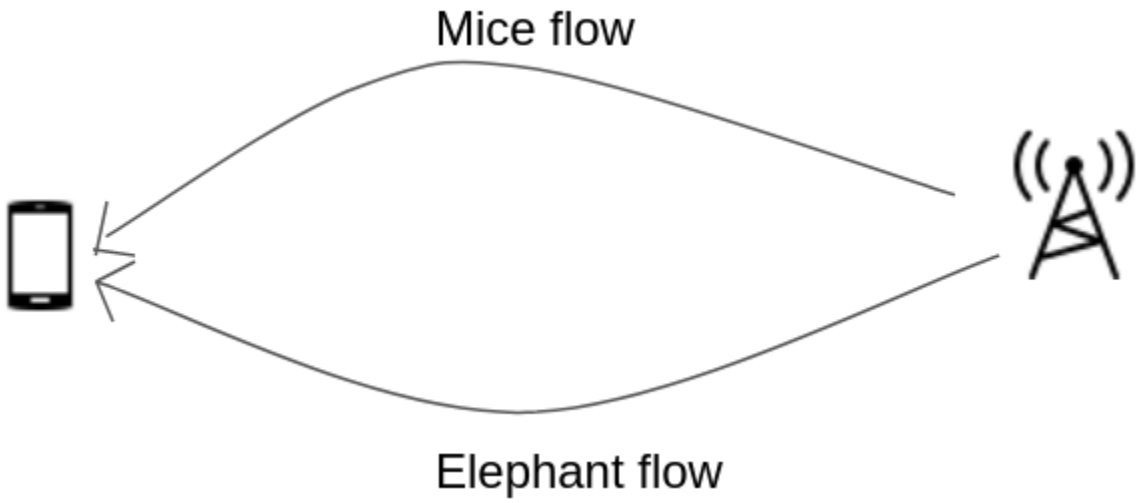



Fig: Scenario 1 (2 downlink flows)

Simulation of bufferbloat problem in ns3. There are two downlink flows with single UE and a gNB, by setting 1 packet per sec for mice flow and above 15,000 packets per sec for elephant flow. We refer to the mice flow in this scenario as the vanilla flow ( since it consists of the bufferbloat problem without any solution implemented).
By looking into the RLC logs, we observe that packets of both mice flow and elephant flow are added to the same RLC buffer verified by comparing their respective LCIDs (two different RLC entities have different LCIDs). The above observations show that our simulation perfectly encapsulates the bufferbloat problem.

```
 8 M_LCID : 3
 9 Tx Buffer: New packet added
10 NumOfBuffers = 1
11 txBufferSize = 1530
12 Send ReportBufferStatus = 1532, 0
13 LteRlcUm:DoTransmitPdcpPdu(0x555c4198c120, 1, 3, 1530)
14 M_LCID : 3
15 Tx Buffer: New packet added
16 NumOfBuffers = 2
17 txBufferSize = 3060
18 Send ReportBufferStatus = 3064, 0
19 LteRlcUm:DoTransmitPdcpPdu(0x555c4198c120, 1, 3, 1530)
20 M_LCID : 3
21 Tx Buffer: New packet added
22 NumOfBuffers = 3
23 txBufferSize = 4590
24 Send ReportBufferStatus = 4596, 0
25 LteRlcUm:DoTransmitPdcpPdu(0x555c4198c120, 1, 3, 1530)
26 M_LCID : 3
27 Tx Buffer: New packet added
28 NumOfBuffers = 4
29 txBufferSize = 6120
30 Send ReportBufferStatus = 6128, 0
31 LteRlcUm:DoTransmitPdcpPdu(0x555c4198c120, 1, 3, 1530)
32 M_LCID : 3
33 Tx Buffer: New packet added
34 NumOfBuffers = 5
35 txBufferSize = 7650
36 Send ReportBufferStatus = 7660, 0
37 LteRlcUm:DoTransmitPdcpPdu(0x555c4198c120, 1, 3, 1530)
38 M_LCID : 3
39 Tx Buffer: New packet added
```

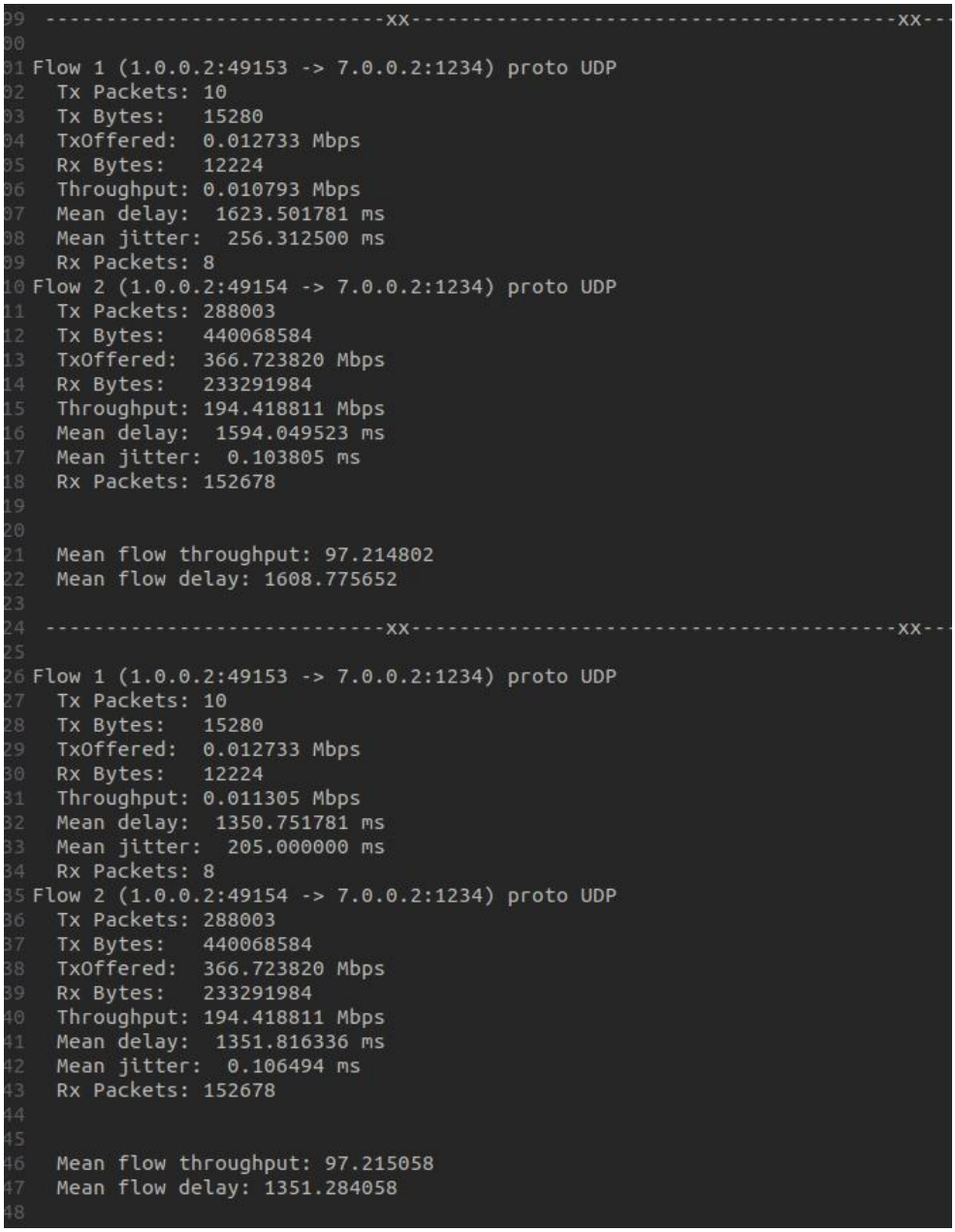
## Phase 2: Solution to Bufferbloat Problem

## Implementation 1:

Due to demanding TCP's congestion control characteristics, bulky services will attempt to dominate access to the wireless resources, bloating the buffers in their path and preventing a fast low-latency packet delivery of any other flow that travels the same data path. Huge buffers ensure that all radio channels are used. Due to the dynamic nature of the radio data link in cellular networks, which changes the radio channel capacity, a

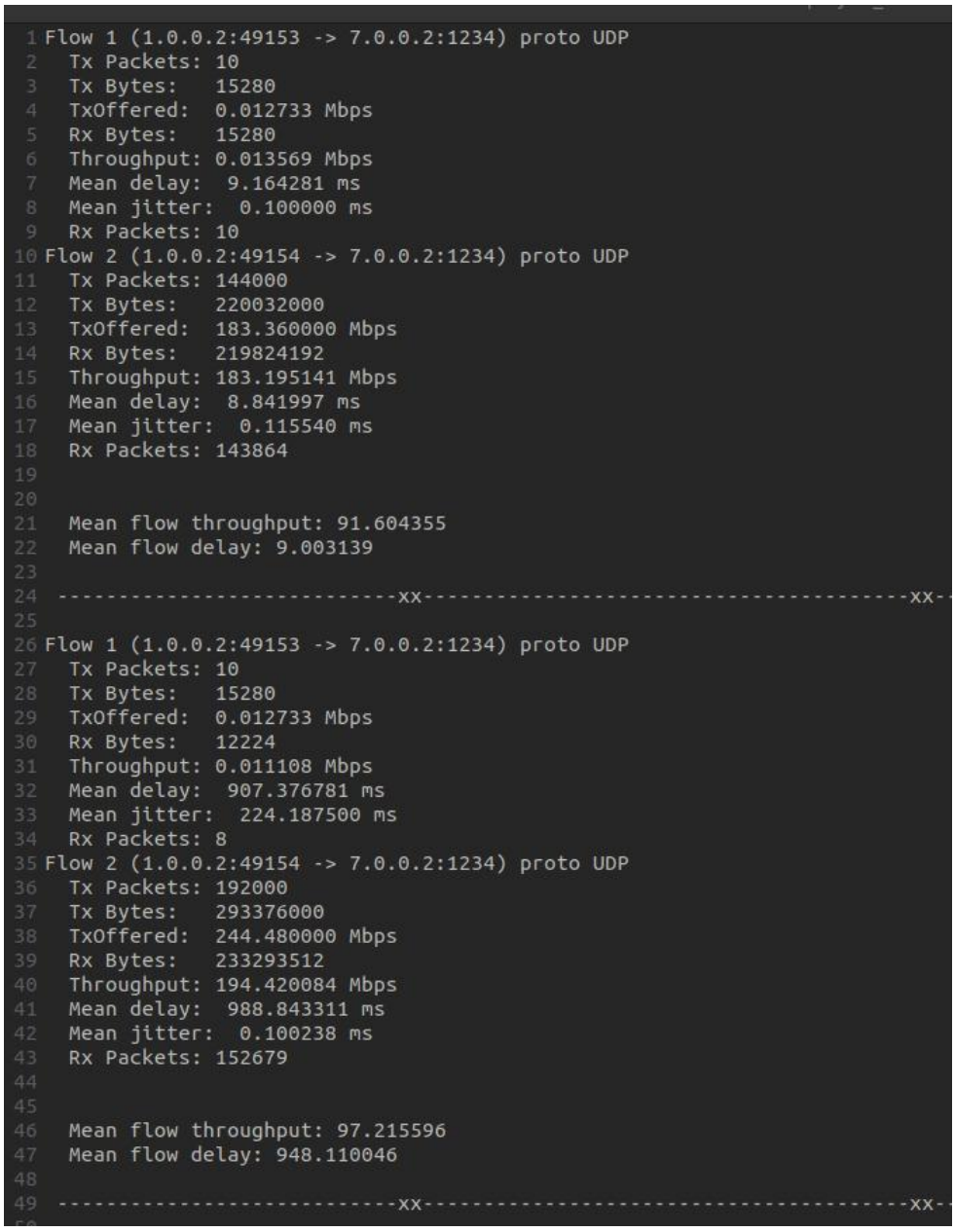
resource under-utilization scenario could occur even though the buffer size is controlled to prevent buffer bloating.

Therefore in order to fully utilize the radio channel and to reduce the delay we vary RLC buffer size which thereby changes the delay. In this implementation we plan to understand the relation between buffer size and the occurring delay for a packet with the help of an experiment ( they are inversely proportional ).



## Implementation 2:

Screenshot given below shows the results of CoDel implemented bufferbloat scenario with respect to flow details. First two downlink flows with single UE and a gNB, by setting 1 packet per sec for mice flow and another by setting 15,000 packets per sec for elephant flow.
Next flow is with single UE and gNB by setting elephant flow as 20,000 packets per second.



```
Flow 1 (1.0.0.2:49153 -> 7.0.0.2:1234) proto UDP
  Tx Packets: 10
  Tx Bytes:   15280
  TxOffered:  0.012733 Mbps
  Rx Bytes:   15280
  Throughput: 0.013569 Mbps
  Mean delay:  9.164281 ms
  Mean jitter:  0.100000 ms
  Rx Packets: 10
Flow 2 (1.0.0.2:49154 -> 7.0.0.2:1234) proto UDP
  Tx Packets: 144000
  Tx Bytes:   220032000
  TxOffered:  183.360000 Mbps
  Rx Bytes:   219824192
  Throughput: 183.195141 Mbps
  Mean delay:  8.841997 ms
  Mean jitter:  0.115540 ms
  Rx Packets: 143864


  Mean flow throughput: 91.604355
  Mean flow delay: 9.003139

----------------------------xx----------------------------------------xx--

Flow 1 (1.0.0.2:49153 -> 7.0.0.2:1234) proto UDP
  Tx Packets: 10
  Tx Bytes:   15280
  TxOffered:  0.012733 Mbps
  Rx Bytes:   12224
  Throughput: 0.011108 Mbps
  Mean delay:  907.376781 ms
  Mean jitter:  224.187500 ms
  Rx Packets: 8
Flow 2 (1.0.0.2:49154 -> 7.0.0.2:1234) proto UDP
  Tx Packets: 192000
  Tx Bytes:   293376000
  TxOffered:  244.480000 Mbps
  Rx Bytes:   233293512
  Throughput: 194.420084 Mbps
  Mean delay:  988.843311 ms
  Mean jitter:  0.100238 ms
  Rx Packets: 152679


  Mean flow throughput: 97.215596
  Mean flow delay: 948.110046

----------------------------xx----------------------------------------xx--
```

Initially, there is an undesirable high delay on mice flow caused by varying other network traffic that is the elephant flow causing bufferbloat problem. This problem is solved by implementing the CoDel algorithm on the RLC buffer. The algorithm given below is the implementation of CoDel.

## Algorithm:

**Step 1:**

When a packet is enqueued we attach a timestamp to specify the enqueue time.

**Step 2:**

Calculating the current queue delay for each packet that is being sent. (cur_qdelay = dequeue time - enqueue time)

**Step 3:**

When cur_qdelay > threshold for the 1st time, store the current time of the simulation(1st observed delay). This implies that there is a possibility of congestion occurring in the queue.

**Step 4:**

Continue monitoring the current queue delay of the packets and the moment that the current time - 1st observed delay('cur' in the code) > interval time, we enter the dropping phase.

If at any point the current queue delay is less than the threshold before we reach the interval time we don't enter the dropping phase and reset the values(1st observed delay = 0, count_new = 0, next_drop_time = 0).

**Step 5:**

When cur_qdelay < threshold then comes out of the dropping phase, the interval timer is reset. It will set again when waiting for the cur_qdelay>threshold.

**Dropping phase:**

In this phase, we drop the packet that cause to enter this phase while increasing the drop packet counter by one and then calculate the next drop time which is used to calculate the next time at which a packet should be dropped, this calculation given by

next_drop_time = current_time + interval/sqrt(count_new).

## Simulation Parameter:

| Simulation Parameter | Value |
|---|---|
| Number of UEs | 1; 2 Downlink UDP Flow per UE from the Remote Host. |
| Number of gNBs | 1 |
| Locations of UE (in meters) | (0,10) |
| Base Station position | (0,0) |
| RLC mode | UM mode (default in ns3) |
| gNB Tx Power | 43 dBm |
| S1-U Link Delay between gNodeB and P-GW | 2 ms |
| P2P link between P-GW and Remote Host | Data Rate: 10 Gbps<br>Link Delay: 5 ms |
| Channel model | 3GPP, LoS |
| Channel bandwidth | 50 MHz |
| Central frequency | 6 GHz for numerology 1 |
| Scenario | RMa_LoS |
| Shadowing | disabled |
| Numerologies | 1 (FR1: 6GHz) |
| Application Type | UDP Client and UDP Server |
| BandWidth Part | 1 bandwidth part (1 for DL).<br>Create one component carrier (CC) and set the parameters “Numerology”, ”Pattern”, and ”TxPower”. |
| RLC MaxTxBufferSize | 999999999 |
| Antennas for all the UEs | NumRows: 2<br>NumColumns: 4<br>AntennaElement: IsotropicAntennaModel |
| Antennas for all the gNbs | NumRows: 4<br>NumColumns: 8 |

| | |
|---|---|
| | AntennaElement: IsotropicAntennaModel |
| BeamformingMethod | DirectPathBeamforming |
| Error Model | NrEesmIrT1 |

| | |
|---|---|
| Packet Size | 1500 bytes |
| AmcModel | ShannonModel |
| Height of Base Station and UE | 10 Meters / 1.5 Meters |
| Total simulation time | 10 seconds |

## Performance Metrics:

In order to observe the bufferbloat problem, in elephant flow we compare packets lost with the packets sent per second in both implementations, one with vanilla flow and the other another where we implement CoDel to improve performance of the traffic on the network.

Further, comparing delay with respect to packets sent per second in elephant flow in both the implementations that is vanilla flow and CoDel implementation. The delay varies with the number of packets transmitted per second, showing possibilities of bufferbloat. Comparing the delay as the RLC buffer changes will show the bufferbloat issue.

## Performance Result:

The below two graphs have been generated for the RLC buffer max size of 999999999.

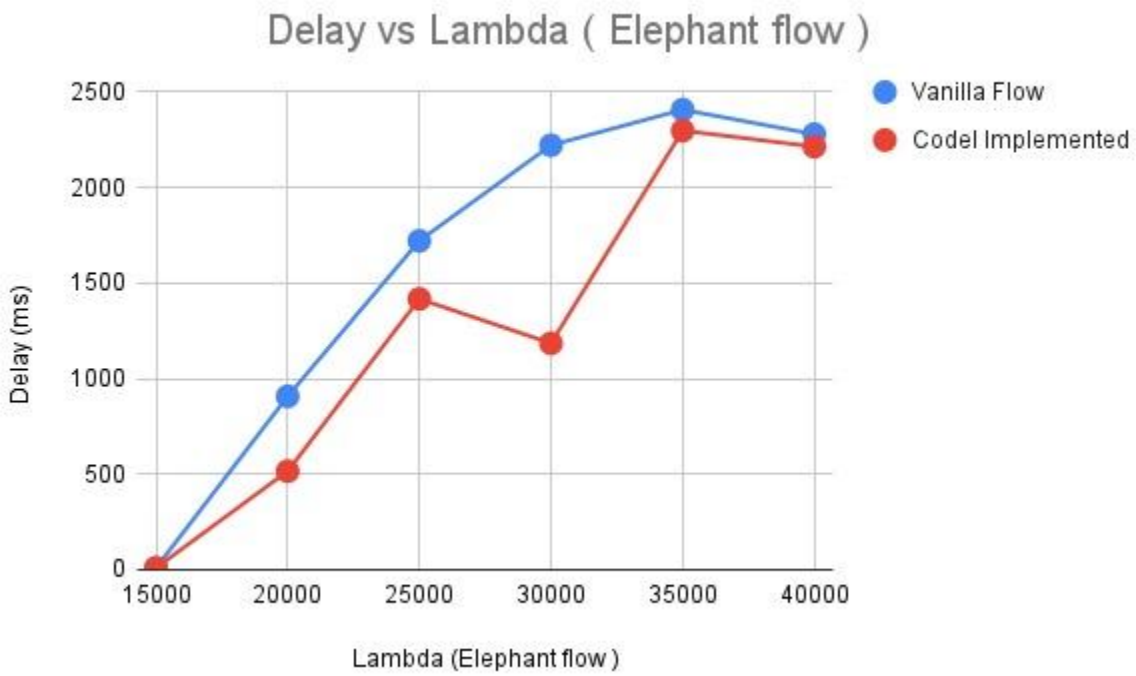



Fig: Delay vs Lambda

The two flows in the above graph led us to the conclusion that the delay in the case of the vanilla flow is greater than the delay after the implementation of Codel.

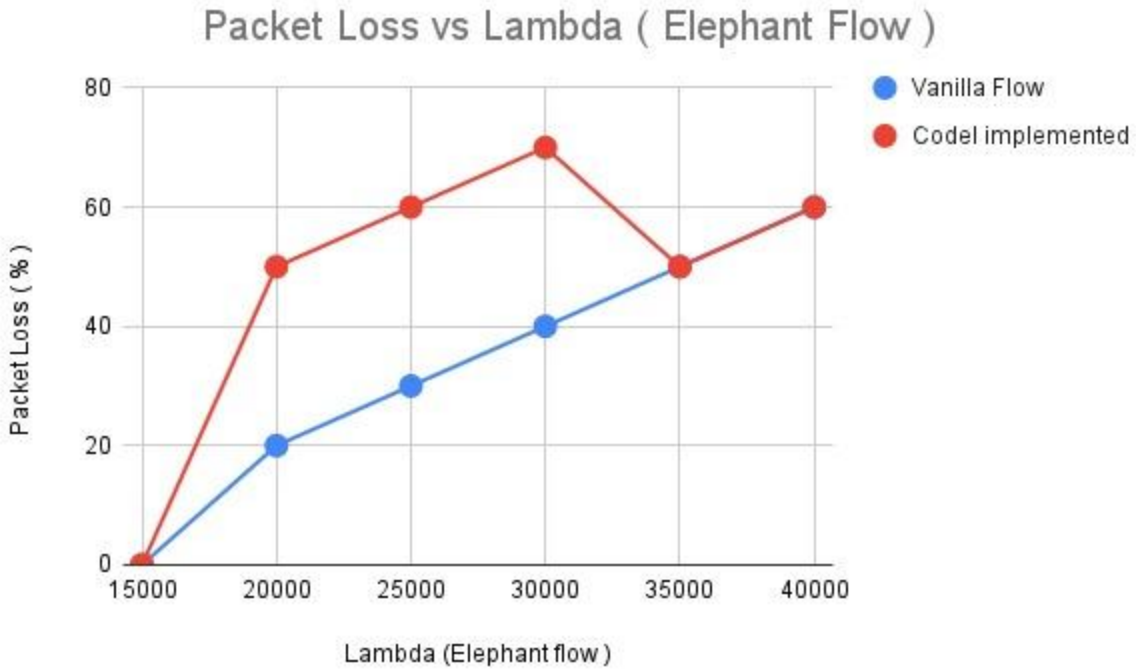



Fig: Packet Loss vs Lambda

The above graph shows that the packet lost in case of CoDel implementation is higher than the vanilla flow.

The two graphs above demonstrate that the delay decreases when Codel is used, but it's at the expense of greater packet loss when compared to a vanilla flow.

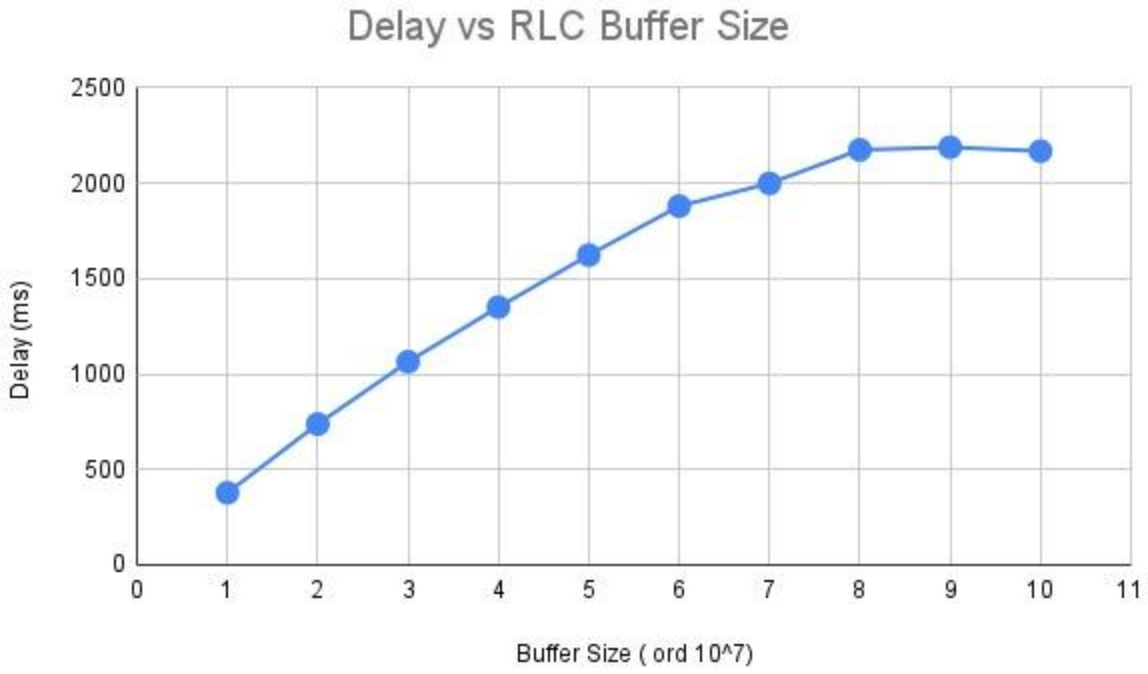



Fig: Delay vs RLC buffer size

In this graph we observe that the RLC buffer size is inversely proportional to the delay. With the delay appearing to flatten as we approach the higher buffer sizes. This graph has been generated for the elephant flow having the lambda value of 30,000.

## Conclusion:

From the graphs and the report we conclude that on implementing the CoDel algorithm we get a better delay even though there is some packet loss and by manipulating the parameters such as threshold and interval of CoDel we can get an optimal condition where we can have good improvement in delay without great packet loss percentage.

**Future Description:**

- The current implementation can be extended to the other mode of the RLC i.e. the AM mode since our current implementation is only in the UM mode.
- A proper implementation of the CoDel algorithm can be done which involves creation of a CoDel.h,CoDel.cc and a Queue.cc, Queue.h file that involve in creation of the queue for the RLC as well as are involved in its

maintenance allows multiple algorithms to be created that can simply be extended to this Queue.

- Implementing other AQM algorithms mentioned in the Dynamic Buffer Sizing paper to compare them to give the required result.
- Implementation of a UPF to be able to demonstrate the router based implementation where the GNB is connected to the Internet through the router( connected via the UPF).
- The current buffer size manipulation program requires us to manually change the buffer size, can implement a dynamic buffer sizing algorithm to compare their performance with the respective AQM algorithms.

**Reference:**


- **Base Paper** : Nichols, K.; Jacobson, V.; McGregor, A.; Iyengar, J. Controlled Delay Active Queue Management. RFC 8289. January 2018. Available online: https://tools.ietf.org/html/rfc8289
- Mikel Irazabal, Elena Lopez-Aguilera, Ilker Demirkol, Navid Nikaein, "Dynamic Buffer Sizing and Pacing as Enablers of 5G Low-Latency Services", IEEE Transactions on Mobile Computing, vol.21, no.3, pp.926-939, 2022.
- https://www.ericsson.com/en/reports-and-papers/ericsson-technology-review/articles/virtual-aqm-for-mobile-networks
- Bufferbloat Demo at IETF 86 - YouTube
- http://ce.sc.edu/cyberinfra/workshops/Material/NTP/Lab%2018.pdf


**Team Contribution:**

**Medha:** When through the research papers to find the base paper for project implementation. Involved in base case scenario, and in verifying RLC logs. Tried implementing CoDel on the router connecting from the gNB to the internet. Involved in writing the report and the presentation making.

**Prathyush:** Wrote the code for the codel implementation,was a part of writing the code for the base case scenario, generated the graphs and output files for the report and was involved in the report writing as well as presentation making process.

**Manisha:** Was involved in the pgw based implementation for the router connection to the internet and involved in a part of the report writing.

**Anti-Plagiarism:**

PLAGIARISM STATEMENT

*I certify that this assignment/report is our own work, based on our personal study and/or research on our personal/lab equipment and that we have acknowledged all material and sources used in its preparation, whether they be books, articles, reports, lecture notes, and any other kind of document, electronic or personal communication. We also certify that this assignment/report has not previously been submitted for assessment in any other course, except where specific permission has been granted from all course instructors involved, or at any other time in this course, and that we have not copied in part or whole or otherwise plagiarized the work of other students and/or persons. We pledge to uphold the principles of honesty and responsibility at CSE@IITH. In addition, We understand our responsibility to report honor violations by other students if we become aware of it.*

**Names and Roll Nos:** Sree Prathyush Chinta (CS19BTECH11043),
Medha Rachel Panna (CS22MTECH11003),
Manisha Mahapatra (CS22MTECH14009),
SOHAN D P (CS19BTECH11028)

**Date:** 7-12-2022

**Signatures:** Sree Prathyush Chinta, Medha Rachel Panna, Manisha Mahapatra, SOHAN D P