Measuring Packet Processing Overheads in the Linux Kernel

Author - Razi Murshed

Course - ECSE 498

Student ID - 260516333

Supervisor - Professor Muthucumaru Maheswaran

Abstract

The advent of cloud computing today is generating need for increased remote procedure calls between various computing platforms. A persistent barrier to executing these remote procedure calls is the high latency of accessing a remote computer. One of the key causes of this high latency are the protocol processing overheads for data packets that occur at the source and destination machines. These protocol processing overheads can be measured in real machines and in clouds with different levels of virtualization and on bare metal machines in order to understand the causes of these latencies and the possible improvements that can made to reduce these latencies. The goal of this project is to design and implement experiments for measuring the protocol processing overheads. The outcome from these experiments include evaluation of the protocol processing overheads for various types of networks such as wired, wireless, local, etc. in different scenarios such as bare metal versus clouds. The aforementioned results and evaluations are then to be summarized and documented to be presented for the project. This paper discusses the various strategies explored in order to design and execute these experiments and provides an analysis of the data collected. It also discusses the intricacies and implications of this project on existing technology and computing practices.

Table of Contents

List of Abbreviations

IP – Internet Protocol

OS – Operating System

OSI – Open Systems Interconnection Model

RPC – Remote Procedure Call

SCTP – Stream Transmission Control Protocol

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

Note: All times mentioned throughout the paper is in microseconds.

List of Figures

Introduction

Project Objectives

One of the main objectives of this project is to design novel experiments in order to measure the time taken by a computer in order to fully process data packets. Packets may be processed in in accordance to various protocols which may lead to varying processing overheads with respect to these different protocols. Another factor that may affect the overheads of packet processing is the type of networking being used by the machines. Various networking types such as wired, wireless and local networks are likely to have different impacts on these experiments. Furthermore, the types of the machines themselves such as bare metal, cloud and virtual machines may vary the results of these experiments. In order to have valuable data and accurate analysis we are required to simulate as many realistic scenarios as possible. This can be done by repeating the measurements with different combinations of parameters mentioned above (network, environment and protocol types) and comparing and contrasting between the results obtained to analyze the effects of these parameters on the overheads for packet processing.

From these experiments, the data can be analyzed and possible sources of latency can be identified. We can then proceed to try and find improvements and isolate unnecessary steps or redundancies in the current processing architectures in order to reduce the latencies and increase speed to transmission and reception of data packets.

Applications

These results and possible improvements may further lead to several applications and exciting possibilities with regard to the future of networking technology and computing. A few of these include –

- Data Over Wi-Fi – This involves sending data from one machine to another machine in the same Wi-Fi network devoid of any IP networking protocols. The data payload is sent over Wi-Fi using a custom protocol that will have minimal header information.
- Cloud Computing Practices – The results from these experiments can be used to reevaluate current cloud computing practices and explore other avenues such as different data paths, integration techniques of cloud computing to IT systems and applications etc.
- Packet Processing Architectures – By finding the areas in which most latency occurs, we could try to change or even create new implementations of packet processing architectures in order to ensure reduction in latency and faster networking.
- Encryption/Decryption Techniques – We can explore the time spent by the current processing implementations on Encryption and Decryption of data and investigate to see whether we can increase processing speeds without compromising security or experiment with cases where security of data may not be a very important issue.
- Tethering of Mobile Devices to Personal Computers – We may be able to increase the speed of the local connection between a Mobile Device and a Personal Computer to access the latter’s functions remotely by eliminating possible latencies in both the devices’ processing architectures.
- Fog Computing - Fog computing extends the Cloud Computing paradigm to the edge of the network, thus enabling a new breed of applications and services [1]. As one of the defining characteristics of Fog computing is low latency [1], the results from these experiments may be used to accelerate and support this novel concept.

Background

This section describes the background knowledge that was acquired in order to be able to successfully understand the requirements of this project and execute it properly. This knowledge was obtained from several literature reviews, reviews of manuals for specific software and through extensive study of the Linux source code.

Networking Model and Packets

Packet networks are the most prevalent forms of networks that exist today where Packet processing in a computer is carried out by a packet processing subsystem that administrates the traversal of a data packet through a multi layered network stack [2]. Currently, the standardized computer networking model is the OSI model (see Appendix A). The packet processing subsystem manages data packets through all four layers of the TCP/IP model a model encompassing the OSI model, staring from the Physical Layer, through the Network and Transport layers all the way to the Application Layer [3]. In a packet-switched network, the host computer composes the item to be sent into a packet and each packet is routed through the network to its destination [2].

Linux Operating System

Linux is an open source UNIX-like OS that is the main object of study for this particular project. Operating Systems are responsible in all computers for packet processing and therefore play a key role in causing the latencies pertaining to this project. Linux is the most compatible OS for our particular project for reasons discussed in the design decisions section.

A simple overview of the architecture of Linux is shown in Figure 1[1] below in order to render a better illustration of the system and its relevance to this project –

[1] Image derived from *http://www.tutorialspoint.com/operating_system/os_linux.htm*

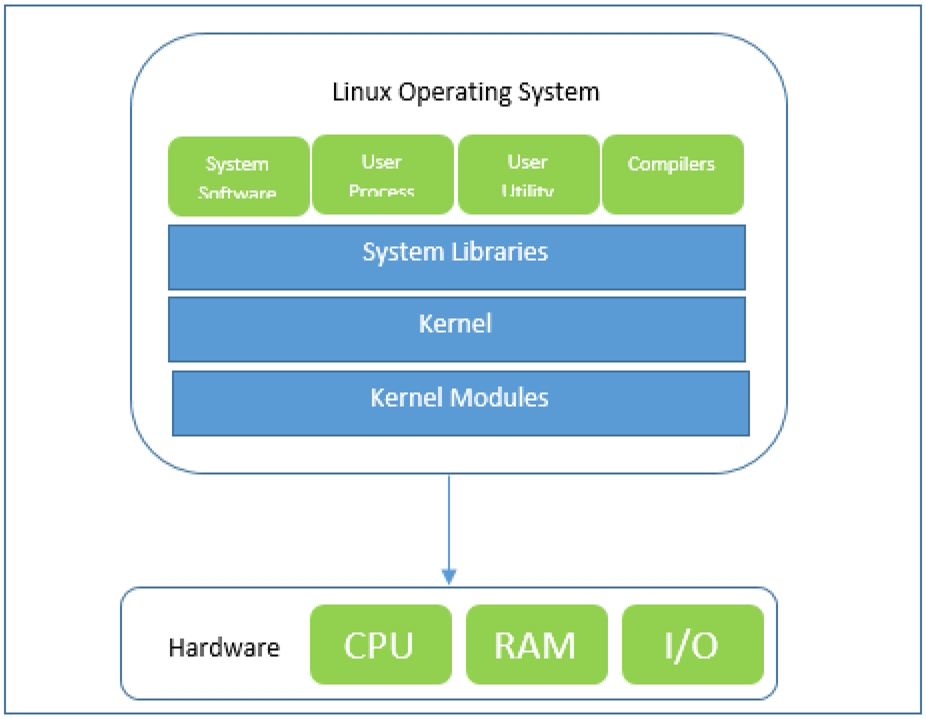


Figure 1: Components of the Linux OS

The Linux Operating System has primarily three components as described below [4] -

- **Kernel** - Kernel is the core part of Linux, responsible for all major activities carried out by the operating system. It consists of various modules and it interacts directly with the underlying hardware. The Kernel provides the required abstraction to hide low level hardware details to system or application programs.
- **System Library** – These libraries can be used by application programs or system utilities to access Kernel's features.
- **System Utility** – The programs are responsible to do specialized, individual level tasks at the user space.

For this project the kernel is very significant as this part of the OS handles the packet processing for the machine.

Networking Protocols

As mentioned in the Networking Model and Packets section previously, a packet has to traverse a multi layered network architecture in order to successfully be transmitted from a source machine and be successfully received at the destination machine. Several protocols are involved in this process and are key to determining the path a packet may take in a computing platform. The following protocols relevant (as described in the design decisions section) to this project are described below –

- IP [5] - IP is the principal communications protocol that resides in the network layer of the OSI model and is responsible for relaying datagrams across network boundaries. IP delivers packets from the source host to the destination by using on the IP addresses in the packet headers. To do this, IP defines packet structures that package the data to be delivered and determines addressing methods that are used to label the datagram with source and destination information.
- TCP – The TCP protocol complements the IP protocol and lies in the transport layer within the OSI model. TCP provides reliable, ordered, and error-checked delivery of a stream of octets between applications running on hosts communicating over an IP network [6].
- UDP [7] - UDP implements a simplistic connectionless transmission model. It has no handshaking dialogues, and thus exposes the user's program to any unreliability of the underlying network protocol. USP does not guarantee delivery, order or duplicate protection. It does however provide data integrity, and port numbers to address different functions at the source and destination of the datagram.

Project Requirements

This sections discusses in detail about the project's main requirements by addressing the main challenges posed by this project and the constraints that bind this project.

Main Challenges

To successfully accomplish the challenges posed by this project, we have to understand and overcome quite a few challenges. Finding a suitable OS was one these challenges. We need to be able to access its packet processing subsystem in order to track packet activity within the OS. It would also be beneficial to have access to the source code of the OS in order to be able to scrutinize these activities closely. Moreover, we will also be able to find the entry and exit points of a packet to the OS and its subsequent layers and therefore be able to time the processing overheads. We also needed to find a detailed path a packet takes through the OS for different protocols in order to be able to measure the time spent in each part of the kernel.

Another key challenge is to overcome the problem posed by the packets themselves. Packet loss is one of the key issues that affect our results. Even in controlled simulation environments, there is a possibility of packet loss occurring. Moreover, our experiments also have to account for garbage packets and unexpected errors that plague transmission and reception of data.

Finally, another challenge is to ensure controlled reception and transmission of data between hosts and clients in order to ensure accuracy of measurements and be able to simulate real time scenarios.

Main Constraints

One of the main constraints for this project is the fact that all types of OS are not available for close scrutiny. Most OS will not allow us to explore within its architecture. Furthermore, there are hardware constraints that arise from the machines we are using. These include but are not limited to the processor architecture, memory of the machine, the networking hardware such as Ethernet cards and cables etc. The project is also restricted by several software constraints because not all software is available or optimal for every type of OS. Moreover, the networks available at the laboratory are the only ones that we can carry out our experiments with. Finally, we would have to adhere to certain standards and protocols when generating and receiving traffic in order to simulate real scenarios.

Design and Results

In this section, we talk about the design of the experiments, the process of these designs and the results obtained so far in the project.

Design Process

The experiments were designed according to the process depicted by the following diagram –

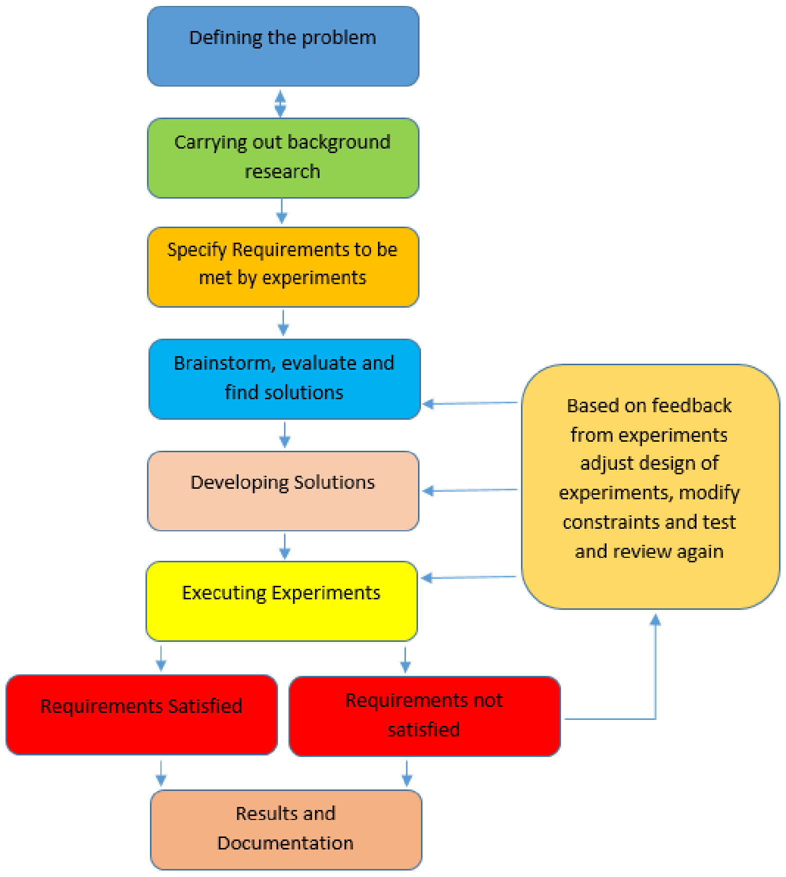

Defining the problem
Carrying out background research
Specify Requirements to be met by experiments
Brainstorm, evaluate and find solutions
Developing Solutions
Executing Experiments
Based on feedback from experiments adjust design of experiments, modify constraints and test and review again
Requirements Satisfied
Requirements not satisfied
Results and Documentation

Figure 2: Design Process followed for this project

Design Decisions

The following design decisions were made for the project so far –

Operating System -Linux Ubuntu 14.04 LTS

The OS chosen to meet the project constraints and overcome the challenges of this project was Linux, specifically the Ubuntu 14.04 LTS flavor. Linux is mostly used in server networks today which would give us valuable measurements of the time taken by a packet inside its kernel. The administrator of the network has full control of the network and access to all network components, therefore leading to increased management and reliability of the experiments conducted in such an environment. Additionally, the driver code for networking hardware is also available allowing us to study it in order to find the transition of a packet from the physical layer into the kernel of the OS. Ubuntu 14.04 LTS was chosen as the flavor as it is a tested and proven OS and the most widely used distribution today. It has a lot of community support and is user friendly. Finally, it is an open source software which allows us to carry out the project with smaller expenses.

Key Software – SystemTap

To access and monitor the activity of a data packet within the Linux kernel, we have decided to use SystemTap, a software capable of monitoring activities within the kernel. SystemTap is native to Linux and therefore most compatible for this particular project. SystemTap operates on a debuggable image of the Linux kernel which mimics the actions of the actual kernel. Additionally, no recompilation of the kernel code is required every time an instrumentation point is changed, increasing the efficiency of conducting the experiments. Furthermore, SystemTap uses C-like syntax, is command line driven and supports embedded C making it very easy to learn and user friendly. Finally, it can monitor kernel functions, variables and data structures in real time to give accurate and valuable data.

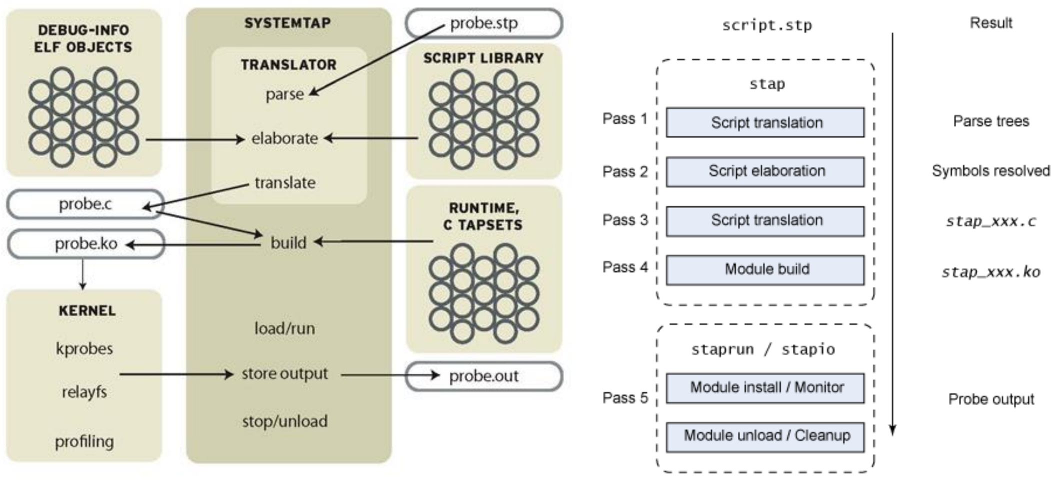


Figure 3: Data flow(left) and process(right) in SystemTap[2]

Figure 3 shows us how SystemTap works in detail. The script written gets translated to a *stap_xxx .c* file which is then loaded as a module and executed within the debuggable kernel when the *stap* or *run-stap*

[2] Image taken from http://www.redhat.com/magazine/011sep05/features/systemtap/

command is run followed by the script name. The module is unloaded and the setup is cleaned up when the script exits or when the process is exited manually (*ctrl+c*). An example of SystemTap script written for the experiments so far is included in Appendix B.

Protocols Used

The protocols to be used for our experiments were determined based on current use and relevance in today's networks. Naturally TCP/IP was chosen because of its position as the most ubiquitous network protocol today. UDP is also used in these experiments because it is increasing in popularity today in mobile applications. The advantage of choosing these two protocols are that both these types of connections can be established using the netcat tool which is built in Linux. This allows us to create a controlled environment in which the experiments can be designed. This results in more accurate measurements and also allows us to detect and account for packet loss. If time permits, we will also look into the Stream Control Transmission Protocol (SCTP) due to its improved resilience and reliability.

Analysis Tools

To analyze the data collected we decided to use Microsoft Excel and Matlab. These tools have built in functions in order to give us detailed information regarding the data and also a visual representation of the trends followed by the collected data sets. Data was written to a tab delimited text file from SystemTap output and then transferred to Excel and Matlab for analysis

Experiments

The experiments have been designed in a similar manner as depicted by the flow chart in Figure 4 –

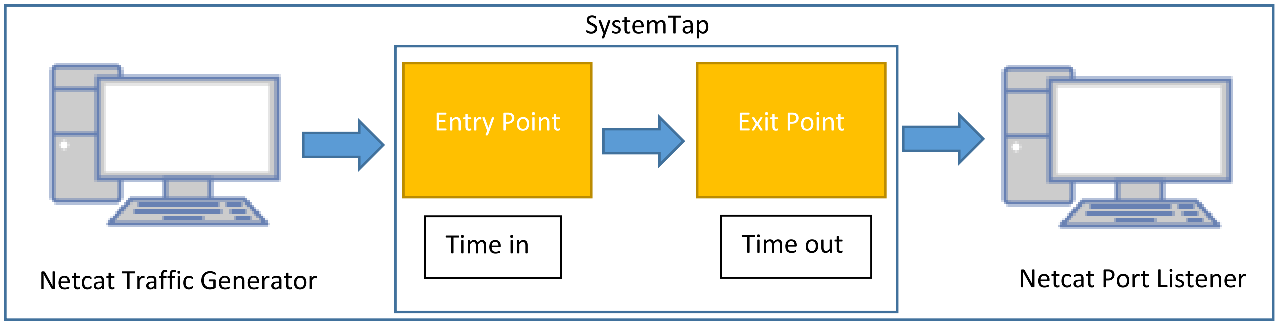


Figure 4: Experiment Design

A server generates traffic of a particular type of protocol using netcat addressed towards a client over a wired Ethernet network. The client listens on a specific port for incoming traffic. The packet sizes are all the same. 10000 such packets are generated in each variation of the experiment using a simple bash script. A SystemTap script runs on the client with instrumentation points at the entry and exit functions. As soon as a particular packet is received by a function (or a function returns) it is timestamped. After the execution of both timestamps, the time out is subtracted from the time in. The result is the time used in processing. Using this template, the following experiments have been designed –

UDP

- End to End – This experiment measures the total time spent by a UDP data packet in the Linux kernel. Here UDP packets are generated by the server. The entry occurs when the Ethernet card interrupts the kernel alerting it that a packet is ready to enter its socket buffer in the client. The function is called *netif_rx* and lies at the interface between the physical layer and the network layer [8]. The exit point is the *udp_recvmsg* function in the transport layer [8]. The results are displayed in the table in Figure 5 –

Raw Average	183.813
Raw Median	161
Raw Mode	139
Standard Deviation	155.754
Average Without Outlier	167.253
Samples	10000
Connection	Wired

Figure 5: Results of UDP End to End experiment results

- UDP Network Layer – This experiment is to determine the time spent by a packet in the network stack of the kernel. The entry point is the same as before, when the Ethernet card interrupts the kernel alerting it that a packet is ready to enter its socket buffer in the client. The function is called *netif_rx* and lies at the interface between the physical layer and the network layer [8]. The exit point is the *udp_recv* function in the interface between the network layer and transport layer [8]. The results are displayed in the table in Figure 6 –

Raw Average	27.964
Raw Median	27
Raw Mode	29
Standard Deviation	10.15
Samples	10000
Connection	Wired

Figure 6 : UDP Network Layer Experiment Results

- UDP Transport Layer – This experiment is to determine the time spent by a packet in the transport stack of the kernel. The entry point is at the *udp_rcv* function at the boundary between the network layer and the transport layer [8]. The exit point is the *udp_rcvmsg* function the indicates the packets' transition into the user space [8]. The results are displayed in the table in Figure 7 –

Raw Average	120.627
Raw Median	115
Raw Mode	115
Standard Deviation	30.082
Samples	10000
Connection	Wired

Figure 7: UDP Transport Layer Experiment Results

All of these results were then plotted in Matlab as histogram charts as follows –

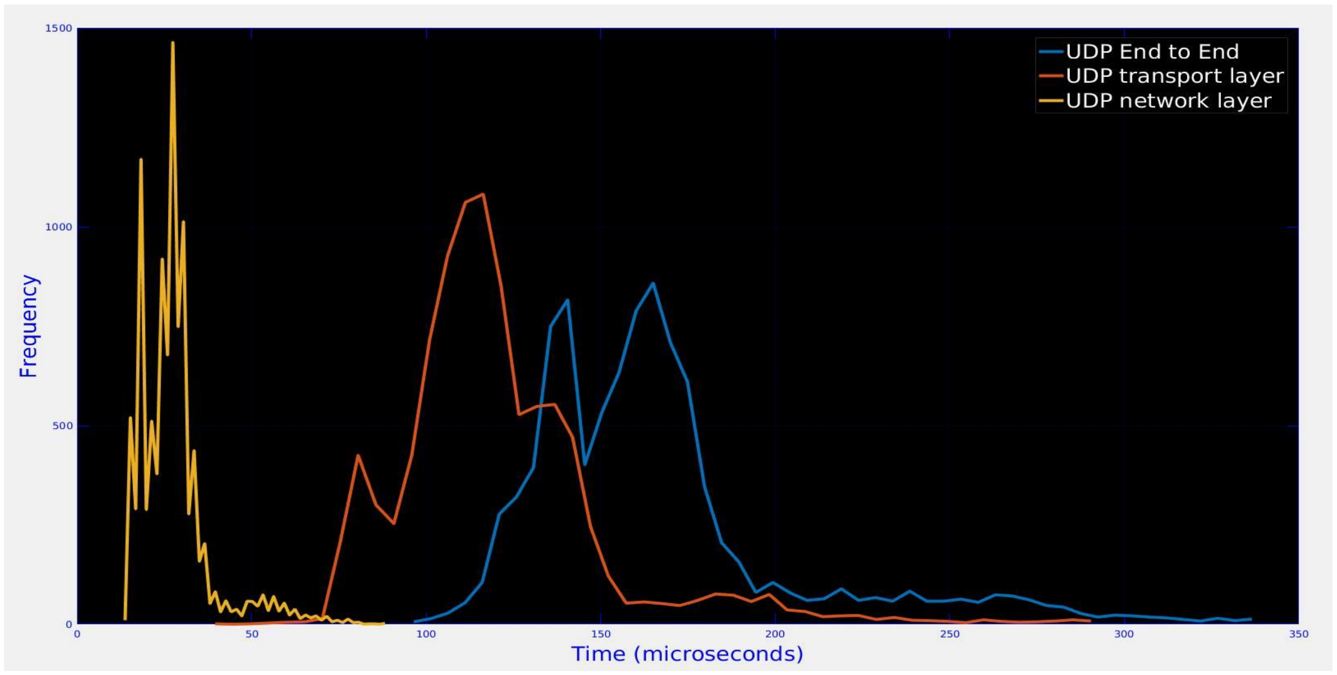


Figure 8: Results for experiments carries out using UDP

As we can see from the chart, an UDP packet spends most of its time in the transport layer a depicted by the red line as opposed to the network layer that is shown by the yellow line for a wired connection.

TCP

- End to End – This experiment measures the total time spent by a TCP data packet in the Linux kernel. Here the server generates packets to be transported over TCP. The entry occurs when the Ethernet card interrupts the kernel alerting it that a packet is ready to enter its socket buffer in the client. The function is called *netif_rx* and lies at the interface between the physical layer and the network layer [8]. The exit point is the *tcp_v4_recvmsg* function in the transport layer [8]. The results are displayed in the table in Figure 9 –

Raw Average	314.129
Raw Median	241
Raw Mode	223
Standard Deviation	314.073
Average Without Outlier	258.345
Samples	10000
Connection	Wired

Figure 9: Results for TCP end to end Experiment

- TCP Network Layer – This experiment is to determine the time spent by a TCP packet in the network stack of the kernel. The entry point when the Ethernet card interrupts the kernel alerting it that a packet is ready to enter its socket buffer in the client. The function is called *netif_rx* and lies at the interface between the physical layer and the network layer [8]. The exit point is the *tcp_recv* function in the interface between the network layer and transport layer [8]. The results are displayed in the table in Figure 10 –

Raw Average	20.258
Raw Median	17
Raw Mode	16
Standard Deviation	8.11
Samples	10000
Connection	Wired

Figure 10: TCP Network Layer Experiments results

The experiment for the TCP Transport Layer was not completed by the time of this report and therefore the results we currently have are plotted in a histogram plot in Matlab –

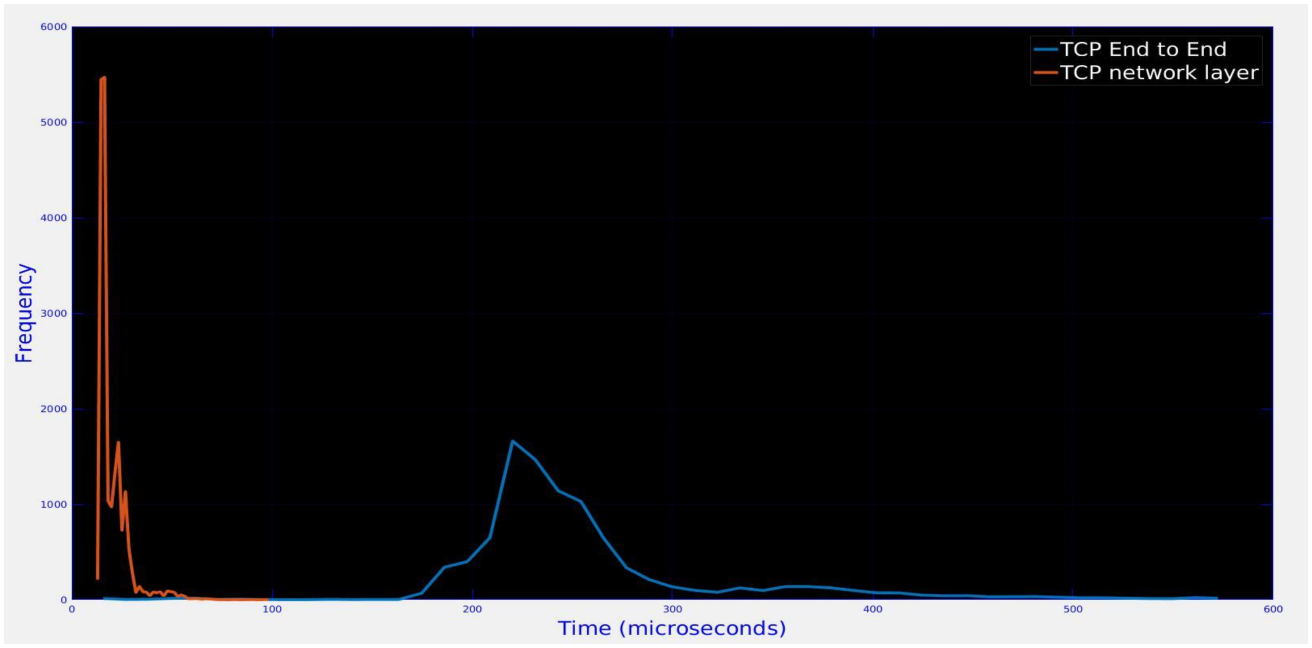


Figure 11: Results for experiments carries out using TCP

Future Plans

This section discusses the plans for the project as it progresses onto the next semester in terms of work to be done, potential design decisions and an estimated timeline –

Activities for Next Semester

In the following semester we plan to make the experiments more intricate and gather more valuable results. We would like to divide timings into smaller fragments in order to find processing time by layer end eventually by function in order to get a more detailed idea of where a packet is spending most of its time in the kernel. We would also like to try various kinds of connections and see its impacts on the results. Currently we plan to include combinations of wireless, wired and local connections. After that we plan to change the environments of the machines to cloud and virtualized environments and repeat our experiments in the new environments to see how much the environments affect the overheads. We would also like to see the effect of running actual traffic instead of simulating in a controlled environment like we are currently doing with netcat. After these activities are completed we would like to do an analysis of where in the kernel a packet is spending most of its time. If time permits, we would also like to find possible improvements to these latencies. Finally, we intend to document our results and create the poster for the presentation.

Proposed Timeline

Tasks	Start Time	End Time	Time Required
Measure Time spent within locations in the kernel	02/09/2016	15/09/2016	2 weeks
Repeat Experiments with different connection types	16/09/2016	30/09/2016	2 weeks
Use a cloud, virtualized and local test environment	01/10/2016	31/10/2016	4 weeks
Data Analysis	01/11/2016	18/11/2016	3 weeks
Documentation	15/11/2016	08/11/2016	3 weeks

Figure 12: Proposed Timeline

Impact on Society and Environment

The intended use of the results of these experiments is to reduce latency in transmission and reception and thus lead to faster communications between machines. This eventually leads to faster increased connectivity between people. The society as a whole benefits by becoming more efficient and mobile in its daily tasks. Businesses and individuals enjoy faster access to the cloud and therefore boundaries that plague collaborative work such as geography and computational resources break down. The education sector benefits immensely by allowing students to collaborate and share resources remotely. Administration becomes standardized and easier for educators. Education in developing countries especially in rural remote areas can be improved by faster communications. Moreover, this can lead to faster research and development as researchers and developers worldwide can collaborate on projects easily and quickly allowing more sharing of knowledge. Finally, the health sector may also benefit from faster communication through things such as remote critical care, reduced operational costs, e-hospitals, mass data record keeping etc. All of the above lead to increased economic benefits as well for industries and individuals due to faster processing, reduction of operational costs by outsourcing storage and computing to the cloud etc.

Conclusion

While the project is still in its nascent stages, several important tasks have been accomplished. Background studies required to execute this project have been carried out. The design environment such as the computer, OS and the connections have been set up. The requirements, challenges and constraints for this project have been analyzed and appropriate solutions have been applied. The basic experiments have been designed and carried out and their results have been analyzed. Finally, work to be carried out for the future has been determined and an approximate timeline for this work to be carried out has been established.

References

[1] F. M. R. Z. J. &. A. S. Bonomi, "Fog computing and its role in the internet of things.," *In Proceedings of the first edition of the MCC workshop on Mobile cloud computing ,* pp. 13-16, 2012.

[2] "Packets and Packet-Switching Networks," [Online]. Available: http://www.linktionary.com/p/packet-switching.html.

[3] H. Zimmermann, ""OSI reference model--The ISO model of architecture for open systems interconnection."," *IEEE Transactions on 28.4,* pp. 425-432, 1980.

[4] R. e. a. Love, " Introduction to the Linux Kernel," in *Linux kernel development second edition*, USA, Pearson Education, 2005.

[5] J. Postel, "Internet protocol," 1981.

[6] "Transmission Control Protocol," [Online]. Available: https://en.wikipedia.org/wiki/Transmission_Control_Protocol.

[7] "User Datagram Protocols," 2014.

[8] A. K. Chimata, "Path Of A Packet In The Linux Kernel Stack," University of Kansas, 2005.

Appendix A

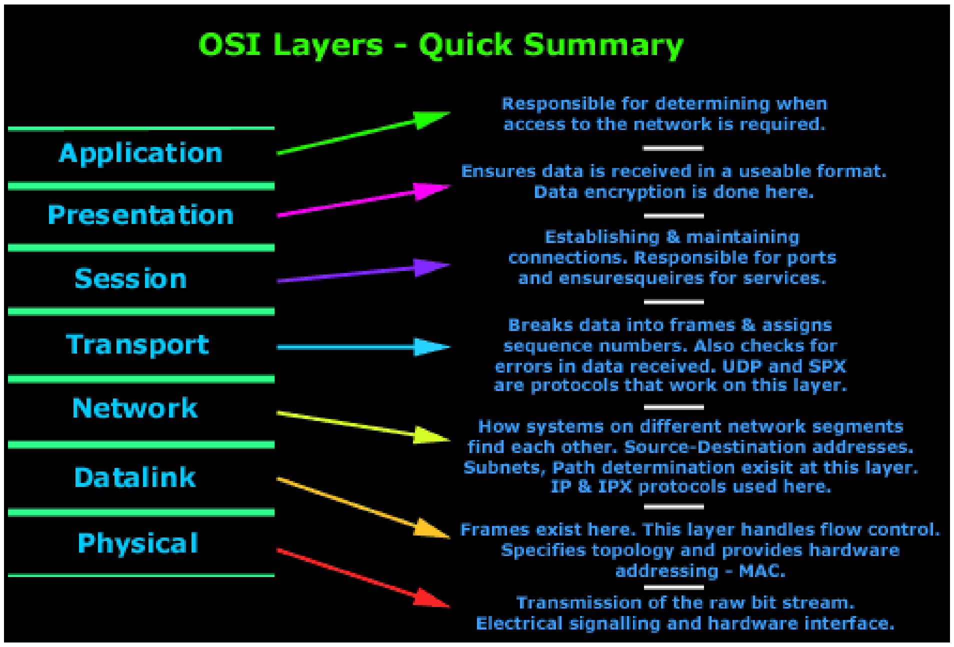

OSI Layers - Quick Summary
Application
Presentation
Session
Transport
Network
Datalink
Physical
Responsible for determining when access to the network is required.
Ensures data is received in a useable format. Data encryption is done here.
Establishing & maintaining connections. Responsible for ports and ensuresqueires for services.
Breaks data into frames & assigns sequence numbers. Also checks for errors in data received. UDP and SPX are protocols that work on this layer.
How systems on different network segments find each other. Source-Destination addresses. Subnets, Path determination exisit at this layer. IP & IPX protocols used here.
Frames exist here. This layer handles flow control. Specifies topology and provides hardware addressing - MAC.
Transmission of the raw bit stream. Electrical signalling and hardware interface.

Figure 13 : Summary of OSI Layers[3]

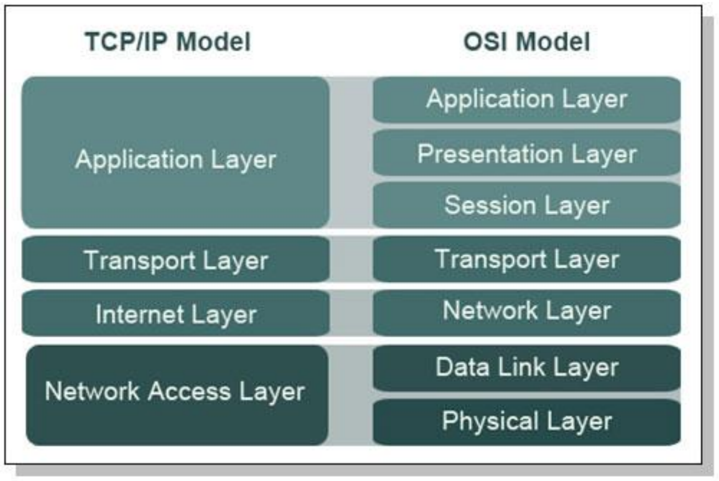
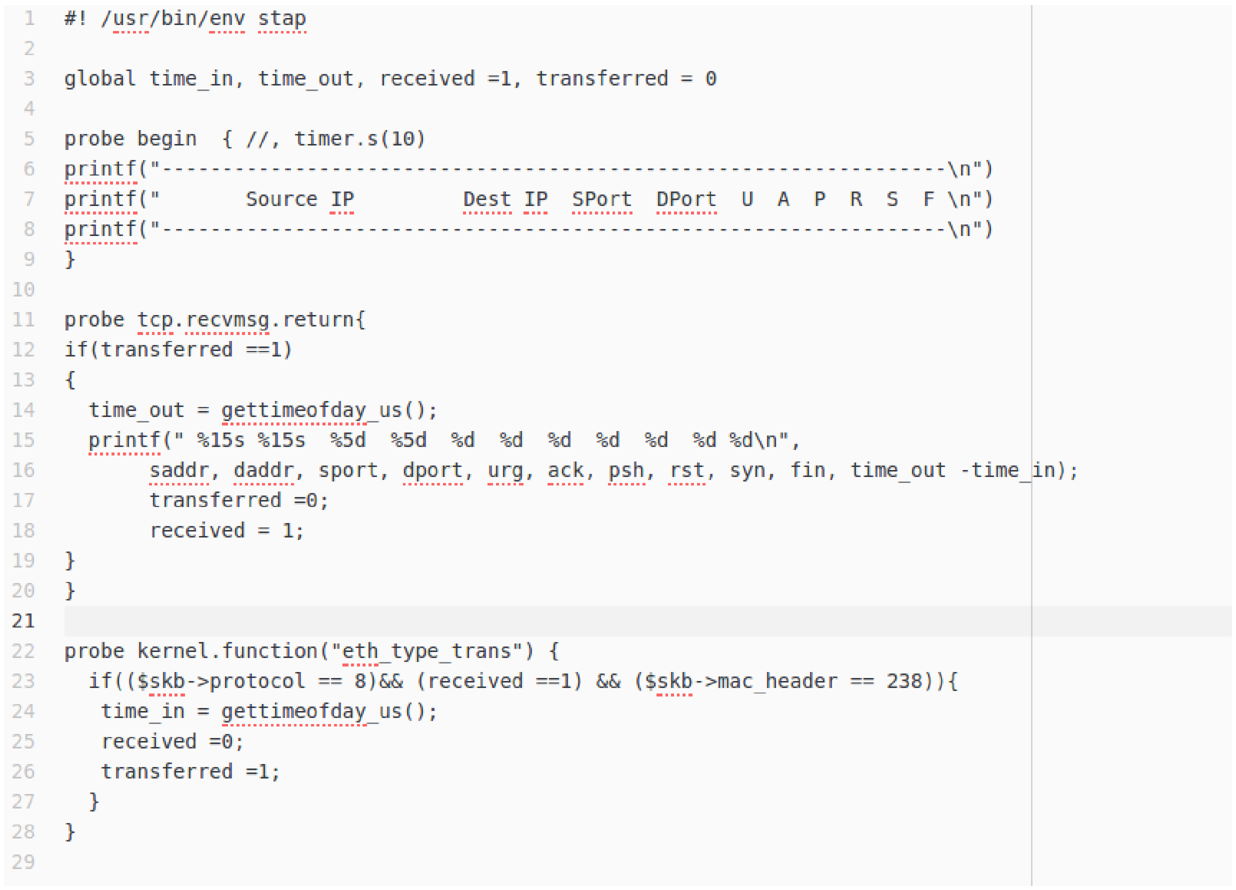


Figure 14: Relationship between TCP/IP and OSI models[4]

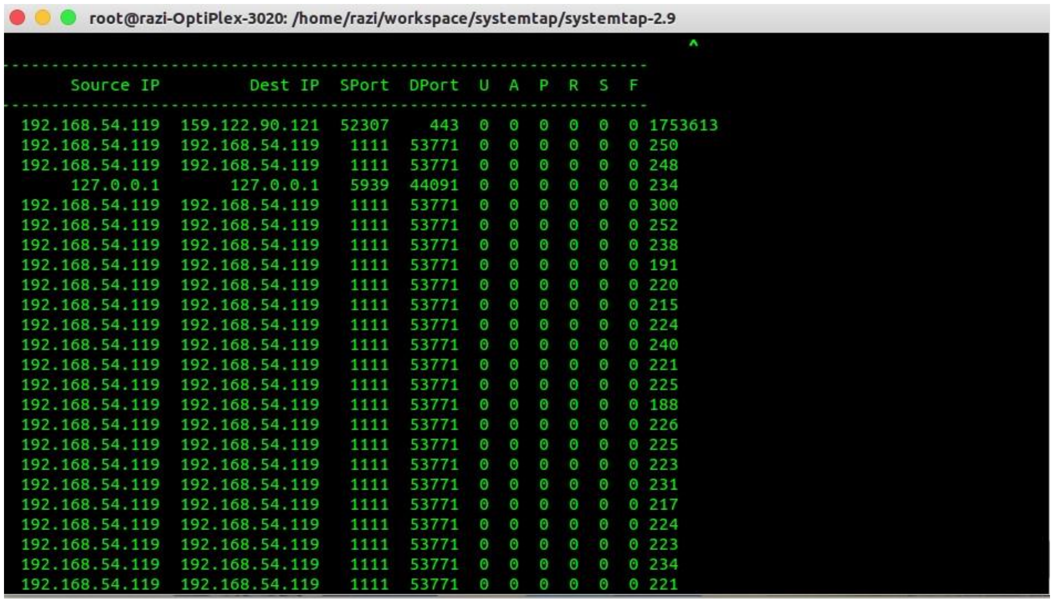
[3] Image taken from http://best1articles.blogspot.ca/2013/03/working-of-7-layers-of-osi-network-model.html
[4] Image taken from http://best1articles.blogspot.ca/2013/03/working-of-7-layers-of-osi-network-model.html

Appendix B



```
#! /usr/bin/env stap

global time_in, time_out, received =1, transferred = 0

probe begin  { //, timer.s(10)
printf("-----------------------------------------------------------------\n")
printf("       Source IP         Dest IP  SPort  DPort  U  A  P  R  S  F \n")
printf("-----------------------------------------------------------------\n")
}

probe tcp.recvmsg.return{
if(transferred ==1)
{
  time_out = gettimeofday_us();
  printf(" %15s %15s  %5d  %5d  %d  %d  %d  %d  %d  %d %d\n",
       saddr, daddr, sport, dport, urg, ack, psh, rst, syn, fin, time_out -time_in);
       transferred =0;
       received = 1;
}
}

probe kernel.function("eth_type_trans") {
  if(($skb->protocol == 8)&& (received ==1) && ($skb->mac_header == 238)){
   time_in = gettimeofday_us();
   received =0;
   transferred =1;
  }
}

```

Figure 15: Sample SystemTap Script



root@razi-OptiPlex-3020: /home/razi/workspace/systemtap/systemtap-2.9

```
-----------------------------------------------------------------
      Source IP         Dest IP  SPort  DPort  U  A  P  R  S  F
-----------------------------------------------------------------
  192.168.54.119  159.122.90.121  52307    443  0  0  0  0  0  0 1753613
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 250
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 248
       127.0.0.1       127.0.0.1   5939  44091  0  0  0  0  0  0 234
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 300
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 252
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 238
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 191
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 220
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 215
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 224
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 240
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 221
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 225
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 188
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 226
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 225
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 223
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 231
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 217
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 224
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 223
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 234
  192.168.54.119  192.168.54.119   1111  53771  0  0  0  0  0  0 221
```

Figure 16: Terminal output of SystemTap script