FACULTY
OF INFORMATION
TECHNOLOGY
CTU IN PRAGUE

ASSIGNMENT OF BACHELOR'S THESIS

Title:	Testing environment for probe server
Student:	Atsamaz Akopyan
Supervisor:	Ing. Andrey Reutov
Study Programme:	Informatics
Study Branch:	Web and Software Engineering
Department:	Department of Software Engineering
Validity:	Until the end of winter semester 2020/21

Instructions

The functionality of the probes is collecting, processing, correlating and storing signaling and user data from the monitored network. It means that probes are passively listening to data on the TELECOM network without influencing the network itself in any way.
The aim of the thesis is to design and develop a testing environment for the probe server.
Instructions
1. Identify parts of the probe that can cause bottlenecks and need to be monitored.
2. Analyze ways in which .pcap files can be obtained and generated. Demonstrate the generation of such files.
3. Create a program that finds the optimal traffic load and records utilization of identified parts. The program will consist of the following:
a) Automation of traffic generation according to some configuration.
b) Algorithm to find an optimal traffic load.
c) A data collection algorithm.
d) Presentation of data.
4. Document your implementation.

References

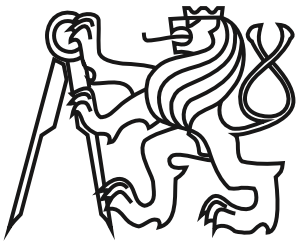
Will be provided by the supervisor.

Ing. Michal Valenta, Ph.D.
Head of Department

doc. RNDr. Ing. Marcel Jiřina, Ph.D.
Dean

Prague February 25, 2019

CZECH TECHNICAL UNIVERSITY IN PRAGUE
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF SOFTWARE ENGINEERING



Bachelor's thesis

Testing environment for probe server

Atsamaz Akopyan Bachelor

Supervisor: Ing. Andrey Reutov

27 June 2019

Acknowledgements

I would like to thank my supervisor Ing. Andrey Reutov for helping and giving valuable advice when writing this thesis.

I would also like to thank my family for giving me this opportunity and supporting me during my studies.

Declaration

I hereby declare that the presented thesis is my own work and that I have cited all sources of information in accordance with the Guideline for adhering to ethical principles when elaborating an academic final thesis. I acknowledge that my thesis is subject to the rights and obligations stipulated by the Act No. 121/2000 Coll., the Copyright Act, as amended. In accordance with Article 46(6) of the Act, I hereby grant a nonexclusive authorization (license) to utilize this thesis, including any and all computer programs incorporated therein or attached thereto and all corresponding documentation (hereinafter collectively referred to as the "Work"), to any and all persons that wish to utilize the Work. Such persons are entitled to use the Work in any way (including for-profit purposes) that does not detract from its value. This authorization is not limited in terms of time, location and quantity. However, all persons that makes use of the above license shall be obliged to grant a license at least in the same scope as defined above with respect to each and every work that is created (wholly or in part) based on the Work, by modifying the Work, by combining the Work with another work, by including the Work in a collection of works or by adapting the Work (including translation), and at the same time make available the source code of such work at least in a way and scope that are comparable to the way and scope in which the source code of the Work is made available.

on 27th June 2019 .....

Czech Technical University in Prague

Faculty of Information Technology

Citation of this thesis

Atsamaz Akopyan. Testing environment for probe server. Czech Technical University in Prague, Bachelor's thesis. Faculty of Information Technology, 2019.

Abstract

Telecommunication providers may serve over a million customers. These providers need to know how well their service is doing, when there are peak loads, what traffic interests' their users most and exedra. So, if your task is to build a probe, whose job is to copy and analyze a large portion of traffic between the provider and its customers, you must know how well your probe is performing. At Anristu, our probes are x86 servers based on CentOS (Linux distribution, version 6+). Our probe software is responsible for obtaining and filtering packets for future analysis and this Probe software must utilize all the processing power of the system before the system begins to lose packets.

This project resulted in a program that is designed to automate testing of a probe system. The program, Automatic Performance Testing or APT, tests a probe by finding the maximum speed at which the server can analyze traffic, meaning it determines if there are packets being lost and at what traffic speed (i.e. Mb/s). It then gives the user a list of averaged utilization readings for various parts of the probe. The data is represented in a CSV file format. This data is all that the user needs to determine where a bottleneck is and to approve of specific hardware or probe software for traffic analysis

Keywords

CSV, Probe, li-ether-sim, Qmpa5, x commands, inspect commands, FBC or Fiberblaze card

Abstrakt

Poskytovatelé telekomunikací mohou obsluhovat více než milion zákazníků. Tito poskytovatelé potřebují vědět, jak dobře jejich služby fungují, kdy a jaké špičkové zatížení jsou, o jaký provoz mají jejich uživatelé nejvíce zájem a dále. Pokud je tedy vaším úkolem vybudovat sondu, jejíž úlohou je kopírovat a analyzovat velkou část provozu mezi poskytovatelem a jeho zákazníky, musíte vědět, jak dobře vaše sonda funguje. Na Anristu jsou naše sondy x86 servery založené na CentOSu (distribuce Linuxu, verze 6+). Náš sondový software je zodpovědný za získání a filtrování paketů pro budoucí analýzu a

tento software sondy musí využívat veškerý procesní výkon systému dříve, než systém začne ztrácet pakety.

Výsledkem tohoto projektu byl program, jehož cílem je automatizovat testování systému sond. Program, automatické testování výkonnosti nebo APT, testuje sondu tím, že najde maximální rychlost, při které server může analyzovat provoz, což znamená, že určuje, zda se ztratí pakety a rychlost provozu (tj. Mb / s). Poté dává uživateli seznam průměrných hodnot využití pro různé části sondy. Data jsou reprezentována ve formátu souboru CSV. Toto je vše, co uživatel potřebuje, aby určil, kde je úzké místo, a aby schválil specifický hardware nebo software sondy pro analýzu provozu.

Klíčová slova

CSV, Probe, li-ether-sim, Qmpa5, x commands, inspect commands, FBC nebo Fiberblaze card

Contents

Table of Figures

1 Introduction

1.1 Probe subsystem in MasterClaw

The 1.1 section explains the purpose, functionality and diagnosis of the probe. For this project, it is not necessary to understand exactly how the probe functions but rather get an idea of what a probe is and which of its parts are important.

Every network will experience some sort of problems and our goal is to make sure that issues can be addressed before they can lead to complaints from customers. Anritsu's troubleshooting toolbox is designed to resolve problems efficiently across wireless and fixed IP networks. The product, MasterClaw or MC for short, uses real time and historical data that it extracts from the network to support the entire troubleshooting process from the location and analysis of the problem to problem impact analysis and verification of corrective actions. Operators can examine customized data views through an intuitive web/browser-based portal or a desktop application where technical network data is presented as understandable operational information.

MasterClaw is a distributed un-intrusive system for troubleshooting, monitoring and service assurance of GSM, GPRS, UMTS, LTE, VoIP, IMS and other networks. The probe subsystem consists of number of probes and info-servers, plus some additional elements like GPS boxes and antennas, TAPs, SPAN port configurations and LAN connections. The probe subsystem is interfacing the monitored network in one side and MC server on the other side, as shown on the picture below.

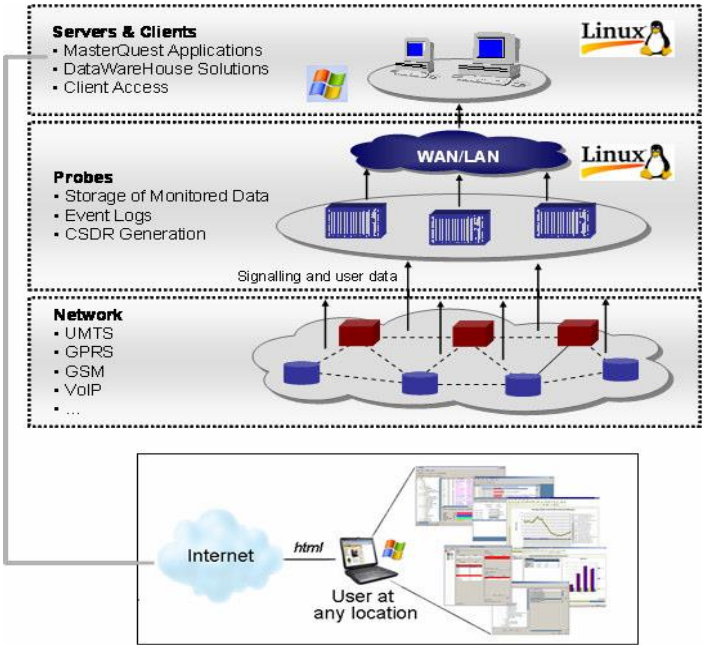


Figure 1 Probe subsystem in MasterClaw architecture [1]

The probe subsystem is attached to the monitored network in un-intrusive way. It means that MasterClaw probes are passively listening to data in the network without influencing the network itself in any way. For maintenance and troubleshooting, the probes and info-servers can be accessed by SSH or sftp using the MasterClaw or root accounts.

1.2 Probe functionality overview

As seen from a higher level, the functionality of MC probes is collecting, processing, correlating, storing, and signaling user data from the monitored network.

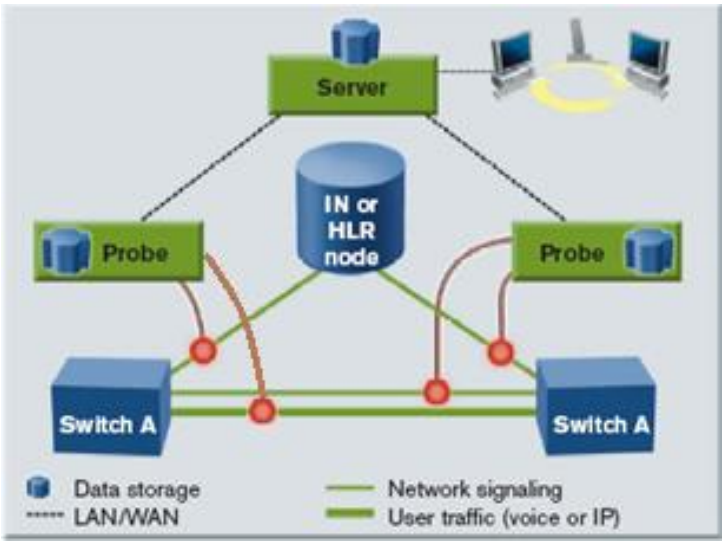


Figure 2 Probe's functionality schema [1]

MC probes provide the ability to monitor, record and basically process the signaling of various protocols. The main subjects of this processing include:

- Capturing and storing signaling data (PDUs) into event logs on probe disk
- Correlating PDUs belonging to one call/transaction/session and gathering summary information from them into CSDRs, storing CSDRs into Csdr logs probe disk
- Generating statistical counters and alarms (aggregated in memory, read by statistical server)

The probe needs to filter and copy large amounts of traffic without losing any packets. The probes are sold by Anritsu to its customers who are telecom network operators.

Figure 3 Anritsu's Lab Servers in Prague

This is Anritsu's lab in Prague and this lab consists of HP servers placed in several racks. These servers are based mostly on Intel's Xeon platform and they are of different generations. The servers are identical copies of what our customers have. Anritsu uses enterprise server hardware. With server platforms we are open to more hardware options, such a greater amount of CPU cores, up to two CPU's, greater RAM capacities, more storage options and higher PCIE bandwidth. This is necessary because a telecom provider may have over a million customers and analyzing all this traffic puts heavy load on the hardware. CPU, RAM and storage disks are usually almost fully utilized.

Anritsu probe servers[1] run a custom Linux distribution called CentOS or Community Enterprise Operating System, which is a computing platform that is functionally compatible with its upstream source, Red Hat Enterprise Linux [2]. The following OS versions are supported:

- 64bit Red Hat Enterprise Linux 7 (7.3)
- 64bit CentOS-7 (7.3)
- 64bit Red Hat Enterprise Linux 6 (6.6)
- 64bit CentOS-6 (6.6)

[1] A server on which the probe runs.

1.3 Components of the probe

- Qmpa5 software package uses dynamically loadable libraries for decoding of protocols and normalization of CSDR's. The package provides the ability to monitor, record and basically process the signaling of various protocols.
- li-ether-sim's purpose is to intercept the signaling traffic or user plane traffic carried on IP. To strip the tunneling protocols and to deliver the content to qmpa5 as PDU messages. Further the generated statistics, alarms are delivered to qmpa5 via TCP socket-based SIM interface. In addition to standard ethernet interfaces the IP traffic may be captured also using special board types for higher performance using included kernel drivers.

The operating systems that are installed on probes are packaged with Anritsu's custom terminal commands that were written by testing teams and allow for diagnostics and troubleshooting of probes and their different components. "X commands" are used for diagnosis and troubleshooting of qmpa5. "Inspect" commands are used for diagnosis and troubleshooting of li-ether-sim.

When a probe is deployed, traffic is relayed to the probe server by via either ethernet or fiber optic ports at that are the rear IO of the servers. When a probe is connected to a traffic source via ethernet, the amount of traffic that can passthrough is limited up to 1Gbps but fiber optic allows for a much higher bandwith. Anritsu servers do not come with fiber optic ports built in to their motherboards so Anritsu purchases custom made Fiberblaze-cards that plug into PCIE x16 [2] slots on the motherboard. This special board type supports higher performance, i.e. over 1Gb's. Fiberblaze cards or FBC's are FPGA based interface cards developed and supplied by Silicom Denmark. Fiberblaze card is traffic a capturing card. These cards have the following features:

- Line speed capture, with packet loss protection, and detection.
- Multiple clients of the Fiberblaze card
- Buffers transferred directly into memory using DMA transfers.
- Defragmentation making sure that network streams are filtered to the same buffers.
- Deduplication of incoming packets reducing post processing.
- Filtering performed at line speed on Fiberblaze card.
- Choice between Non-blocking PRBs and additional HW buffering.

These features are very useful and therefore these cards are used very often.

[2] PCIe (peripheral component interconnect express) is an interface standard for connecting high-speed components. The number after the x tells you how many data lanes that port has.

1.4 Motivation

Currently, there is no automated way to test how quickly some version of a probe server can run and how many packets it can process. There is also no automated way to tell if a new version of probe software is faster or slower than previous versions. This means that older systems being updated to newer, potentially slower, versions could risk data loss[3] on already deployed probes. Furthermore, changes in hardware and newer systems also means that previously known maximums need to be retested.

Because of lack of automation, for Anritsu to find the maximum bandwidth/load a probe server can handle, it must be done manually by employees and this takes up many man hours. For an employee to benchmark[4] one specific traffic load they must somehow initiate traffic flow through a probe and then wait and check if data loss occurs. Whilst they are waiting and constantly checking for data loss on the probe, they must also pass dozens of commands to the terminal and record the output. Some of these commands yield several screen's worth of trouble shooting and diagnosis data that the employee must navigate through quickly. They must repeat this process until they find an optimal speed. Then that must be repeated for different protocols and configurations. Needless to say, executing all these instructions can be very error prone. Hense, sometimes we are unsure of whether to even trust the results obtained. Even worse is when there are results that contradict one another, and everything needs to be retested.

There needs to be a way to automatically benchmark a server running some probe instance, where an employee could simply setup a probe, configure an automation script/program for a test and then focus on something else while the script/program finds an optimal speed with benchmarking data. This should take up as little time as possible.

[3] Data loss means that the probe is losing packets, i.e. it is not managing to filter/copy or do whatever it needs to do with these packets quickly enough and has to skip some.

[4] "*In computing, a benchmark is the act of running a computer program, a set of programs, or other operations, in order to assess the relative performance of an object, normally by running a number of standard tests and trials against it."* [16]

1.5 Previous work on this issue

Manually trying to search for data loss and record utilization of all the different parts of the probe by passing diagnosis commands to the terminal was too time consuming. This is because there are too many commands to execute and too much output to record. Especially the case when all these commands must be repeated for every traffic load tested.

Previously, an employee was given the task to automatically collect all information about a running probe. The result was a bash shell script called the Probe-info script. This script automatically collects various utilization data about the probe server. The script can record protocol loads, li-ether-sim load, qmpa load, FBC load etc.

However, the script primarily serves a different purpose, and that is to read the configuration of a probe on a server so that this probe can be recreated later. This is used to help recreate bugs. The issue is that, if used in benchmarking, executing this script several times yields several megabytes worth of redundant data. Making things worse, the Probe-info script also takes several minutes to complete and only reads utilization data once. Therefore, outputted readings are plagued with anomalies and it is questionable if conclusions can be made off of a single reading. Running this script several times is an option, but this again takes up time. Also, the script has to be re-run for every speed that is being benchmarked. Furthermore, reading through the output of the script to find relevant information is made difficult by the sheer amount of folders, files and data generated.

Given time, running the script several times to obtain reliable readings that can be averaged and trimmed is possible. Then again, the problem with redundant data and many man hours wasted persists.

1.6 Ideal solution

The ideal solution would be a script or program that is designed entirely for benchmarking a specific probe instance on a server. This script or program should be such that an employee only has to configure a file and then run the program. The program then must find an optimal traffic load that the server can handle. At the same time, it must record all the relevant server utilization data, trim and average the data and then present it to the employee in a clear and readable format.

Anritsu often adds extra features for their probes. This ideal solution should also be expandable so that extra features such as probe parts to be monitored can be added for testing. This program should also run on the existing probe server environments without the need to install any compilers or interpreters or such. The program will be packaged with the probe server OS.

1.7 Outline of project

To build an ideal solution for the project, we must first analyze requirements and the tools that could be used. Then decide which of these tools will suit the requirements best. This project discusses the ways to solve the problem, the tools that can be used, the tools that were chosen and why. Then there is the implementation phase that shows snippets of the code and explains logic and reasoning for this specific implementation. The testing and deployment phase discusses issues found when testing and deploying the solution in a working environment. The future work section talks about possible extensions and improvements that can done. In conclusion there will a reflection on the tools chosen, functionality of the program or script and its use in the working environment.

2 Analysis and design

This chapter will briefly explain where a source of potential bottlenecks is and what the requirements are. It will discuss which tools are going to be used to solve these requirements and why the tools were chosen.

2.1 Potential bottlenecks and what needs to be monitored

A bottleneck is as the name suggests, is when the capacity of an application is severely limited by some single component, like the neck of a water bottle slowing down water flow. There are several software parts in a probe that could cause a bottleneck and if a bottleneck occurs then data loss will follow. Obviously, data loss is something that needs to be avoided and so parts of the probe that can cause bottlenecks need to have their utilization monitored.

First off, basic CPU, RAM and disk utilization needs to be monitored to make sure that neither of the hardware components are a limiting factor. If for instance, total CPU utilization is at 99.9% then by monitoring this number we can conclude it was responsible for data loss, i.e. server cannot handle more traffic as it is hardware limited.

Secondly, previously mentioned probe parts such as qmpa5 and li-ether-sim are resource heavy. They can consume large amounts of CPU or RAM and the extent of their allocation is set in probe's configuration. Given a multiple core server, several threads can be assigned to qmpa5 and several threads to li-ether-sim. If for instance, qmpa5 is utilizing the cores it was assigned to 99.9% then that will cause data loss. The same applies to li-ether-sim and therefore, both parts need to be monitored for their CPU and RAM utilization. A probe can deal with several protocols and for each protocol several threads can be assigned. The utilization of these threads and RAM consumption must also be monitored.

Other data such as raw save, data save, disk write, compression ratios and CDRS/s for each FID will have to be monitored. With additional configuration data such as cache allocated to qmpa5 and etc.

Fiberblaze cards process traffic directly, and if they encounter a spike in traffic that cannot be proccessed quickly enough, that traffic will be saved onto their buffer. This buffer can overflow if the spike in traffic lasts long enough. Therefore, fiberblaze cards too can become a source of data loss and will also need to be monitored.

2.2 Functional and non-Functional Requirements

The project's requirements will be split into functional and non-functional. Functional requirements will be related to the functionality of the system and non-functional requirements will describe the characteristics of the system. All functional and non-functional requirements are all considered to be mandatory for the system to function properly.

Functional requirements will be noted as Fn, where n is the number of the requirement.

Non-functional requirements will be noted as Nn, where small n is again the number of the requirement.

Functional Requirements

F1. The program must be configurable by the user. Based on the configuration, the program should know how to get some traffic source to send traffic to the probe. It must know how it will test different traffic loads and benchmarking the probe. The program should be able to be its own traffic source if required.

F2. The program must record idle utilization of the system so that the user can see probe server load before any traffic is replayed.

F3. Given a configuration, the program must be able to find the highest traffic load for a given acceptable data loss in Mb/s or Gb/s.

F4. The program must generate benchmarking data that shows the utilization of the entire server and all the identified probe components which could be responsible for bottlenecks. The generated data should also describe the probe that was tested.

F5. A user should be able to open the output in Excel and then graph it. This is because that's what the team uses to read performance data.

F6. The program should have the option to run a probe info script whilst playing traffic at the optimal speed once it's done benchmarking so that the probe can be recreated later.

F7. All the output that the program generates must be a trimmed average to avoid anomalies.

Non-functional Requirements

N1. The program must be compatible with both versions of CentOS that Anritsu uses on their probes. CentOS version 6.6 and version 7.0.

N2. Adding extra features to the program, such as expanding it to read more utilization data on for newly implemented features on a probe should not represent a problem. A programmer should only have to add these extra features and not have to rewrite anything.

2.3 Technical Challenges

- Some data can only be obtained by passing commands to the Linux terminal. Therefore, whatever language is used, it must be able to communicate directly with the Linux terminal.
- Terminal commands will yield a lot of irrelevant information which needs to be filtered out to obtain a reading which will be written into a variable. For example, if qmpa5 load in Mb/s is required, and the output from the terminal is "qmpa5 load = 412351 b/s", then we are only interested in the number and the significant figure. This irrelevant information will need to be filtered through quickly.
- To obtain all the mandatory data, a lot of commands will have to be passed to the terminal. For every command, there is a different output that needs to be filtered in its own unique way to obtain the required variable.
- All the data must be stored somewhere, and this data is unique, therefore it cannot all be stored in one container. For every piece of unique data, a separate variable will have to be used and then there must be a container for the trimmed average of these variables. And to obtain this trimmed average, data must be collected several times at different intervals in time.
- An algorithm must benchmark various traffic loads (speeds) for stability and make sure no data loss occurs. It must also take readings during a benchmark at a specific speed. Then it must quit once it found the biggest load that yields no data loss. User's may want different algorithms and test using different traffic generation methods.
- Probes servers are not only running different versions of CentOS but also have different probe versions which may have slightly different diagnosis output to terminal commands.

2.4 Application design

It would be a good idea to name the solution so that it can be referred to later. Automatic Performance Testing or APT for short is something that is easy to remember, and my supervisor was satisfied with this naming. Next, for APT to fulfill the requirements, we must consider what it must do. It would help to write down every major action that APT must perform, in order, from first to last. The testing team would prefer if APT is launched directly on the probe that is being tested.

Task	Simplified overview of tasks APT must perform.	Requirements fulfilled
1.	APT runs on the server and reads the configuration file.	F1 and N1
2.	APT takes all idle utilization readings and information about the probe.	F2 fulfilled. F4 partially fulfilled.
3.	APT connects to the specified traffic source and starts a flow traffic there.	F3 partially fulfilled.
4.	APT will record utilization data whilst traffic is playing.	F4 partially fulfilled.
5.	APT will test for data loss.	F3 partially fulfilled.
6.	APT will determine the maximum traffic load at which there was no data loss. This is done once benchmarking has finished.	F3 fulfilled.
7.	APT will write out all recorded data in an excel compatible format.	F4 and F5 fulfilled
8.	APT starts probe info script and starts flow of traffic at the optimal traffic speed which had no data loss.	F6 is fulfilled.

In the first task, it makes sense to read the configuration before taking any idle readings as this way, we can be certain that the user input is correct, and APT should continue without error for task two.

To perform the third task, there must be a way to continuously play traffic to the probe at specified speed/load. The probe itself cannot be responsible for generating this traffic as that would hinder the utilization readings because continuous generation of traffic would put some load on the probe.

Task four, five and six will have to be performed by some algorithm and users may want to test in different ways.

Task eight will result in files generated by probe info and those will have to be placed in a folder along with the output data file. The folder should have a time stamp, so the user knows when the testing was conducted. Files generated by probe info will allow the user to see probe configuration and recreate the probe if needed.

It does not make sense to use a database to store either variables or output. All the variables holding utilization data during APT's execution can be temporarily held on the servers RAM. The output can be stored in the file with a timestamp and. The output file will be written once APT finished running and all the data is available.

The configuration of APT can be a text file in a folder with the APT executable. The user will be able to edit this file to input their configuration. This file should be called "configFile" or something similar so that the user can easily identify where they need to specify their configuration. This configuration file must be simple enough so that the user will manage to navigate it without any training. After editing this file and inserting a correct configuration the user will start APT which will read the file. However, APT must also handle invalid input/configuration in the file and warn the user where they made a mistake. That would be a suitable way to configure APT.

The basic algorithm could find an optimal traffic load by using steps or iterations. If a certain traffic load is stable, then increment the load and test again. Repeat until an unstable load is found, then the optimal stable load will be the previously tested load. In this algorithm, increments and starting traffic load are specified by the user. This is the iterative algorithm which will be designed. This is because it allows to benchmark all traffic loads from the very low speeds up to a high unstable point and at the same time generate data on probe's utilization during all those increments.

Yet, aside from the incremental algorithm, the testing team would also like to specify several traffic loads on their own, which the program should test. For instance, test speeds of 50, 100, 500, 2000, 5000 Mb/s and only these speeds, then report on findings. These traffic loads/speeds will have to be defined in the configuration file.

Several utilization readings can be taken during a benchmark of one specific speed. This will allow for averaging and trimming of data. Such that, qmpa5's threads are utilized at 75% at a traffic load of 500Mb/s. The user should be able to choose between the iterative and the manual algorithm in the configuration file.

2.5 Traffic Generation

Since the task is to find an optimal traffic load on the probe, then there needs to be some way to put variable traffic loads on probe. Also there must be a way to switch between different types of traffic for different probe configurations. Probes receive traffic from telecom provider's switches but in this case a deployed probe needs to be receiving traffic from a server. This server should be named the "Traffic generator". The traffic generator will be connected directly to the probe, either via an ethernet or fiber optic port.

The following picture depicts two servers that are connected via a fiber optic LC port.

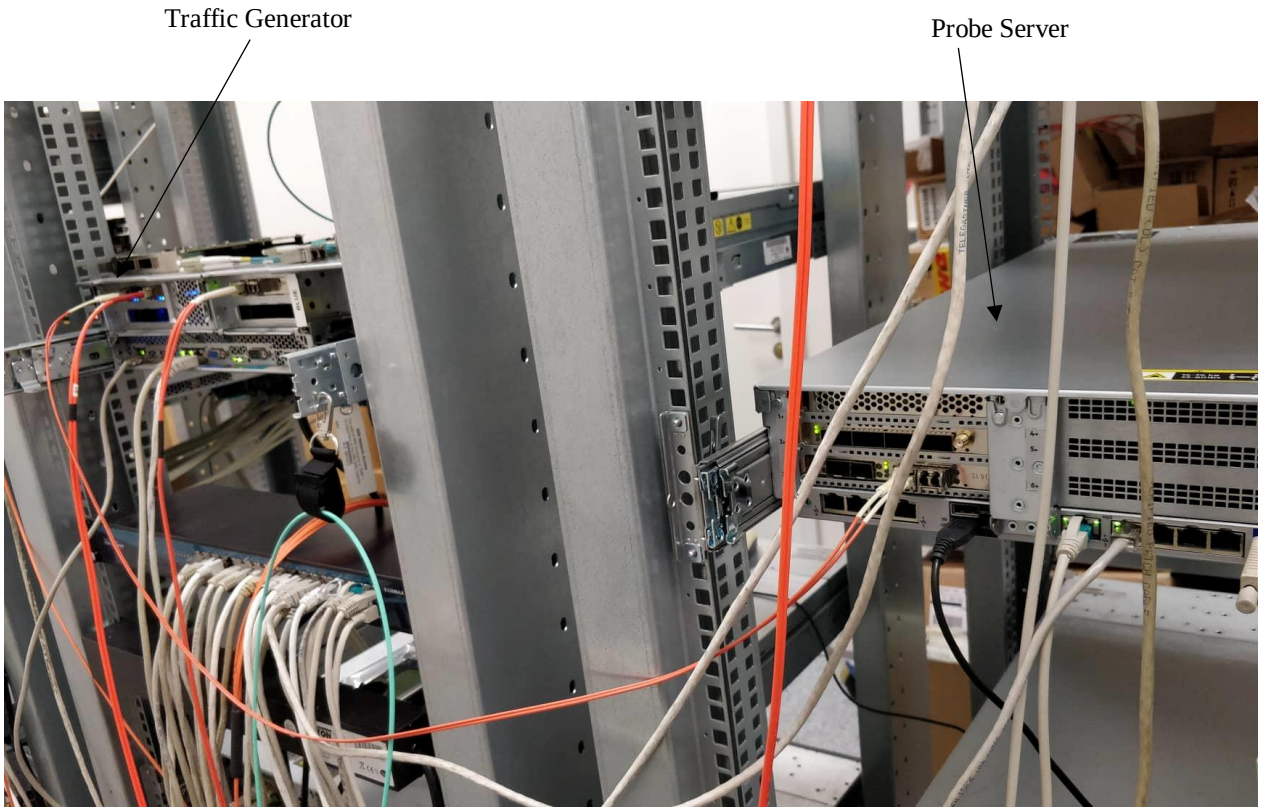


Figure 4: Traffic Generator and a Probe Server

This traffic generator will send out traffic via a port specified in the configuration. It will do so using some tool. This traffic will be in the from a .cap file that gets replayed over and over. A '.cap' file extension signifies that the file contains packets collected by a packet sniffing program and contains raw data captured over a data transimission. This data will need to be captured in some way, it will also need to be a real simulation of traffic for a probe. The probe will receive these packets through a specific port and its objective will be to filter and save data.

Next problem however, is where to obtain real .pcap files. Anritsu used to be able to request sample traffic files from the telecom network providers and they would end up providing .pcap files for various protocols. Unfortunately, with the new european GDPR [3] laws, this has become difficult. GDPR stands for General Data Protection Regulation and was agreed upon by the European Parliament and Council in April of 2016. It has come into effect on May 25, 2018 and companies that are not compliant will be subject to penalties and fines. The two key privacy and data protection requirements of GDPR that make obtaining .pcap files an issue are:

- Requiring the consent of subjects for data processing
- Anonymizing collected data to protect privacy

Futhermore, this is imposed upon all members of the European Union, which includes the Czech Republic. Any company that markets their goods or services to EU residents is subject to these laws. In the end, this has an impact on data protection globally and means that some telecom providers outirght refuse to provide Anritsu with their traffic samples even if they are not in a nation that is part of the EU.

Fortunately, there are sample captures that can be obtained from trusted sources online. One of such sources is Wireshark [4]. They provide samples for many protocols, and these samples can be played directly to our probe. However, some samples with GTP protocols are not available and obtaining/editing packets to create new patterns will require using some tools. These tools will be mentioned later in the Tools Chosen section.

2.6 Depicting the application

Usually use case analysis is used to identify system requirements and to design classes that later fulfill these requirements [5]. For this application to use as little employee time as possible, the only use case is that the user starts APT and waits its completion. Therefore, there is no need for a use case diagram. A deployment diagram can be used to show the structure/architecture of the system. In this diagram, devices represent two servers, a probe and the Traffic generator. Artifacts represent executable files, such as APT itself and the executable that will start traffic replay. Deployment spec is the configuration file that will be required by APT.

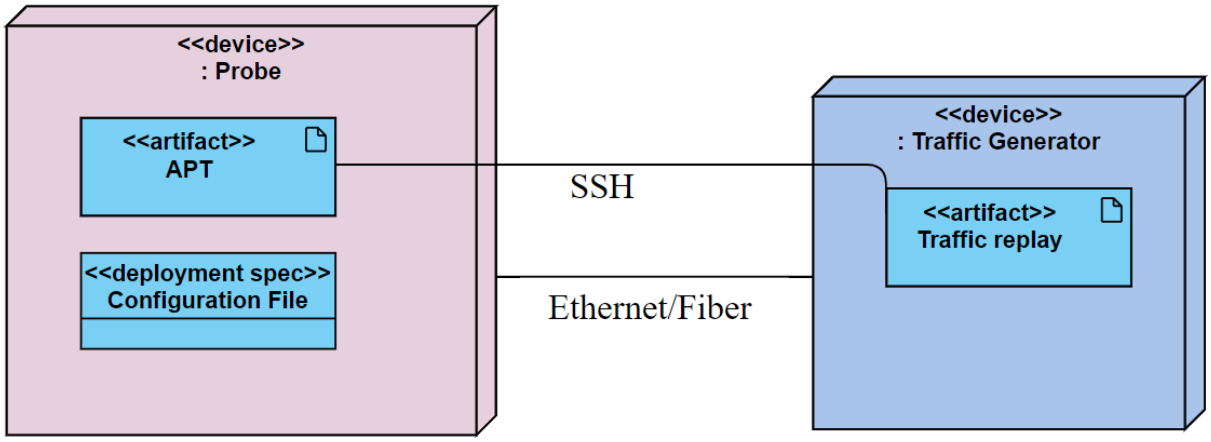


Figure 5 Deployment diagram

Figure 6 is an activity diagram. Activity diagrams look like flowcharts and they are used to model the activity flow of a system. [6] An activity diagram should do the following:

- Draw the activity flow of a system.
- Describe the sequence from one activity to another.
- Describe the parallel, branched and concurrent flow of the system.

This will help to understand in what how APT should operate. The following figure depicts APT running with the iterative algorithm.

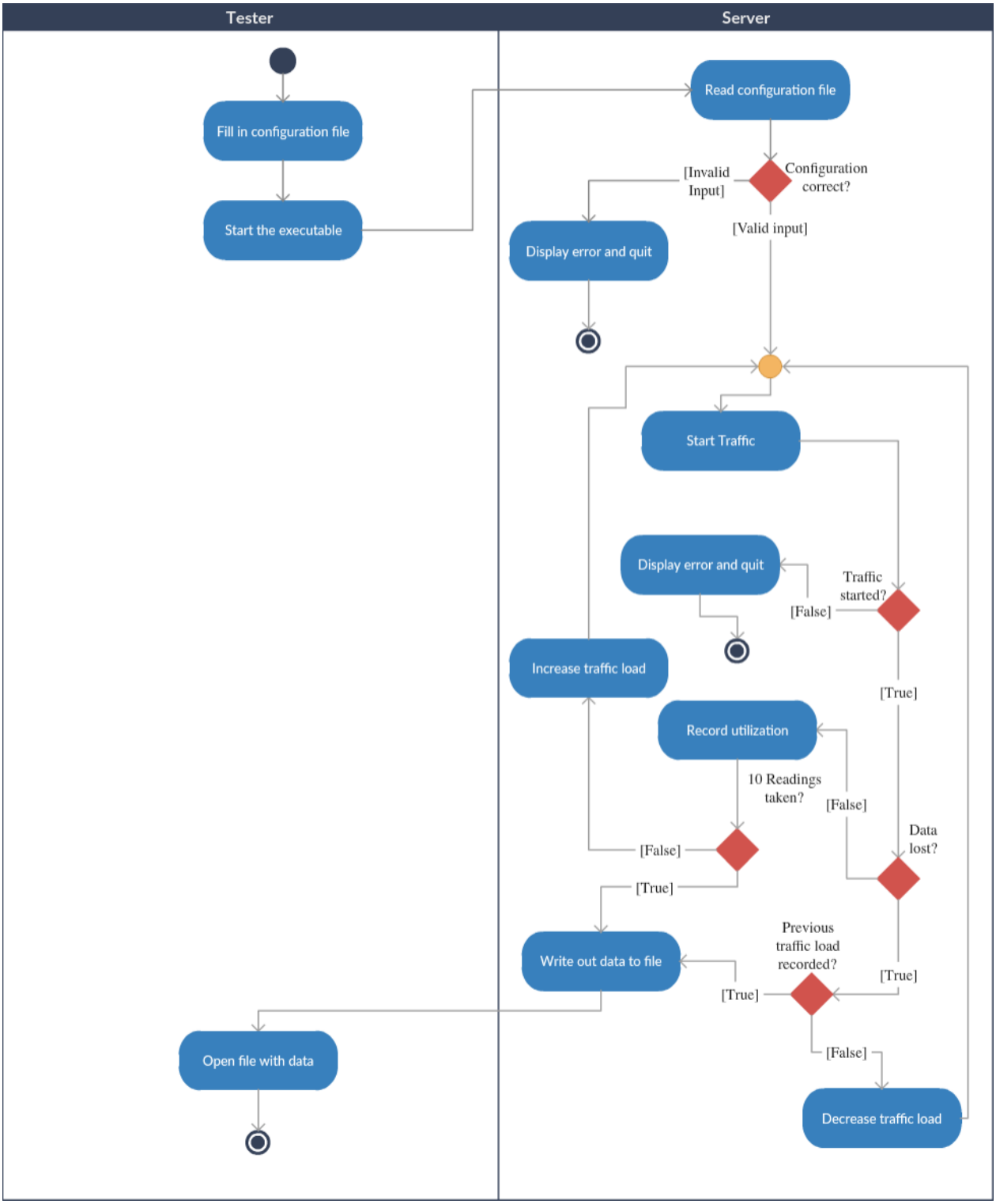


Figure 6 Activity diagram describing APT with incremental algorithm

2.7 Tools chosen

2.7.1 Scapy and WireShark

Traffic files can be generated with a tool called Scapy. It is a python program that is designed for packet manipulation. With it, packets can be forged, decoded and edited. Most importantly, it can sniff packets and then generate a .pcap file from those packets. Then using some tool, that .pcap file can be replayed to the probe. The probe will be unaware that the packets are just being replayed by some traffic generator. Wireshark, an open source packet analyser can be used to open these .pcap files and view contents.

2.7.2 TcpReplay/TXSD

The user can initiate the replay of traffic by using several commands that Anritsu pre-installs on all their servers's OS. These commands can be launched from the terminal and include "TCP replay", TXSD and TXS. Tcpreplay is an open source tool can only use ethernet. Other tools like TXSD and TXS use fibre ports on FBC's. These tools are used for editing and replaying previously captured network traffic.

However, APT cannot simply SSH to a traffic generator and start traffic replay by using one of the previously mentioned commands. First, the traffic generator, like all Anritsu servers, is protected by a password. An SSH command cannot bypass this and playing traffic will lock the terminal to that process as it is executing. Since the goal is to continuously play traffic on a traffic generator and look for data loss on the probe, being locked to the terminal is a problem.

2.7.3 Programming language

All data that needs to be collected can be obtained from terminal commands. The already existing script Probe-info is written in bash shell and the team is familiar with it, so it would seem logical to use bash shell scripting for this task. However, this Probe-info script is almost 1,000 lines, which makes it difficult to debug and even more difficult to add features. Also, additional lines must be added to filter output from the terminal as Probe-info script does not do that. Furthermore, working with lots of variables to trim and take averages proved to be quite difficult so something else would have to be used.

I proposed to use Python programming language as it is easier to debug and work with than bash scripting. Python also comes with great string manipulation features and is generally praised for its usability as a scripting language. However, our compilation server does not have the Python interpreter installed and the team is not familiar enough to easily expand APT if a need to add new

features arises. Furthermore, probes do not come with Python installed. Therefore, setting up a python environment just for this task is not an option.

A C++ compiler is preinstalled on all probe servers and the programming team uses this as their primary language when working with probes. They chose to write probe software in C++ because after conducting multiple tests, comparing C++ with other languages like Java, they concluded that C++ built the fastest programs. Therefore, expanding this program, were it written in C++ will not be a problem for them.

Importantly, for this task, C++ is capable of communicating with the Linux terminal via the system() function call. Also, it will be easier to debug and split the application into several files. However, the supported C++ version on our central compilation server is an older, g++ (GCC) 4.4.7 20120313 (Red Hat 4.4.7-11) version. The code must be compatible with this version of the compiler. The code will be written in Visual Studio Code, community edition, a free integrated development environment (IDE) tool by Microsoft to make things easier with debugging and writing the code.

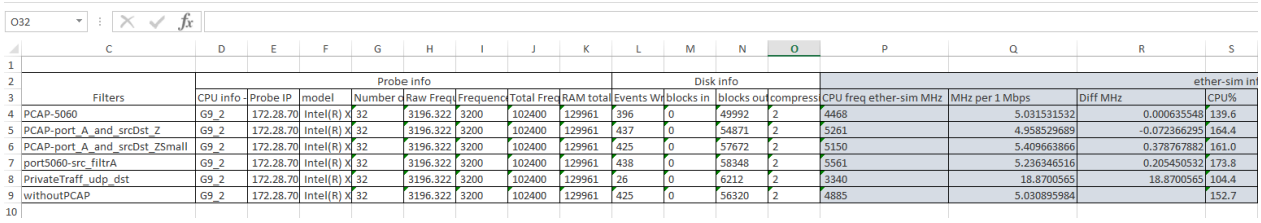
2.7.4 SSHPass

An issue with using a C++ program to SSH to a server is that APT will not be able to send a password if the server is password protected. As mentioned previously, what will happen is that if APT attempts to SSH to a password protected server, the server will reply with a request for a password and APT will hang indefinitely.

To solve this problem, a utility like SSHPass could be used. SSHPass runs in a "keyboard-interactive" mode which is required for password authentication, yet, it is running in a non-interactive mode for the user. [7] SSHPass runs SSH in a dedicated TTY to which it has direct access. TTY (TeleTYpewritter), also known as a terminal, is a device implemented in software which allows us to interact with the system by passing some data and then displaying some output that was produced by the system. [8] SSHPass runs the SSH in TTY to trick the SSH into thinking that it is obtaining a password from an interactive user. Running this command in the terminal as "SSHPass –p" allows the user to enter a password and IP address of the server that the user is connecting to. This command alone can allow us to make a system() call in C++ and SSH to the server we need.

2.7.5 CSV File

The summary of performance results will be displayed in a CSV file format. "*CSV means comma-separated values. This is a file format which is a text file that uses a comma (usually) or some other delimiter to separate values. This means that a CSV file can store columns of data in plain text.*" [9]This text file will be generated once APT successfully finishes testing. Anritsu testing teams already use manually put together CSV files to display performance data in Microsoft Excel, therefore they will be familiar with the output format of APT.

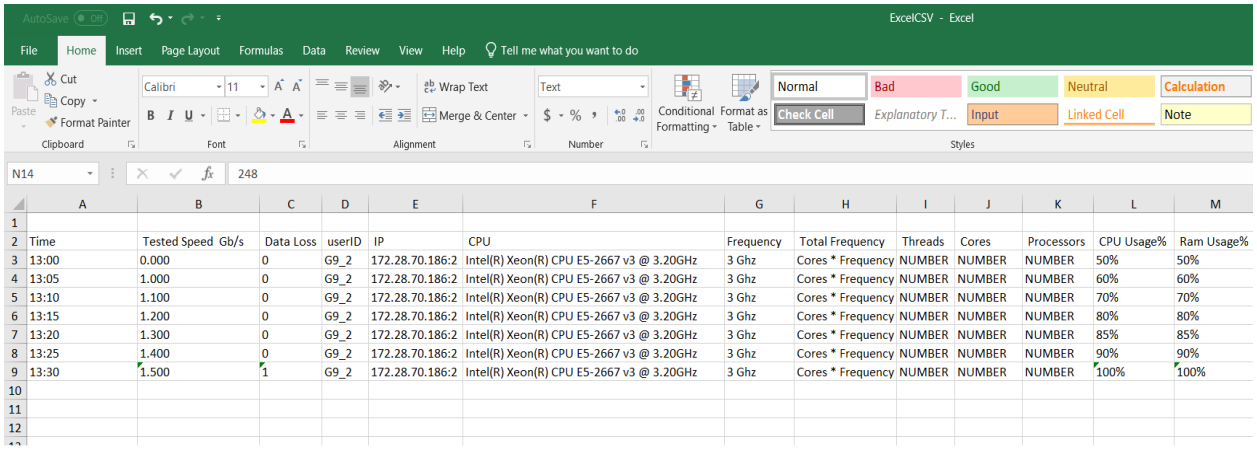


	Probe info								Disk info				CPU freq ether-sim MHz	MHz per 1 Mbps	Diff MHz	ether-sim int
Filters	CPU info	Probe IP	model	Number o	Raw Freq	Frequenc	Total Freq	RAM total	Events Wr	blocks in	blocks out	compress				CPU%
PCAP-5060	G9_2	172.28.70	Intel(R) X	32	3196.322	3200	102400	129961	396	0	49992	2	4468	5.031531532	0.000635548	139.6
PCAP-port_A_and_srcDst_Z	G9_2	172.28.70	Intel(R) X	32	3196.322	3200	102400	129961	437	0	54871	2	5261	4.958529689	-0.072366295	164.4
PCAP-port_A_and_srcDst_ZSmall	G9_2	172.28.70	Intel(R) X	32	3196.322	3200	102400	129961	425	0	57672	2	5150	5.409663866	0.378767882	161.0
port5060-src_filtrA	G9_2	172.28.70	Intel(R) X	32	3196.322	3200	102400	129961	438	0	58348	2	5561	5.236346516	0.205450532	173.8
PrivateTraff_udp_dst	G9_2	172.28.70	Intel(R) X	32	3196.322	3200	102400	129961	26	0	6212	2	3340	18.8700565	18.8700565	104.4
withoutPCAP	G9_2	172.28.70	Intel(R) X	32	3196.322	3200	102400	129961	425	0	56320	2	4885	5.030895984		152.7

Figure 7 Example of CSV File used by Anristu

This is a screenshot of excel, here excel was used to open a CSV file that the team put together as results of their testing on the probe server G9_2.

Writing a CSV file with C++ will not be difficult as all the output is just a text file and it will be simple to expand in the future. This CSV file will have a comma as the delimiter because this is most common and that's what Anritsu team's use. Then the file can be designed in a way that every row will represent a performance report at a given speed and every column will be an averaged reading for a component.



Time	Tested Speed Gb/s	Data Loss	userID	IP	CPU	Frequency	Total Frequency	Threads	Cores	Processors	CPU Usage%	Ram Usage%
13:00	0.000	0	G9_2	172.28.70.186:2	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	3 Ghz	Cores * Frequency	NUMBER	NUMBER	NUMBER	50%	50%
13:05	1.000	0	G9_2	172.28.70.186:2	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	3 Ghz	Cores * Frequency	NUMBER	NUMBER	NUMBER	60%	60%
13:10	1.100	0	G9_2	172.28.70.186:2	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	3 Ghz	Cores * Frequency	NUMBER	NUMBER	NUMBER	70%	70%
13:15	1.200	0	G9_2	172.28.70.186:2	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	3 Ghz	Cores * Frequency	NUMBER	NUMBER	NUMBER	80%	80%
13:20	1.300	0	G9_2	172.28.70.186:2	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	3 Ghz	Cores * Frequency	NUMBER	NUMBER	NUMBER	85%	85%
13:25	1.400	0	G9_2	172.28.70.186:2	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	3 Ghz	Cores * Frequency	NUMBER	NUMBER	NUMBER	90%	90%
13:30	1.500	1	G9_2	172.28.70.186:2	Intel(R) Xeon(R) CPU E5-2667 v3 @ 3.20GHz	3 Ghz	Cores * Frequency	NUMBER	NUMBER	NUMBER	100%	100%

Figure 8 What output of APT could look like

An example of what the CSV file could look like once opened in Microsoft Excel.

2.7.6 GitLab

There is a need for a version-control system. This is because saving different versions and tracking changes between these versions will simplify the debugging process. If some change accidentally breaks APT it will be easy to look back, see what the change was and revert to a stable version. Also, the ability to save APT somewhere other than the local PC would present a benefit. As with data integrity, if something should happen to the PC all the work won't be lost. Furthermore, a web-based repository is preferred as the project has to be shared between all offices that work with our lab.

Git is a distributed version-control system for tracking changes in source code and this tool suits our needs [10]. It is used to manage software development and for coordinating work between programmers. Git tracks all changes in a git repository. The repository contains a set of commit objects. It is simple to use and all that is required to start a git project is that using Git Bash you create a git repository. Once you write some code file you add it to a commit and once you commit there is a version of the project.

GitLab is a repository manager that Anritsu uses for its DevOps and it runs git [11]. It allows programmers to work concurrently as git supports nonlinear workflows via branching. However, this is not something that this project will require and therefore there will only be one branch, the master branch. However, this project still will utilize GitLab because it is a web-based repository meaning that it is possible to push commits and have the new versions be saved on a web server. This helps in sharing the project, as on our repository anyone who needs this tool can quickly download it. It also supports issue tracking which could come in handy when after taking a break from the project you need to remember where you left off and which features still need implementing.

2.7.7 Virtualization

The default company OS is Windows 10. When determining in which OS to write APT, an attempt was made to try and use CentOS 7.0 with virtual box. However, finding the right drivers to setup the GUI at the correct resolution in CentOS on a virtual machine proved too time consuming. Therefore, the decision was made to write the code in Ubuntu 18.04 which may not be exactly what the probes are running but it is similar enough not to cause any issues. The code is to be uploaded central compilation server for compilation. Then the executable can be started on a probe to test APT.

2.7.8 PuTTY and WinSCP

It will be important to test APT on our servers during the development phase to make sure that all features are working. Anritsu personal computers run on Windows 10 and therefore, additional tools are needed to communicate with CentOS based servers. PuTTY and WinSCP are the tools chosen for this task and the team already uses these them frequently. PuTTY is a free and open-source terminal emulator. [12] It will be used because it supports the SSH network protocol. Via this protocol we will connect to the servers, traverse their directories, modify files and run our program. WinSCP is also a free and open source program and we will use it for its SCP client that runs on our Windows machine [13]. It offers a GUI file manager that will greatly simplify file transfer between our local machine and the server.

2.8 Application architecture

In C++, majority of tasks are solved using an object-oriented programming model. In object-oriented programming, or OOP, programs are built around objects. An object can be a physical entity such as a car or a human. These objects can be described by several properties like type of vehicle, vehicle color, etc. In OOP objects communicate with one another via messages and use encapsulation to hide data so that this data cannot accidentally be modified. The only way to affect data is through the object's methods. Benefits of OOP include scalability and reusability. For these and many other reasons, OOP is widely used and should be considered. Especially for the scalability, as this is important because APT needs to allow for future extensions.

On the other hand, attempting to build an OOP solution results in a problem. This is because in this application most of the data will have to be read through the terminal and then written out which requires many unique functions and not objects. The application is not fit for OOP. For example, to obtain a certain value such as load of qmpa5 in Mb/s, a unique terminal command is required. As mentioned before, the output of the command has filtered to get the Mb/s number. A command to get ether-sim load in Mb/s is different and needs separate code to filter its Mb/s figure.

Therefore, the solution will utilize many different functions and each function will write into a separate variable and all of this will have to be grouped in some way. Using only traditional classes to design APT does not make much sense, as there is no need for multiple instances of variables/functions and no need for inheritance of any kind. A possible non-OOP solution could be using namespaces[5] or utility classes[6], each in separate files. This solution does not use the object-oriented programming model because the OOP model does not suit this application's demands and design. Instead APT will be developed in procedural programming where functions carry a series of computation steps to be performed.

It is important to consider where to temporarily store the collected data before it's written out. A utility class named like *DataBase.variable* can be used to temporarily store all the recorded variables in RAM during APT's execution before they need to be written out. Since these variables don't take up much space, even if there are hundreds of them, they will use a relatively small percentage of available memory, especially so, when considering the enterprise hardware of these probe servers.

Now to consider the groups of functions that will have to be used in collection of this data by APT and how these groups functions will differ.

---

[5] "*Namespaces are used to organize code into logical groups and to prevent name collisions that can occur*" [15]

[6] Utility class contains just static methods that are related. The class is stateless and cannot be instantiated.

The first group of data that will have to be collected is information about the probe server itself. Data such as the name of the probe, its hardware specifications, IP address etc. This data will be static and won't change during APT's execution. This data can be grouped into a namespace or utility class that can be named StaticData. Then there is the utilization data that will vary at different time intervals during APT's execution, and this data can be grouped into its own namespace with a name DynamicData. StaticData and DynamicData namespaces are located in separate files and are collections of functions.

For documentation there needs to be a user guide that will explain how to use APT. How to configure the config file and execute it. Then there needs to a programmer's guide that will explain the how APT is built and most importantly, how it obtains data. Then it should describe how extra features can be added.

Several classes will have to be used, but again these will only need to have one instance. Most importantly there needs to be a controller class, which will initialize and prepare the probe for testing, initialize other classes and handle their communication.

Other classes will be required, such as a class to read the configuration file, a class to write out the CSV file and a class to handle traffic. A utility class that will hold variables and output of commands can be named "Database". A working tree structure of APT files in GitLab should like the following:

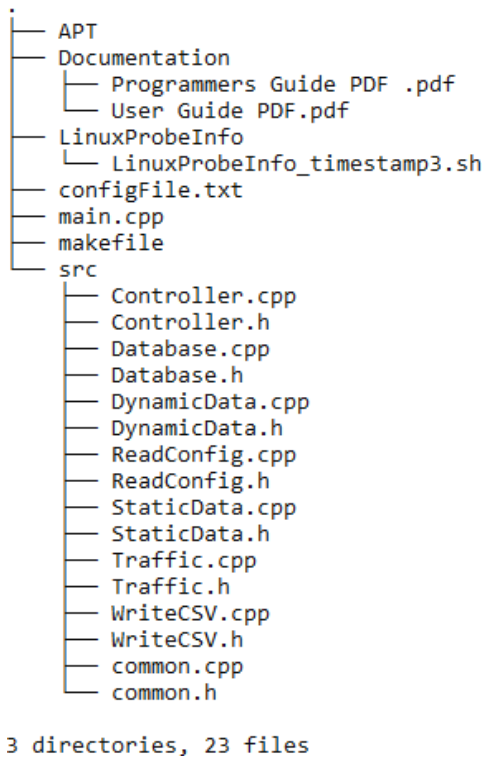


Figure 9 Working tree structure in GitLab

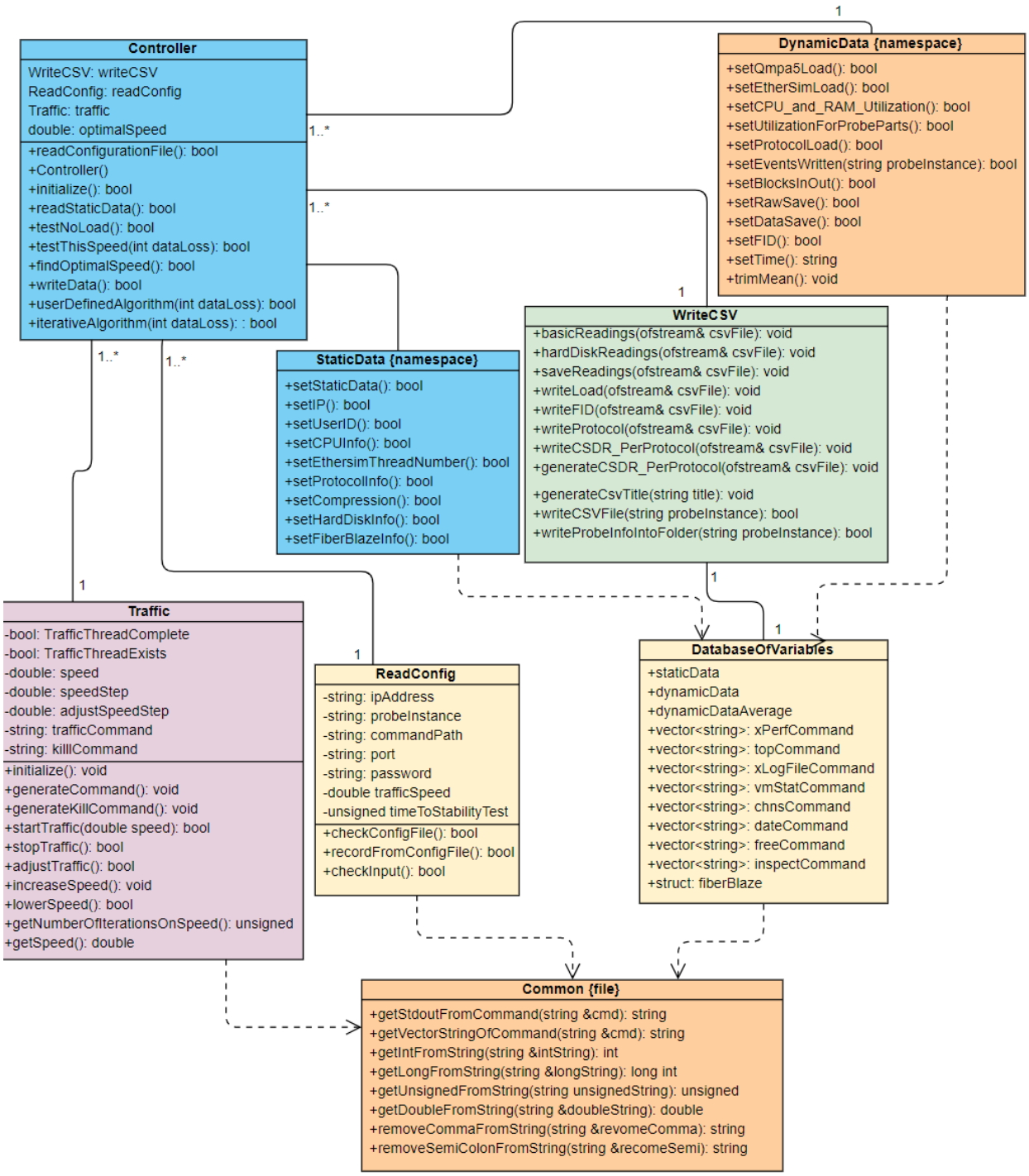


Figure 10 Class Diagram

The class diagram above does not show all the functions and variables as during the implementation stage several more functions and variables were required.

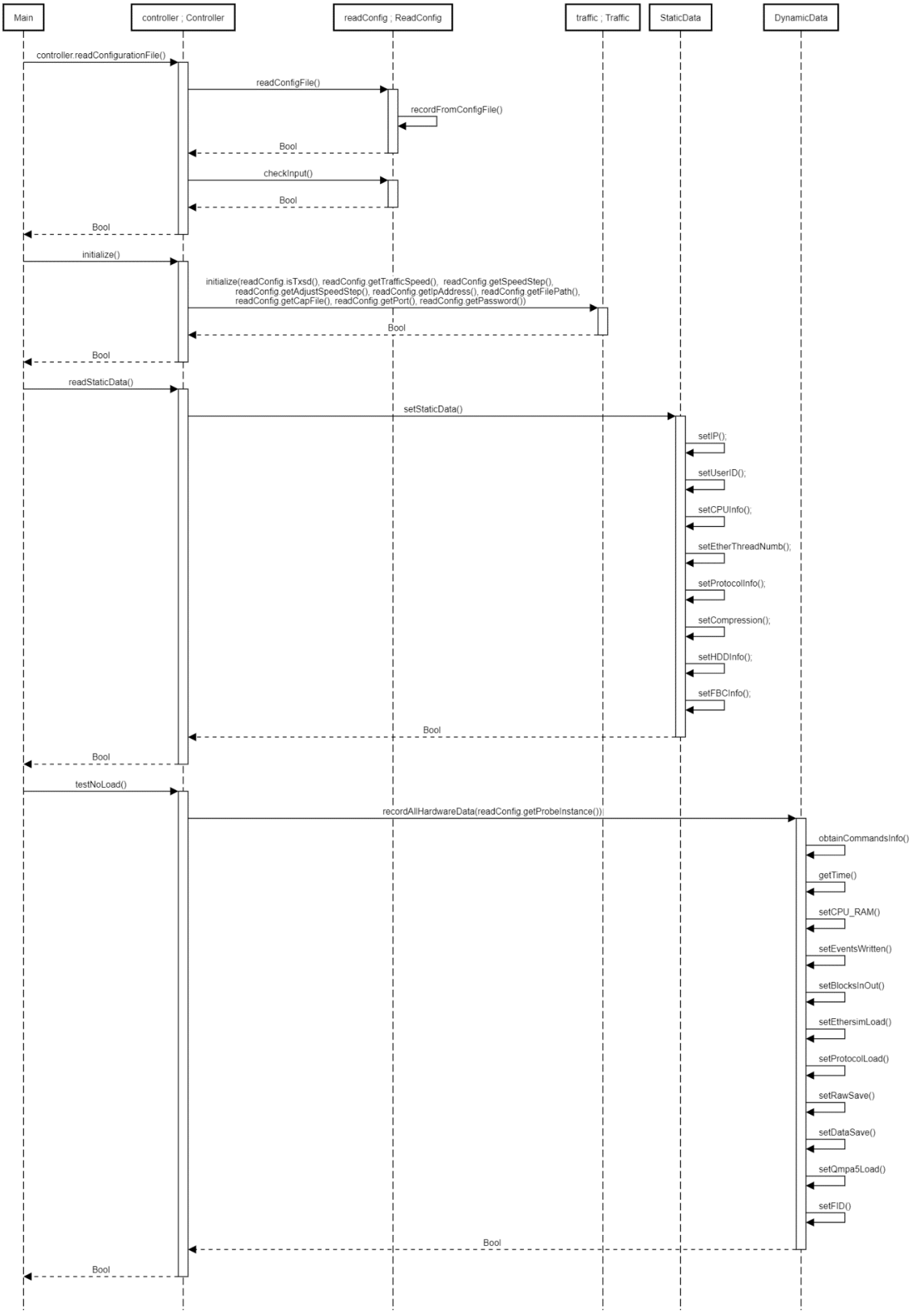


Figure 11 Sequence Diagram shows steps done prior to starting traffic

The figure above shows the sequence of steps taken by APT prior to starting traffic replay. As soon as the user starts the program, the main function will send commands to the controller. Prior to starting traffic, the configuration file is read and checked for its input. Traffic class is initialized with the data it requires to start traffic. Static data or information about the probe itself is read and dynamic data is read once to display idle load of the probe. StaticData and DynamicData namespaces write directly into Database class's variables.

3 Implementation

3.1 Generating .pcap file demonstration

The following commands are to demonstrate the creation of a .pcap file from scapy. To install scapy in Ubuntu, the command “apt install scapy” in the terminal will start the installation process. It is also advised to run scapy in sudo mode.

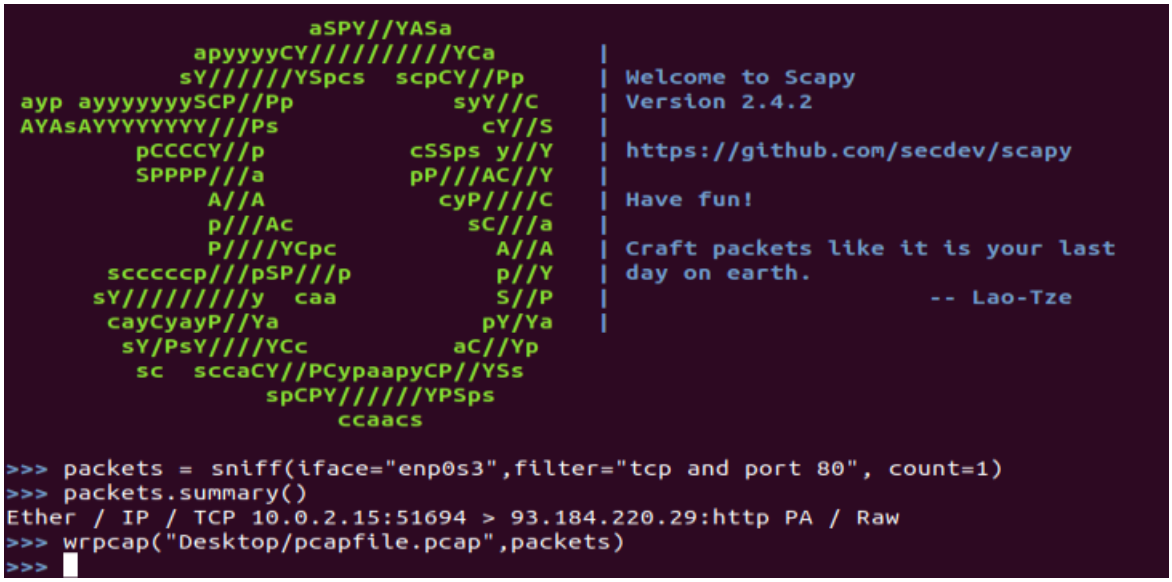


Figure 12 Sniffing traffic and writing out to .pcap

From the above example, a file pcapfile.pcap was generated from sniffing with a TCP and port 80 filter on network interface “enp0s3” which is default in virtual Ubuntu guest.

The resulting file can be opened with wireshark with the following summary. For this example count was set equal to 1, this is the number of packets to capture. It is possible to capture much larger files had the count been set to something higher or even capture all until user deems enough packets have been captured.

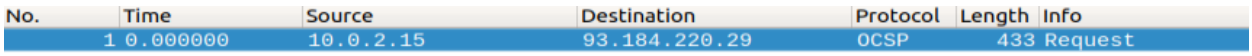


Figure 13 Generated .pcap file as seen by Wireshark

Now .pcap files can be generated from sniffing real traffic and these files can be passed straight on to a traffic generator to be replayed in a loop.

3.2 Traffic replay demonstration

The replay of traffic can be initiated on the traffic generator with either tcpreplay or TXSD command. In the following example, a replay of traffic is initiated on "Paul" server using tcpreplay.

```
Poul:/var/opt# ./tcpreplay -l 0 -M 500 -i dummy0 /var/opt/traffic/sgsn2_6_gtp99.cap
sending out dummy0
processing file: /var/opt/traffic/sgsn2_6_gtp99.cap
processing file: /var/opt/traffic/sgsn2_6_gtp99.cap
processing file: /var/opt/traffic/sgsn2_6_gtp99.cap
^C
Actual: 1747272 packets (748057291 bytes) sent in -1561027720.29 seconds.             Rated: -0.5 bps, -0.00 Mbps, -0.00 pps
Poul:/var/opt#
```

Figure 14 Tcpreplay demonstration

The -l option is the number of loops with 0 signifying an infinite amount i.e. keep looping until stopped by the user. -M is the speed at which to replay given traffic file in Mb/s. The -i option signifies the interface or server port. In this example, to not take up another server's time just for traffic generation, a virtual "dummy0" port is used and it means that the probe is its own traffic generator (this capability was one of the requirements). Whereas in a real scenario, the traffic generator would send out traffic to a port like "eth0" or ethernet zero because that would be the port with which it is connected to the probe. /var/opt/traffic/… is the location of the traffic file that will be replayed to the probe and the file that was generated with Scapy can be used here.

3.3 Configuration File

All the information needed to initiate traffic must be obtained from the configuration file that the user fills in. The goal is to make this process as easy as possible for the user. The simplest the configuration file can be is if it were a text file that can be opened and edited with any text editor. There the user should fill in data fields with appropriate data. The following image is of the configuration file opened in Notepad++.

```
Prerequisites:
Make sure that sshpass is installed!!!


Configuration:

TXSD = false
IP_Address = 172.28.70.178
Command_Path = /var/opt
Traffic_File = /var/opt/traffic/sgsn2_6_gtp99.cap
Use_Algorithm = 1

Algorithm_0 Finds optimal traffic load by incrementing traffic speed
Number_Of_Data_Readings = 3
Traffic_Speed = 700
Speed_Step = 100
Adjust_Speed_Step = 0.2

Algorithm_1 Records utilization of specified traffic loads.
Speeds_To_Test = 50 200 500 1200 2000 4000

Port = dummy0
Probe_Instance = 31
Time_To_Stability_Test = 10
Password = nettest
Acceptable_Data_Loss = 0
Generate_Probe_Info = false

Qapm5_Restart = false
```

Every field has a name and is then followed by an equal sign, which is followed by the information that the user is supposed to fill in. Since the user may choose which algorithm APT should use, there are fields for both. In a real use case, each field also has some short description to tell the user what exactly APT expects, however due to space constraints, this description is not shown here.

3.4 Starting traffic replay

To start this traffic in APT, the C++ code must somehow communicate with the Linux terminal. Previously it was mentioned that it is possible to do so through the system() function call. However, as soon as the function returns, the terminal it interacted with is closed. Therefore, connecting to a password protected server and starting traffic must be done with only one command. The command that is passed, is an sshpass command and this is what happens if the user will type in the command manually on the probe:

```
Poul:~/AutomaticPerformanceMeasurement# sshpass -p nettest ssh -o StrictHostKeyChecking=no
 root@172.28.70.178 'cd /var/opt && ./tcpreplay -l 0 -M 500 -i dummy0 /var/opt/traffic/sgs
n2_6_gtp99.cap'
*******************************************************************************
                            NOTICE TO USERS


This computer system is the private property of Anritsu, whether
individual, corporate or government.  It is for authorized use only.
Users (authorized or unauthorized) have no explicit or implicit
expectation of privacy.

Any or all uses of this system and all files on this system may be
intercepted, monitored, recorded, copied, audited, inspected, and
disclosed to your employer, to authorized site, government, and law
enforcement personnel, as well as authorized officials of government
agencies, both domestic and foreign.

By using this system, the user consents to such interception, monitoring,
recording, copying, auditing, inspection, and disclosure at the
discretion of such personnel or officials.  Unauthorized or improper use
of this system may result in civil and criminal penalties and
administrative or disciplinary action, as appropriate. By continuing to use
this system you indicate your awareness of and consent to these terms
and conditions of use. LOG OFF IMMEDIATELY if you do not agree to the
conditions stated in this warning.

*******************************************************************************
sending out dummy0
processing file: /var/opt/traffic/sgsn2_6_gtp99.cap
processing file: /var/opt/traffic/sgsn2_6_gtp99.cap
^C
Killed by signal 1.
```

Figure 15 SSH to traffic generator and start traffic in one command

Now it is possible to start APT on the probe and have it SSH to a server to start a replay of traffic from a generated .pcap file in one command. This again, will also work if the probe is its own traffic generator. However, as seen above where tcpreplay was running, the terminal was attached to this process and no more commands could be passed until traffic replay was interrupted. It would be possible to start this process in the background i.e. detach it from its controlling terminal, so the function would return instantly and the main thread in C++ can continue. But there needs to be constant control over that process. This is in case there is a need to stop traffic and start new traffic at different speed. Killing the process is possible but for that the process's ID needs to be remembered. The decision was made to just start a new thread in C++, as this way there is always

a thread attached to the process. If traffic speed needs to be stopped the thread can be cancelled, speed variable edited, and the thread started again. Before the thread starts any traffic, a command to "killall" either tcpreplay or txsd is ran. This is to make sure that there is no traffic running on the traffic generator.

To create a thread, POSIX pthreads were used and this is because of the compiler in our central compilation server, g++ (GCC) 4.4.7 20120313 is an older version. C++ 11 features such as *std::thread* are not supported with this version of the compiler.

A deadlock is when "*A set of processes is deadlocked if each process in the set is waiting for an event that only another process in the set can cause.*" [14] In APT no deadlock can occur because threads do not share resources. There are a total of two threads, the main thread and the thread that will ssh to the traffic generator to start traffic. The main thread waits for the traffic thread to start traffic and then it records utilization, etc. If the traffic thread returns without initiating traffic, then the main thread gives the user some sort of error and quits.

3.5 Taking readings

Once APT initializes traffic it is time to record utilization of the probe. To do so commands must be passed to the terminal and the output must be recorded in some way. The best way to do so, would be to have the output from the terminal be recorded in a vector that stores every word outputted by the terminal in some index. A word would be a string or character that is separated by spaces from another string or character.

The following function is responsible for sending commands to the terminal and recording their output as a string:

```
std::string getStdoutFromCommand(std::string &cmd) {
    /// This function pushes string 'cmd' as a terminal command  ///
    std::string data;
    FILE * stream;
    const int max_buffer = 256;
    char buffer[max_buffer];
    cmd.append(" 2>&1");

    stream = popen(cmd.c_str(), "r");
    if (stream) {
        while (!feof(stream))
            if (fgets(buffer, max_buffer, stream) != NULL) data.append(buffer);
        pclose(stream);
    }
    return data;
}
```

Then, using another function the string can be separated into words as mentioned above. These words can be stored in a vector of strings and this vector can be passed on to functions that will filter the vector to obtain some piece of data, such as qmpa5 load. Therefore, APT will pass on commands to the terminal to get some specific data and store the output of a specific command in a vector of strings/doubles or whatever is required for that data. The following snipped of code shows commands being passed to the terminal to set vectors of strings that will be filtered later.

```
database.setInspectCommand(getVectorStringOfCommand(
        "inspect" + probeInstance + " dr| grep -v dpdk |
        grep -o 'speed=.*\\|disk_write=.*\\| compress.*'"));

database.setTopCommand(getVectorStringOfCommand("top -b -n 2"));

database.setVmStatCommand(getVectorStringOfCommand("vmstat 1 2"));

database.setXLogFileCommand(getVectorStringOfCommand("x" + probeInstance +
    " logfile verbose"));

database.setChnsCommand(getVectorStringOfCommand("x" + probeInstance + " chns"));

database.setFreeCommand(getVectorStringOfCommand("free | grep Mem"));

database.setDateCommand(getVectorStringOfCommand("date"));
```

It is necessary to find CPU/RAM utilization of qmpa5, li-ether-sim and of protocols. Linux's top command provides a real time view on the running system. It shows lists of processes or threads that are currently being managed by the Linux Kernel. Top is capable of displaying CPU and RAM utilization by process ID or by process name and is present on all probes by default. It was decided to use the Top command not only to monitor qmpa5, li-ether-sim etc. But to also show the overall CPU and RAM utilization of the probe. That is because this data will already be present. However, since Top runs real time, parameters such as "-b -n 2" have to be passed. -b means to run in Batch mode which is useful for sending output from Top to programs. In this mode Top will run for a set amount of iterations and then quit so that APT can save the output and continue.

3.6 Filtering output

As already mentioned, filtering will have to be performed uniquely for each command to obtain the desired value. The following is a short example of filtering dynamic data and this function will be repeated several times to later calculate a trimmed average:

```
bool DynamicData::setEventsWritten(const std::string &probeInstance){
    std::vector<std::string> logFile = database.getXLogFileCommand();
    if(logFile.size() < 2 || logFile[0] == "-bash:"){
        std::cout << "Error getting 'x" + probeInstance + " logfile!'" << std::endl;
        ERROR = true;
    }
    for (unsigned int i = 0; i < logFile.size(); ++i) {
        if(logFile[i] == "Events" && logFile[i+1] == "Written"){
            database.mEventsWritten.push_back(getDoubleFromString(logFile[i + 3]));
        }
    }
    //Check if output has an acceptable size to avoid seg fault.
    return true;
}
```

Again, because of an older compiler, usual C++ 11 operations such as converting strings to double using stod are unavailable and therefore, functions that handle conversions of strings will have to be created. These functions are placed in the common file, this file will also hold the functions that pass commands to the terminal and save output.

One of the requirements was to display a trimmed average and therefore several readings at each traffic load are taken. The time at which these readings are taken is spread out evenly across the specified duration of the benchmark. A decision was taken to only record ten readings per benchmark of a specific traffic load. Then trim four of the values, two highest, two lowest and take the average of the remaining six values. The reason for this simplification is to reduce the number of inputs a user will have to make in the configuration file. It is one less thing to worry about and ten readings is a safe enough value. The testing team does not see the need to further increase or ever decrease this number. The trimmed and averaged values will be written into vectors of averaged values. Each index of this vector corresponds to a traffic load tested. The variables that were used to calculate this average will be reset so that they can be used for further testing.

For the algorithm which will test user defined traffic speeds APT will read and store these loads in a vector and will test from lowest load to highest. This is until the traffic speed runs into unacceptable data loss that was specified by the user. If the first tested speed is found to be unstable then the optimal speed will be displayed as 0, given that none of the listed traffic speeds were stable. The following function will perform this task.

```
bool Controller::userDefinedSpeedsAlgorithm(int &dataLoss, int
&dataLossPriorToPlayingTraffic) {
    std::vector<double> speedsToTest = readConfig.getSpeedsToTest();
    for (unsigned i = 0; i < speedsToTest.size(); ++i) {
        traffic.startTraffic(speedsToTest[i]);
        std::cout << "Current speed: " << traffic.getSpeed() <<
        database.dataMagnitude << std::endl;
        /*Check previous data loss*/
        dataLossPriorToPlayingTraffic = getDataLoss(database.getXPerfCommand());
        /* Call to a function, run in a loop looking for data loss.
        has defined in the configuration file. Also check for error.*/
        if (test_This_Speed(dataLoss, dataLossPriorToPlayingTraffic) == -1) {
            std::cout << "Error" << std::endl;
            return false;
        }

        if (dataLoss <= readConfig.getAcceptableDataLoss()) {
            std::cout << "Testing passed with acceptable data loss. Speed was: "
            << traffic.getSpeed()<< database.dataMagnitude << std::endl;
            DynamicData::setFID();
            DynamicData::setQmpa5Load();
            traffic.cancelTraffic();
            m_DataPassStorage.push_back(traffic.getSpeed());
            DynamicData::setAverage();
            tested_Speeds.push_back(traffic.getSpeed());
            database.dataLossPerTestedSpeed.push_back(dataLoss);
        }
        if (dataLoss > readConfig.getAcceptableDataLoss()) {
            if(i != 0){
                m_OptimalSpeed = speedsToTest[i-1];
            } else {
                m_OptimalSpeed = 0;
            }
            break;
        }
        if(i == speedsToTest.size() - 1){
            m_OptimalSpeed = speedsToTest[i];
        }
    }
  return true;
}
```

3.7 Error handling

Errors are reported via bool variables. Functions where critical errors can occur are Boolean. These functions will output an error message with what went wrong and return false. Potential errors include a utilization reading being unavailable, data in configuration file that makes no sense etc. Controller will check if the functions it calls return false, if they do, then the controller will return false as well. The main function calls several functions of the controller and if either of them return false then the program quits. This way the user is notified of where the error that occurred and APT quits in a safe manner.

4 Testing and Deployment

4.1 Manual testing

Testing was conducted throughout the development phase however, access to a variety of different probes was limited. This is because probes are constantly required by our probe testing team as they have to use these probes for their own tasks. Therefore, majority of testing was done on just two probes, the probe server “Paul” and probe server “G10”. These two probe servers are of different OS versions and are quite far apart in terms hardware capability.

Code was compiled on Anritu’s central compilation server and then APT was uploaded onto the probe server for testing. This had several advantages over simply testing functions for their ability to filter data and pass commands to the terminal.

- Probes runs on different OS’s to the development environment and therefore, commands such as top are different and need to be filtered differently.
- If the task is to test the functions which collect dynamic data, then there is no need to create different versions of testing data to get an average for one entire traffic load. And there is no need to create different versions of testing data for every traffic load.
- Two probe servers tested have different versions of probes running on them, therefore, APT gets to run and filter various versions of terminal commands.

4.2 Sanity Testing

A sanity test is a check to evaluate if the result can possibly be true and the test can be conducted rationally. This test was performed before giving the probe testing team access to APT in order to make sure that APT is not randomly crashing and stable enough for them to use it and conduct further testing. The calculated trimmed averages outputted in the CSV file by APT were also tested by manually checking if the values make sense and referencing them with what the utilization was like on the probe.

4.3 Usability testing

Several employees, who are part of the probe testing team began to try out APT. Initially they needed some guidance from the user manual but then were able to quickly learn how to use the configuration file and run APT to get the output they required. They were very quick to give further suggestions for improvement.

4.4 User testing

The testing of APT done by our team in a real environment proved essential as it found a few problems. This form of testing turned out to be the most thorough, because the testing team constantly changes their probe configurations to mimic those of our customers. Therefore, APT got to run on a far greater variety of probes.

As a result, during this testing stage several problems were found. A few probes had x commands which APT was unable to filter. This is because these versions commands were not presented or available during the development stage. Therefore, several functions have been expanded to account for varied versions of x commands. Furthermore, it turned out that during the implementation stage, newer versions of x commands and inspect commands were released and were not accounted for. These also had to be added and APT was accepted for use within the company.

Unfortunately, is no way to avoid having to expand APT every time a new version of a command is added. This is because so far there is no standard way in which data will be represented when using diagnostic commands on the probe. APT will have to be continuously updated whenever new versions of diagnostic commands are released. After development of APT, a suggestion was made to the team that writes diagnostic command to use a specific standard. This will make it possible to use one filter that should always function correctly despite newer versions.

4.5 Performance testing

Theoretically, APT has an impact on the probe as it takes up some CPU processing time for itself. Therefore, several trials were conducted, and the results of total CPU utilization/RAM utilization or found optimal traffic load when running APT versus not running APT, during traffic load, were found to be within margin of error. This may have been due to the fact that the tests were conducted on a modern probe server with 28 CPU cores which could mean that older probe servers with fewer cores may fare worse. APT will seriously only utilize one CPU core and then again, not fully as there are many sleep calls throughout APT. If APT were to have an impact, it would slightly reduce optimal traffic load figure but at the same time it would ensure further stability at that load. Possible solution to this problem is to modify APT to run locally as this would only require the probe server to display terminal commands but all the filtering, saving and calculating would be processed by the local computer. Given that APT is now constantly being expanded, this will be mentioned in the Future Work section.

4.6 Security

APT does not represent or add any security risks. This is because anyone who uses APT is already a company employee and are entrusted with handling probe servers. They are also well aware of the passwords required to SSH to these servers and even if the passwords are saved in the configuration file in open form, no one who is unauthorized will have access to the configuration file or APT itself.

5 Future Work

This chapter will discuss possible extensions and enhancements for the system created. There are two categories of possible enhancements, function and non-functional.

5.1 Functional extensions

- *Run APT from user's PC on Windows:* When APT was in the planning stage, company computers ran on Windows 7 which does not support SSH natively and the testing team would rather run APT on the probe directly. However, during development Windows 7 stopped receiving security updates and Anritsu decided to upgrade to Windows 10 which supports SSH natively. This means it is more convenient to extend APT to run form the user's computer and connect to probe servers via SSH. This would solve the possible issue of APT interfering with benchmarking results and the testing team likes the idea more now.
- *CPU Utilization per thread:* Knowing utilization of every CPU's thread would help in discovering bottlenecks that occur due to lack in parallelism of probe software.
- *Writing out CSV file whilst running the benchmark:* At this moment, if something were to go wrong during the execution of APT, or the user no longer wants APT to continue and stops APT's execution using 'ctrl c', all recorded data will be lost. Therefore, it would be convenient to have APT write out recorded data as it is executing.
- *Connect to traffic generator before taking readings:* APT could connect to traffic generator right after taking readings from the configuration file. This way APT can see whether or not the traffic generator is up and running prior to recording anything. This means that were something to go wrong, the user would be notified much more quickly than right now. Currently, APT spends some time taking idle readings before connecting to the traffic generator.

- *Algorithm to quickly find the optimal traffic load*: This problem can be thought of as finding an optimal speed (optimal traffic load) in an infinite sized array with binary search. It must have some user specified minimum where searching for a more optimal value is pointless. I.e. traffic load 1500Mb/s is stable, 1600Mb/s is not stable so do not continue if the difference is less than 100Mb/s. 1500Mb/s is optimal. The algorithm starts off by guessing a traffic load and then check if the load number does not cause data loss. If no data loss is found, then add to the previous speed some random value (within reason). If the value is too big then cut it by half and repeat. This algorithm will yield the same result as the iterative algorithm because in both algorithms the user will specify the minimum increment. The only advantage of this algorithm is it speeds up the process of finding the optimal load but at the cost of testing less traffic speeds. It was decided that this is not yet necessary but can be added as a feature to just find the optimal load without reading utilization readings.

5.2 Non-Functional extensions

- *GUI for configuring APT:* A GUI could be built to simplify working with the configuration file. In the GUI the user could simply enter configuration details into fields without the risk of corrupting or losing the configuration file. APT's updates during the benchmarking phase could also be formatted in a more readable manner as compared to now where they simply get printed to the terminal and are not easy to read.

6 Conclusion

The goal of the thesis was to write a program that can automate probe server testing for Anritsu's testing teams. All parts of the probe that could cause bottlenecks have been identified. Creation of .pcap files was demonstrated. The third part of the task was to replay traffic on a server from some traffic generator according to configuration set by the user. This part of the task was completed successfully, and APT can play traffic to a probe just like a human would be able to setup manually.

The next part was to record various measurements of the probe during traffic playback. At this moment APT records data on all the identified parts that could cause bottlenecks and this solution can be expanded. Lastly, the task was to find maximum traffic load without data loss or very limited data loss that would be set by the user. This goal was also fulfilled as APT takes small steps or iterations to find a traffic load just before data loss occurs.

There are however, things to improve. The code could be cleaner, and variables could be sorted in an alphabetic order to make reading and finding code easier. Some filtering algorithms are a bit too complex for the task and Linux's terminal commands such as grep could be used to simplify things for these functions. This means that the code could be optimized both for readability and performance, as the probe is already loaded with traffic and any additional traffic only adds to inaccuracies in detection of data loss.

Furthermore, testing during the development phase should have been performed on more probe servers with varied probe versions so that all the different types of terminal probe diagnosis commands could be collected. Unit tests could have been used on functions that filter static data.

The testing team finds APT useful and continues to use it because it drastically reduces time spent on benchmarking. As expected there are requests to add features as more people begin to weigh in with their feedback and overall, they like and utilize this solution.

7 Bibliography

[1] Anritsu, "Probe Installation and Configuration Guide version 8.0.1," 2019-04-10.

[2] CentOS, "About," CentOS, [Online]. Available: https://www.centos.org/about/. [Accessed 2019].

[3] J. D. Groot, "What is the General Data Protection Regulation? Understanding & Complying with GDPR Requirements in 2019," digitalguardian.com, 15 May 2019. [Online]. Available: https://digitalguardian.com/blog/what-gdpr-general-data-protection-regulation-understanding-and-complying-gdpr-data-protection. [Accessed 2019].

[4] WireShark, "SampleCaptures," WireShark, [Online]. Available: https://wiki.wireshark.org/SampleCaptures. [Accessed 2019].

[5] S. Ramachandran, "Use Case Analysis: Tutorial & Examples," 2018. [Online]. Available: https://study.com/academy/lesson/use-case-analysis-tutorial-examples.html.

[6] tutorialpoint, "UML - Activity Diagrams," Tutorials Point, 2019. [Online]. Available: https://www.tutorialspoint.com/uml/uml_activity_diagram.htm. [Accessed 2019].

[7] L. m. page, "sshpass(1) - Linux man page," linux.die.net, [Online]. Available: https://linux.die.net/man/1/sshpass. [Accessed 2019].

[8] H. Arora, "Linux tty Command Tutorial for Beginners (with Examples)," howtoforge.com, 27 Jun 2018. [Online]. Available: https://www.howtoforge.com/linux-tty-command/. [Accessed 2019].

[9] C. Hope, "How to create a CSV file," 13 11 2018. [Online]. Available: https://www.computerhope.com/issues/ch001356.htm. [Accessed 2019].

[10] Git, "About," git-scm.com, [Online]. Available: https://git-scm.com/about. [Accessed 2019].

[11] GitLab, "About GitLab," GitLab, [Online]. Available: https://about.gitlab.com/company/. [Accessed 2019].

[12] C. Perrin, "Use PuTTY as an SSH client on Windows," techrepublic.com, 7 March 2008. [Online]. Available: https://www.techrepublic.com/blog/it-security/use-putty-as-an-ssh-client-on-windows/. [Accessed 2019].

[13] SiteGround, "WinSCP Tutorial," siteground.com, [Online]. Available: www.siteground.com/tutorials/ssh/winscp. [Accessed 2019].

[14] J. Trdlička, "Deadlock," Department of Computer Systems, FIT, Czech Technical University in Prague, Operating Systems Lecture 6, 2018.

[15] M. J. M. S. C. R. S. C. M. B. Gordon Hogenson, "docs.microsoft.com," Microsoft, 08 30 2017. [Online]. Available: https://docs.microsoft.com/en-us/cpp/cpp/namespaces-cpp?view=vs-2019. [Accessed 2019].

[16] P. J. Fleming and J. J. Wallace, "How not to lie with statistics: the correct way to summarize benchmark results. Communications of the ACM.," 218–221. doi:10.1145/5666.5673. ISSN 0001-0782. , 1986-03-01.

Acronyms

APT – Automatic Performance Testing

GUI – Graphical User Interface

FBC – Fiberblaze cards

SSH – Secure Shell

OOP – Object Oriented Programming

CSV – Comma Separated Value(s)

PDU – Protocol Data Unit

LAN – Local Area Network

RAM – Random Access Memory

CPU – Central Processing Unit

TTY – TeleTYpewritter

GDPR – General Data Protection Regulation

HDD – Hard Disk Drive

Contents of enclosed CD

```
.
├── Documentation
│   ├── Programmers Guide PDF .pdf
│   └── User Guide PDF.pdf
├── exe ..................................................... Folder with the executable implementation
│   ├── APT
│   └── configFile.txt
├── readme.txt .............................................. Description of the contents
├── src
│   ├── Thesis.docx ......................................... Thesis
│   ├── demonstration ....................................... Demonstration of APT
│   └── impl ................................................ Source code of the implementation
└── text
    └── Thesis.pdf .......................................... Thesis in PDF
```