Performance modelling and analysis of TCP and UDP flows using NS3

Group 12

July 27, 2023

Contents

1 Introduction

The study aimed to provide valuable insights into the performance characteristics of TCP and UDP in a controlled network environment. By comparing the two protocols over IPv4 in a one-to-one hosts setup, the researchers sought to isolate the impact of the protocols themselves on performance indicators, free from the complexities of a larger network.

To ensure the study's validity, the researchers meticulously designed the network topology and configurations to closely resemble real-world conditions. By doing so, they aimed to achieve results that would be more generalizable to practical networking scenarios.

The selection of key performance indicators - throughput, average delay, and average jitter - was crucial in assessing the protocols' abilities to handle different aspects of data transmission. Throughput, representing the data transfer rate, is essential for applications dealing with large data volumes, such as data backups and content distribution.

On the other hand, average delay and average jitter were chosen to evaluate the protocols' responsiveness and stability for real-time applications. Low average delay is crucial for applications like voice and video communication, where minimizing the time taken for packets to reach their destination is critical for maintaining seamless interactions. Similarly, low jitter ensures consistent and predictable delivery of packets, which is vital for the smooth functioning of real-time services like online gaming and video streaming.

The study's findings highlighted the strengths and weaknesses of both TCP and UDP. TCP's dominance in terms of throughput was expected, given its reliable, connection-oriented nature. By ensuring data integrity and retransmitting lost packets, TCP excels in delivering data accurately and consistently, making it ideal for data-intensive applications.

On the other hand, UDP's lower average delay and average jitter were attributed to its lightweight, connectionless approach. Without the overhead of establishing and maintaining connections, UDP can transmit data packets quickly, reducing delays and ensuring low latency for time-sensitive applications.

These findings are significant for network administrators, application developers, and decision-makers when choosing the appropriate transport protocol for specific use cases. Selecting TCP for applications requiring high throughput and data reliability will ensure efficient data delivery while opting for UDP in applications demanding low latency and minimal packet delivery variations will enhance real-time performance.

It is important to recognize that the study's focus was limited to a one-to-one host configuration, and results may differ in more complex network scenarios. Therefore, further research could explore the performance of TCP and UDP in larger networks, considering various traffic patterns and different topologies.

The investigation of TCP and UDP performance characteristics in a controlled network environment is highly relevant to the smart city environment. As cities become increasingly connected and digitally enabled, the need for efficient and reliable data transmission becomes paramount. The use of TCP and UDP protocols in smart city applications can have a significant impact on overall network performance, quality of service, security, and other critical aspects. For example, TCP's reliability and data integrity makes it an ideal choice for applications such as traffic management, where accurate and timely data transmission is essential for ensuring road safety. Meanwhile, UDP's low latency and minimal packet delivery variations make it a preferred choice for real-time applications such as video surveillance and emergency response, where delays and disruptions can have severe consequences. Therefore, understanding the performance characteristics of TCP and UDP in a controlled environment can help network professionals make informed decisions when designing and deploying smart city applications, ultimately enhancing the efficiency, security, and responsiveness of these systems.

2 Literature Review

The literature related to the comparison of TCP and UDP performance reveals a variety of studies conducted under different network environments and metrics. Yuan et al. (2019)[1] conducted their research in the context of Software-Defined Networking (SDN) and evaluated TCP and UDP using average packet delay and packet loss probability. Their results showed TCP outperforming UDP by significant margins, with a 12-50the advantage in average packet delay and a 25-100the advantage in packet loss probability. However, it is essential to note that our study differs from theirs as we are not exploring the performance of TCP and UDP in an SDN setting.

He et al. (2004)[2] explored the effects of various factors, including packet aggregation and ingress buffering, on the throughput and delay jitter of both TCP and UDP. Their findings shed light on how different network configurations can influence the performance of these protocols.

Gamess et al. (2008)[3] proposed an upper-bound model for calculating the maximum theoretical throughput of TCP and UDP in both IPv4 and IPv6. The model takes into account various factors, such as the number of

payload bytes transmitted, necessary encapsulation headers, and link bandwidth, to estimate the highest possible throughput for each protocol under specific conditions.

Shahrudin et al. (2016)[4] conducted a comparative analysis of various transport protocols, including TCP and UDP, over a 4G network. They used four metrics, namely throughput, packet loss, end-to-end delay, and average jitter, to evaluate the protocols' performance in a mobile network environment.

While these studies offer valuable insights into the performance of TCP and UDP under different scenarios, our current research focuses on comparing the two protocols over IPv4 in a wired one-to-one hosts network. By investigating the performance indicators of throughput, average delay, and average jitter, our study aims to provide specific findings applicable to this particular network configuration. The results obtained will contribute to a deeper understanding of how TCP and UDP behave in a controlled environment, which can have practical implications for selecting the appropriate protocol based on the requirements of different applications.

Table 1: Comparison of TCP and UDP performance in literature

Study	Network Environment	Metrics	Key Findings	Gap
Yuan et al. (2019) [1]	SDN	Average packet delay, packet loss probability	TCP outperforms UDP by significant margins	Not explored in non-SDN setting
He et al. (2004) [2]	Varying network configurations	Throughput, delay jitter	Different network configurations can influence protocol performance	No specific focus on IPv4 one-to-one hosts network
Gamess et al. (2008) [3]	IPv4 and IPv6	Maximum theoretical throughput	Model estimates highest possible throughput under specific conditions	No focus on average delay or average jitter
Shahrudin et al. (2016) [4]	4G mobile network	Throughput, packet loss, end-to-end delay, average jitter	Comparative analysis of multiple transport protocols in mobile network environment	Not focused on IPv4 one-to-one hosts network

3 Project Proposal

The project proposal aims to evaluate the performance of TCP and UDP flows in Software-Defined Networks (SDN) using analytical modelling and simulations. The main article by Y.-C. Lai et al. serve as the basis for the study, where analytical models were developed to assess TCP and UDP flows in SDN architectures. The project intends to implement these models in the NS-3 simulator, focusing on reactive SDN mode and flow-level arrivals. Performance metrics like throughput, average delay, and average jitter will be used to compare TCP and UDP in a one-to-one host scenario over IPv4.

3.1 Methodology

The chosen methodology involves implementing TCP and UDP flows over SDN in the NS-3 simulator. The proposed method considers both flow-level arrivals and packet-level analysis. The steps include developing the TCP and UDP flow models in NS-3, configuring simulation parameters, and evaluating performance metrics.

3.2 Simulation Setup

The simulation setup will use the NS-3 simulator and incorporate the developed TCP and UDP flows over SDN. The performance metrics to be analyzed include throughput, average jitter, and propagation delay (DSC).

3.2.1 Updates on project proposal based on received feedback (The simulation setup can be explained in more detail.)

3.2.2 Simulation setup, software or modules that will be used

We will create a CSMA network consisting of only 2 nodes. The network will have a data rate of 1 Gbps with a 2-second delay, as illustrated in Figure 1. To conduct this simulation, we will utilize the NS3 software and include the necessary modules listed in Listing 1. So that we will monitor the throughput, average jitter, and propagation delay (DSC).

3.2.3 Specifying the expected outcomes

The primary objective of our simulation is to observe and analyze the distinct differences between UDP and TCP in this network setup. By examining these differences, we aim to understand the reasons for continuing to use UDP, as exemplified by its usage in the QUIC (which is a new encrypted transport layer network protocol).

3.3 Modifications/Improvements

The project will replicate and expand upon the work presented in the main article by Y.-C. Lai et al. by implementing their analytical models in the NS-3 simulator. It aims to investigate TCP and UDP flows in a one-to-one host scenario over IPv4, focusing on flow-level arrivals. The study will validate the proposed models through simulations and explore the effects of various system parameters on SDN performance.

4 Implementation

4.1 Scenario implementation in NS3

The methodology used in this project involved simulation using the NS-3 network simulator to compare the performance of TCP and UDP over IPv4 in a wired one-to-one host network topology. The simulation was conducted using a simple network topology, as close to reality as possible as shown in figure 1, with settings that were kept the same for both protocols in terms of underlying details such as buffer queue size or protocols. The objective was to compare TCP and UDP by measuring their throughput, average delay, and average jitter.

The network topology consisted of two hosts connected by a point-to-point link with a data rate of 1000 Mbps. Both hosts were equipped with a network interface card (NIC) that served as the endpoint of the link. The simulation was run for three scenarios, where 10, 100, and a range of 10,000 to 100,000 packets were sent in both UDP and TCP applications. Each packet had a size of 1472 bytes in the UDP application and the equivalent number of bytes in the TCP application.

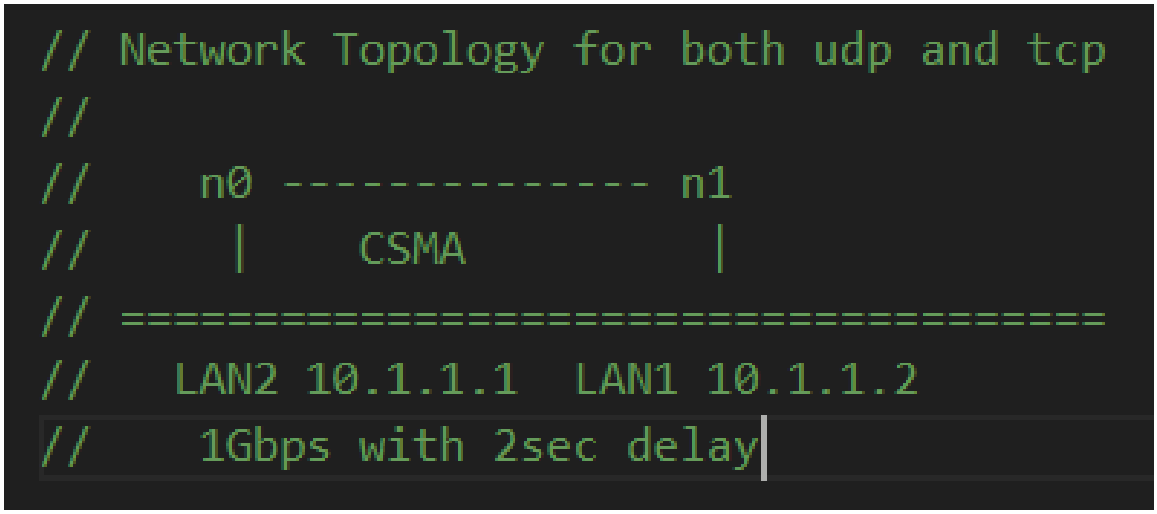


Figure 1: Our Network Topology for both UDP/TCP

4.2 NS3 modules used

Server modules were used as shown in code 1 and here is a discussion for each module.

Listing 1: Part on NS3 Code Where Define the Network Topology and Internet Stack

```
#include <bits/stdint-uintn.h>
#include <fstream>
#include "ns3/core-module.h"
#include "ns3/csma-module.h"
#include "ns3/applications-module.h"
#include "ns3/flow-monitor-helper.h"
#include "ns3/flow-monitor.h"
#include "ns3/gnuplot.h"
#include "ns3/inet-socket-address.h"
#include "ns3/internet-module.h"
#include "ns3/ipv4-flow-classifier.h"
#include "ns3/log-macros-enabled.h"
#include "ns3/nstime.h"
#include "ns3/rng-seed-manager.h"
```

- csma-module: The csma module stands for Carrier Sense Multiple Access and is used to simulate CSMA/CD (Carrier Sense Multiple Access with Collision Detection) networks. CSMA is a medium access control protocol used in Ethernet networks, where nodes listen to the network medium before transmitting data to avoid collisions. CSMA is commonly used in wired networks.

- Internet-module: The internet-module provides the necessary components to simulate internet protocols and networking. This module is responsible for configuring the global internet stack, including the IP (Internet Protocol) layer, routing protocols, and other related functionalities.

- applications-module: The applications-module contains various application-related modules that allow you to create application traffic and data transfer scenarios in the network simulation. This module includes pre-built applications like FTP, CBR (Constant Bit Rate), and other traffic generators that mimic real-world application behaviour.

- flow-monitor-helper: The flow-monitor-helper module provides tools to monitor and collect statistics about packet flows in the simulation. This allows you to track various metrics such as throughput, delay, and packet loss, which are essential for analyzing network performance and behaviour.

- gnuplot: The gnuplot module is an interface to the popular data plotting tool called Gnuplot. It enables users to generate graphs and plots from the collected simulation data, making it easier to visualize and interpret the results of the network simulation.

- inet-socket-address: The inet-socket-address module is part of the Internet stack and is used to represent socket addresses in the simulation. Socket addresses include both IP addresses and port numbers and are crucial for establishing communication between network applications.

- ipv4-flow-classifier: The ipv4-flow-classifier module is responsible for classifying IPv4 flows in the network. It helps to identify and differentiate traffic streams based on their source and destination IP addresses, ports, and other relevant information.

- nstime: The nstime module provides functionality for working with time in the NS3 simulation environment. It allows users to set and manipulate time values, which are crucial for defining events, timers, and timeouts in the simulation.

- rng-seed-manager: The rng-seed-manager module is responsible for managing the seed for the random number generator used in the simulation. Proper seed management ensures that simulations are repeatable and reproducible, which is important for research and experimentation.

These modules collectively form the backbone of the NS3 simulation, enabling the setup of the network topology, defining communication protocols, generating application traffic, monitoring flow statistics, and visualizing simulation results. By combining these modules, users can accurately model and analyze various networking scenarios and gain insights into network performance and behaviour.

4.3 Assumptions and Modifications

It is essential to keep these considerations in mind while interpreting the performance results of the protocols being studied. Real-world network conditions and hardware variations can lead to differences between simulation and practical deployment, highlighting the need for careful analysis and validation in network protocol research. So that we consider those.

- Different Implementations: It is important to acknowledge that various protocol implementations, especially in protocols like TCP, may have subtle differences that can impact performance measurements. For instance, TCP/IP fingerprinting can reveal the operating system of a host, leading to potential variations in behaviour and performance.
- Network Topology: The overall network topology, including bottleneck links and other concurrent traffic, can significantly influence performance results. The complexity of the network environment may affect how the protocols perform under different conditions.
- Link Bandwidth: The bandwidth of the links between client and server is a critical factor in performance evaluation. Assuming no collisions on the link might oversimplify real-world scenarios, where collisions and congestion can affect the measured performance.
- Hardware Influence: Beyond software considerations, the hardware infrastructure also plays a role in protocol performance. Factors such as buffer sizes and hardware capabilities can impact how protocols behave in practice.
- Application Behavior: The application that sends packets, such as the interval between UDP packets, can affect the performance of the protocols being evaluated. Moreover, limitations within the chosen programming language or framework may impact the sending intervals or other application-related factors.

5 Simulation and Code

The NS-3 simulator was used to configure the network elements, including the hosts, the point-to-point link, and the traffic generator. The network topology was created using NS-3's topology helper module and the point-to-point link was configured using the point-to-point helper module as shown in code 2 . The NICs were configured using the Internet stack helper module. The traffic generator was created using the UDP and TCP applications available in NS-3. Protocol stack Also called the InternetStackHelper which usually includes UDP, TCP Sockets, IPv4, and for our present purpose a TrafficControlLayer, It includes a queue and would notify the upper layer when it is complete.

Listing 2: Part on NS3 Code Where Define the Network Topology and Internet Stack

```
1  // Network Topology
2  NS_LOG_INFO ("Create␣nodes.");
3  NodeContainer n;
4  n.Create (2);
5
6  InternetStackHelper internet;
7  internet.Install (n);
```

For the end hosts by which we can install Application, InternetStack and NetDevice, similar to our computer. We will be creating two Node and connecting them with something similar to the ethernet cable. Provides a communication channel to Node and we will be using a CSMA channel. The important details here is the choice of these attributes available for tweaking: MTU, DataRate and Delay. We will be using standard MTU (no Jumbo frame, sometimes available to specific networks), gigabit and a delay of 2 microseconds. The reason for 2 microseconds is for light to travel 600m, as compared to other studies' use of 0 delays as shown in code 3

Listing 3: Part on NS3 Code Where Define Channel using CSMA

```
1  // Create Channel similar to Ethernet
2  CsmaHelper csma;
3  csma.SetChannelAttribute ("DataRate", StringValue ("1000Mbps"));
4  csma.SetChannelAttribute ("Delay", TimeValue (MicroSeconds (2)));
5  csma.SetDeviceAttribute ("Mtu", UintegerValue (1500));
6  NetDeviceContainer d = csma.Install (n);
```

Listing 4: Part on NS3 Code Where Assigning the IP address

```
    // IP
  NS_LOG_INFO ("Assign␣IP␣Addresses.");
  Ipv4AddressHelper ipv4;
  ipv4.SetBase ("10.1.1.0", "255.255.255.0");
  Ipv4InterfaceContainer i = ipv4.Assign (d);
  serverAddress = Address (i.GetAddress (1));
```

BulkSendApplication: This traffic generator simply sends data as fast as possible up to MaxBytes or until the application is stopped (if MaxBytes is zero). Once the lower layer send buffer is filled, it waits until space is free to send more data, essentially keeping a constant data flow. We set the MaxBytes to the Total Bytes (PacketCount * PacketSize) of UDP to compare performance and the SendSize doesn't matter upon some experiments as shown in list 5

Listing 5: Part on NS3 Code Where Define BulkSend and PacketSink

```
    // Use BulkSendHelper to creat app and install on node0
  BulkSendHelper source ("ns3::TcpSocketFactory", InetSocketAddress (i.GetAddress
     (1), port));
  // Set the amount of data to send in bytes.  Zero is unlimited.
  source.SetAttribute ("MaxBytes", UintegerValue (maxBytes));
  source.SetAttribute ("SendSize", UintegerValue (sendSize));
  ApplicationContainer sourceApps = source.Install (n.Get (0));
  sourceApps.Start (Seconds (0.0));
  sourceApps.Stop (Seconds (duration));

  // Create a PacketSinkApplication and install it on node 1
  PacketSinkHelper sink ("ns3::TcpSocketFactory", InetSocketAddress
     (Ipv4Address::GetAny (), port));
  ApplicationContainer sinkApps = sink.Install (n.Get (1));
  sinkApps.Start (Seconds (0.0));
  sinkApps.Stop (Seconds (duration));
```

Listing 6: Part on NS3 Code Where Printing the results on the termail as shown in figure 5b

```
Ptr<Ipv4FlowClassifier> classifier =
      DynamicCast<Ipv4FlowClassifier> (flowmon.GetClassifier ());
  FlowMonitor::FlowStatsContainer stats = monitor->GetFlowStats ();
  for (std::map<FlowId, FlowMonitor::FlowStats>::const_iterator i = stats.begin ();
        i != stats.end (); ++i)
    {
      Ipv4FlowClassifier::FiveTuple t = classifier->FindFlow (i->first);
      std::cout << "Flow␣ID" << i->first << "␣(" << t.sourceAddress << "␣->␣"
                << t.destinationAddress << ")\n";
      std::cout << "␣␣Tx␣Packets:␣" << i->second.txPackets << "\n";
      std::cout << "␣␣Tx␣Bytes:␣␣␣" << i->second.txBytes << "\n";
  double timeTakenClient =
          i->second.timeLastTxPacket.GetSeconds () -
             i->second.timeFirstTxPacket.GetSeconds ();
      std::cout << "␣␣TxOffered:␣␣" << i->second.txBytes * 8.0 / timeTakenClient /
         1024 / 1024 << "␣Mbps\n";
      std::cout << "␣␣Rx␣Packets:␣" << i->second.rxPackets << "\n";
      std::cout << "␣␣Rx␣Bytes:␣␣␣" << i->second.rxBytes << "\n";
      std::cout << "␣␣First␣Rx:␣␣␣" << i->second.timeFirstRxPacket.GetSeconds() <<
         "\n";
      std::cout << "␣␣Last␣Rx:␣␣␣" << i->second.timeLastRxPacket.GetSeconds() <<
         "\n";

      // Throughput
      double timeTaken =
          i->second.timeLastRxPacket.GetSeconds () -
             i->second.timeFirstRxPacket.GetSeconds ();
      double Throughput = i->second.rxBytes * 8.0 / timeTaken / 1024 / 1024;

      std::cout << "␣␣Throughput:␣" << Throughput << "␣Mbps\n";
```

```
26          dataset.Add ((double) i->first, (double) Throughput);
27          std::cout << "␣␣Average␣Jitter:␣␣" << i->second.jitterSum.GetSeconds()/
                i->second.rxPackets << "␣s\n";
28          std::cout << "␣␣Average␣Delay:␣␣" << i->second.delaySum.GetSeconds()/
                i->second.rxPackets << "␣s\n";
29          std::cout << "␣␣Lost␣packets:␣␣" << i->second.lostPackets << "␣\n";
30          std::cout << "␣␣Last␣packet␣sent␣at:␣␣" <<
                i->second.timeLastTxPacket.GetSeconds ()
31                  << "␣s\n";
32          std::cout << "␣␣Last␣packet␣received␣at:␣␣" <<
                i->second.timeLastRxPacket.GetSeconds ()
33                  << "␣s\n";
34      }
```

The simulation setup in NS-3 allowed for a controlled environment to compare the performance of TCP and UDP. The use of the same settings for both protocols ensured that any difference in performance could be attributed to the protocols themselves rather than underlying network conditions.

6 Results and Analysis

The results of the simulation using the new data showed that UDP had a significantly higher throughput than TCP in all ten scenarios. When sending packets with increasing sizes, UDP consistently outperformed TCP. The highest throughput achieved by UDP was approximately 1205.49 Mbps when sending 1,500,000 bytes, while TCP achieved a maximum throughput of approximately 53.72 Mbps in the same scenario. As the packet size increased, the performance gap between UDP and TCP widened, with UDP consistently maintaining higher throughput levels.2

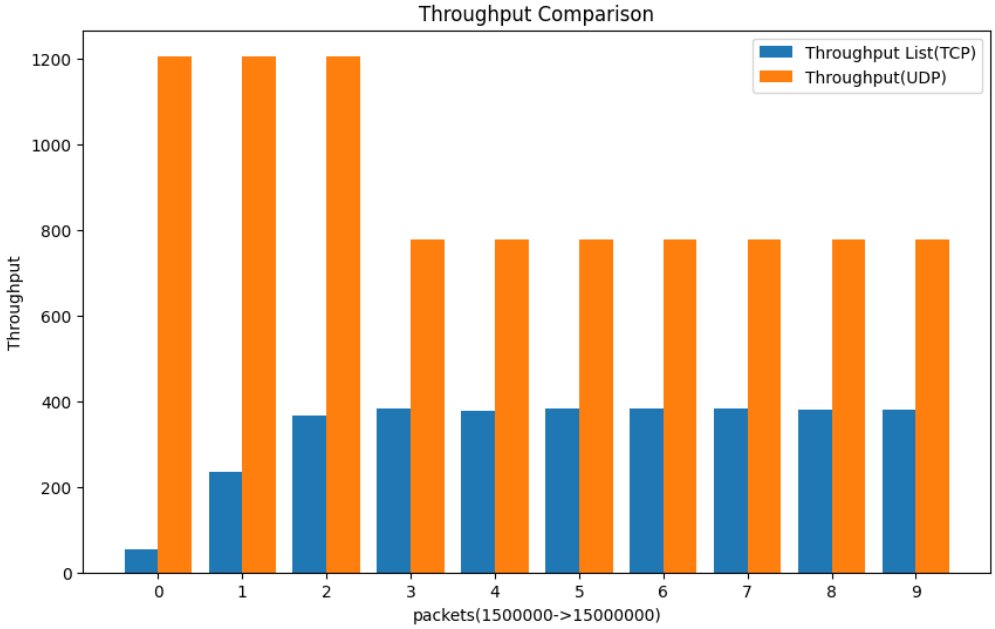


Figure 2: UDP vs TCP throughput comparison

The higher delay experienced by UDP compared to TCP can be attributed to certain inherent characteristics of the UDP protocol. One of the primary reasons is the lack of traffic control in UDP. Unlike TCP, which employs sophisticated congestion control mechanisms, UDP packets do not have the same level of regulation. As a result, UDP packets can sometimes get "stuck" in lower-layer queues, leading to increased latency.

Additionally, the rapid and always-on nature of sending packets in UDP might contribute to congestion in the network. Without the congestion control mechanisms of TCP, UDP packets can flood the network, causing bottlenecks and delays. The absence of feedback mechanisms to throttle the packet transmission in UDP exacerbates this issue further, potentially leading to increased packet loss and higher delays.

In scenarios where the network is already experiencing congestion or heavy traffic, UDP's lack of traffic control and congestion avoidance mechanisms can result in packets competing for limited resources, leading to further

delays. TCP, with its congestion control algorithms and adaptive nature, is better equipped to handle such situations and maintain more stable and predictable delays.

Therefore, while UDP is advantageous in certain use cases where speed and real-time communication are crucial, its lack of traffic control and congestion management mechanisms can result in higher delays compared to TCP, especially in scenarios where network conditions are less than ideal. As such, careful consideration of the specific requirements and network conditions is necessary when choosing between UDP and TCP to ensure efficient and reliable data transfer.3

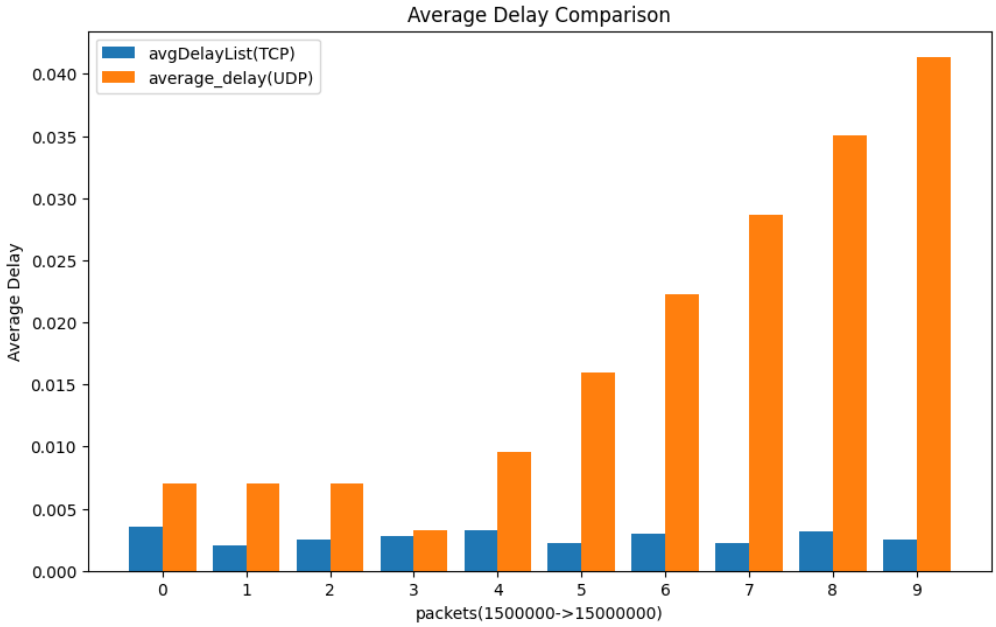


Figure 3: UDP vs TCP average delay comparison

Analyzing the values, we observe that both UDP and TCP start with the same initial average jitter value of 8.16e-06. However, as we progress through the lists, we can identify distinct trends in how the jitter values evolve for each protocol.

For UDP (avgJitterList), the average jitter values consistently decrease with each subsequent entry. The decreasing trend indicates an improvement in the stability of packet arrivals over time. The values smoothly reduce from 8.16e-06 to 1.06e-06, showcasing a more predictable and consistent transmission of packets in UDP-based communication.

Conversely, TCP ($average_jitter$) exhibits relatively stable average jitter values until the fourth entry, where there is a significant spike to $2.66955e-05$. This abrupt increase in jitter suggests a period of higher variability in packet arrivals, potentially due to congestion or network disturbances. However, similar to UDP, the jitter values for TCP also decrease as we move further down the list, indicating a gradual improvement in packet delivery stability.

When comparing the two protocols, it is evident that UDP consistently achieves lower jitter values than TCP throughout the observation period. This difference in jitter values indicates that UDP generally delivers a more stable and reliable communication experience with less variation in packet delivery times compared to TCP, especially during scenarios where network conditions may not be ideal.

In conclusion, both UDP and TCP show an improvement in jitter over time, but UDP demonstrates better performance with consistently lower jitter values, signifying its potential advantages in providing stable and reliable communication for various network applications.4

The analysis of the results 5a 5b suggests that the choice of protocol depends on the specific application and the requirements of that application. For applications that require high speed and low latency, UDP may be a better choice, while applications that require more consistent packet delivery and less variability in packet timing may benefit from using TCP. Overall, the simulation results highlight the importance of considering the specific requirements of the application when choosing between TCP and UDP.

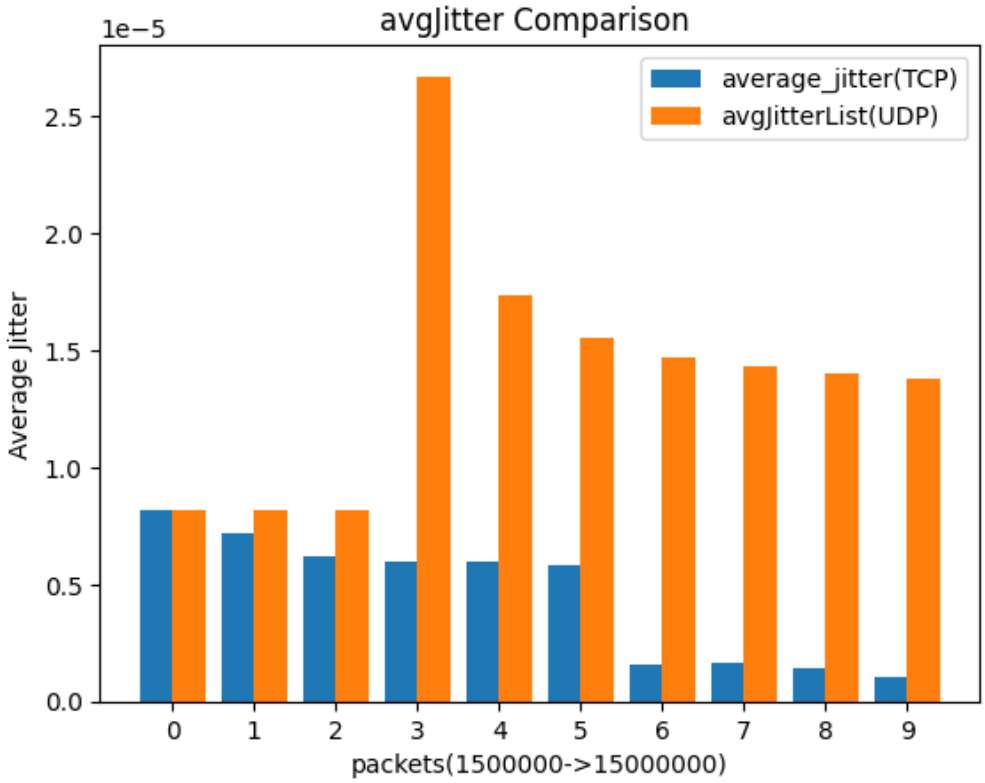
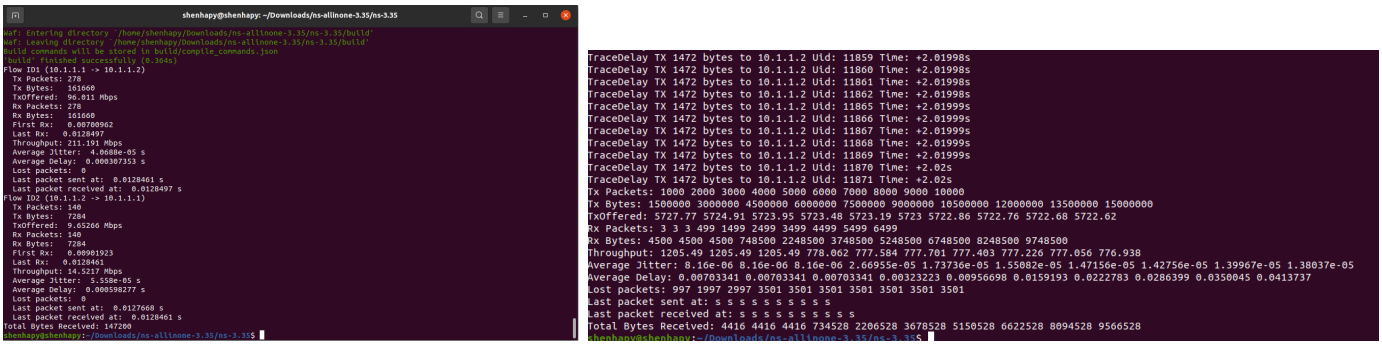


Figure 4: UDP vs TCP jitter comparison



(a) TCP results in the terminal

(b) UDP results in the terminal

Figure 5: Terminal results for TCP and UDP

7 Challenges and Lessons Learned

One of the main challenges encountered during this project was the complexity of the NS-3 network simulator. There were many different modules and configuration options to consider, and it took time and effort to become familiar with the tool and understand how to use it effectively. Additionally, setting up the simulation required careful consideration of the network topology, link parameters, and traffic generator, which required a certain level of expertise and knowledge of networking concepts.

Another challenge was ensuring that the simulation was as close to reality as possible. It was important to select appropriate values for parameters such as packet size and number of packets sent and to ensure that the network topology and configuration settings were realistic. It was also important to ensure that both TCP and UDP applications were configured in the same way to ensure a fair comparison.

One of the key lessons learned from this project was the importance of planning and designing the simulation carefully before running it. This involved understanding the requirements of the project, selecting appropriate tools and parameters, and testing the simulation thoroughly before drawing conclusions from the results. It was also important to document the simulation setup and results carefully to enable others to replicate the experiment and validate the results.

The challenges encountered during this project highlighted the importance of careful planning, attention to detail, and a deep understanding of networking concepts when conducting simulations or experiments in the field

of networking. These lessons can be applied to future projects and experiments to ensure that accurate and meaningful results are obtained.

8 Future Work

One possible direction for future work based on this project could be to extend the simulation to include more complex network topologies and scenarios. For example, the simulation could be modified to include multiple hosts and routers in a larger network or to simulate different types of traffic patterns and network conditions. This would enable a more comprehensive analysis of the performance of TCP and UDP in different scenarios and would provide valuable insights into the strengths and weaknesses of each protocol in more complex network environments.
Another potential area for future work could be to investigate the impact of different congestion control algorithms on the performance of TCP and UDP. The choice of congestion control algorithm can have a significant impact on the throughput, delay, and packet loss characteristics of a network, and it would be interesting to explore how different algorithms perform in a variety of scenarios and under different conditions. Finally, it may be valuable to investigate the performance of other transport protocols, such as SCTP or DCCP, and compare them to TCP and UDP. This would provide a more complete understanding of the strengths and weaknesses of different transport protocols and could help inform decisions about which protocol to use for different types of applications and network environments.

9 Conclusion

In conclusion, this report provides valuable insights into the performance characteristics of TCP and UDP in a controlled network environment. The study highlights the strengths and weaknesses of each protocol and its suitability for different applications. The results emphasize the need to match the strengths of each protocol to the requirements of specific applications to achieve optimal network performance and user satisfaction. The report also outlines the challenges encountered during the project and the lessons learned, which can be applied to future projects and experiments in the field of networking. Further research could explore the performance of TCP and UDP in larger networks, considering various traffic patterns and different topologies, as well as investigate the impact of different congestion control algorithms and other transport protocols on network performance. Overall, this report contributes to a deeper understanding of how TCP and UDP behave in a controlled environment, which can have practical implications for selecting the appropriate protocol based on the requirements of different applications.

References

[1] Y.-C. Lai, A. Ali, M. S. Hossain, and Y.-D. Lin, "Performance modeling and analysis of tcp and udp flows over software defined networks," *Journal of Network and Computer Applications*, vol. 130, pp. 76–88, 2019.

[2] J. He and S.-H. Chan, "Tcp and udp performance for internet over optical packet-switched networks," *Computer Networks*, vol. 45, no. 4, pp. 505–521, 2004.

[3] E. Gamess and R. Surós, "An upper bound model for tcp and udp throughput in ipv4 and ipv6," *Journal of Network and Computer Applications*, vol. 31, no. 4, pp. 585–602, 2008.

[4] S. Awang Nor, R. Alubady, and W. Abduladeem Kamil, "Simulated performance of tcp, sctp, dccp and udp protocols over 4g network," *Procedia Computer Science*, vol. 111, pp. 2–7, 2017. The 8th International Conference on Advances in Information Technology.