



# Train passenger information system load simulation

Ossi Nuutero

OPINNÄYTETYÖ  
Toukokuu 2019

Tieto- ja viestintäteknikka  
Tietoliikenne

## TIIVISTELMÄ

Tampereen ammattikorkeakoulu  
Tieto – ja viestintäteknikka  
Tietoliikenne

NUUTERO OSSI

Matkustajainformatiojärjestelmän kuormitustestaus

Opinnäytetyö 30 sivua, joista liitteitä 1 sivua  
Toukokuu 2019

---

Matkustajainformaatio – ja kuulutusjärjestelmä koostuu monista eri laitteista ja ohjelmistokomponenteista. Koska Testijärjestelmään ei voida kiinnittää kaikkia laitteita, joita oikeaan junaan kiinnitettäisiin, on osa laitteiden dataliikenteestä simuloitava. Ilman tätä on mahdoton tietää, pystyvätkö laitteet ja ohjelmistot suoriutumaat siitä kuormasta, joka niillä tulee oikeassa toimintaympäristössä olemaan.

Opinnäytetyössä simuloitiin dynaamisten reittinäyttöjen kuorma, sekä testattiin kuinka täynnä IP äänivahvistimen kaista voi olla, jotta kuulutuksia pystytään vielä tekemään ilman, että niihin syntyy merkittäviä tai havaittavia häiriöitä.

Kohdelaitteen resurssien käyttöä piti tutkia simuloinnin ja kuormituksen aikana. Resurssien käyttöä tutkittiin lähettämällä SSH yhteyden kautta kohdelaitteille komentorivikyselyitä. Kyselyistä saadut tulokset esitetään kuvaajissa

Dynaamisten reittinäyttöjen näkymä tulee järjestelmän keskusyksiköllä (TPC) olevalta web-palvelulta. Opinnäytetyössä tallennettiin yhden näytön web palvelulle lähettämät HTTP kyselyt ja sitten kyselyt simuloitiin Jmeter ohjelmalla lähetettäväksi saman aikaisesti niin monta kertaa, kuin näyttöjä olisi oikeassa toimintaympäristössä. Simuloinnin perusteella TPC suoriutui kuormituksesta hyvin, eikä sen resurssien käyttö juuri noussut simuloinnin aikana.

IP äänivahvistimien kaistan tukkiminen testattiin lähettämällä sille satunnaista binääri dataa UDP paketteina. Jos Paketteja lähetettiin rajoittamattomasti, alkoi automaattikuulutuksista hävitä osia ja manuaalikuulutuksissa oli paljon viivettä. Rajoittamalla pakettien lähettämistä oli mahdollista haarukoida raja, jonka alapuolella IP äänivahvistin pystyy vielä toimimaan ilman, että kuulutusten taso kärsii.

Jmeter ohjelmalla olisi myös mahdollista simuloida paljon erilaisia kyselyitä ja protokollia, esimerkiksi SQL kyselyt olisi mahdollista simuloida. Tähän ei kuitenkaan ollut tarvetta tässä projektissa.

## ABSTRACT

Tampereen ammattikorkeakoulu  
Tampere University of Applied Sciences  
Information and communication technology  
Telecommunications

NUUTERO OSSI

Train passenger information system load simulation

Bachelor's thesis 30 pages, appendices 1 pages

May 2019

---

Passenger information – and announcement system contains many different devices and software components. A normal test bench can't contain all the devices that would be attached in the real train environment. This is why it is important to be able to simulate the load of those devices that are not included in the test bench. Without load simulation it is impossible to know if the devices are capable to handling the load of a full system.

In this project the load of dynamic route maps was simulated and the maximum bandwidth use of the IP amplifier without any errors or suppressions to announcements was tested.

When simulating a load to the test system, it is important to observe the metrics of resource usage from the target devices. This was done by opening a SSH connection to devices and sending command-line queries to target devices. From the replies it was possible to draw different graphs about the using of resources.

Dynamic route map displays receive their visual information from the web service that is running on the TPC. At first the HTTP-requests send by one DRM display were recorded and then simulated with JMeter to be sent at the same time as many times as there are displays in the real train. After simulation test it was possible to come in conclusion that there were not any problems for TPC to handle the load of all displays in a real train.

IP amplifier bandwidth was tested by sending random binary UDP packages to the IP amplifiers. Without any limitations the announcements started to get suppressed and they had huge delays. With limiting the amount of UDP packages sent, it was possible to find the maximum point where the IP amplifier is still able to provide high quality announcements without any errors.

It is of course possible to simulate also other protocols and requests with JMeter, for example SQL queries but it was not necessary in this project.

---

Key words: load simulation, observing the metrics

## TABLE OF CONTENTS

1	INTRODUCTION .....	6
2	TRAIN PASSENGER INFORMATION SYSTEM .....	7
2.1	Display Sub-System.....	7
2.2	Announcement Sub-System .....	8
2.3	Telephony Sub-System .....	8
3	PROTOCOLS .....	9
3.1	TCP .....	9
3.2	HTTP.....	9
3.3	UDP .....	10
4	JMETER .....	11
4.1	Samplers.....	11
4.2	Listeners .....	12
5	COLLECTING METRICS .....	13
5.1	CPU usage.....	13
5.2	Memory consumption.....	14
5.3	Network usage .....	15
6	LOAD SIMULATION .....	17
6.1	Idle state .....	17
6.2	Simulating HTTP requests .....	19
6.3	Stressing the system with UDP data .....	21
6.4	UDP to IPAMP .....	25
7	DELIBERATION .....	28
	SOURCES .....	29
	APPENDICES.....	30
	Attachment 1. Train passenger information system network .....	30

## Abbreviations and acronyms

CIP	Common Industrial Protocol
DRM	Dynamic Route-Map
HTTP	Hyper Text Transfer Protocol
IP	Internet Protocol
IPAMP	IP Amplifier
LED	Light Emitting Diode
OCC	Operational Control Center
PACIS	Passenger Announcement, Communication and Information system
PCU	Passenger Communication Unit
PIS	Passenger Information System
RTP	Real-time Transmit Protocol
SIP	Session initiation protocol
TFT	Thin-Film Transistor (flat screen display)
TPC	Train PC
UDP	User Datagram Protocol
VoIP	Voice over IP

## 1 INTRODUCTION

In this thesis I investigate different ways to perform load tests to train passenger information system with open source JMeter load testing tool. The goal is to be able to simulate the load of a full information system at the test bench and measure its effects to the system performance and its devices. This is an important aspect of the testing and too often forgotten. The load has to be simulated because it is impossible to fit all the devices from the train to a test laboratory environment which contains multiple different test systems.

The functional part of this theses was implemented in Teleste Information Systems test-laboratory environment. All real usernames, passwords, IP addresses and customer names are withheld from this document for confidentiality reasons.

## 2 TRAIN PASSENGER INFORMATION SYSTEM

To understand the context and the environment where and what for this project was done it is required to be familiar with the basics how the train passenger information system works.

The content and functionality of the passenger information systems varies a lot between different projects and customers. The information system contains at least a few sub-systems. The test system that was used while developing JMeter tests contains display-, announcement- and telephony sub-systems. Each sub-system can contain multiple devices and software components. Communication between the devices is transmitted through the train industrial standard IP network. The network implementation contains DHCP, DNS, and NTP services.

### 2.1 Display Sub-System

The test system contains four different displays from which two are LED-displays and other two are TFT-displays. Both lateral- and frontal LED displays will be positioned outside of the train and both of them are presenting the last station name of the current driving mission. Presented content is provided by the train main computing unit; train PC (TPC).

TFT-displays will be positioned inside of the train. Wider TFT-displays (36.6 inch) are called DRM (Dynamic route map) displays and they are dynamically presenting route, next station, door opening side and other information. Smaller TFT-displays (13.3 inch) function as infotainment-displays. They present advertisements and useful information for passengers. The contents for all TFT-displays are provided by the TPC.

## **2.2 Announcement Sub-System**

Automatic and pre-defined announcement are played by the TPC. Automatic announcements are stored in the database and triggered automatically from the ground operational center through CIP protocol. Announcements are sent to IP amplifiers (IPAMP) via train network using VoIP SIP protocol and the announcements are carried out by the IPAMPs to the loudspeakers. It is also possible to make manual announcements using driver's microphone.

## **2.3 Telephony Sub-System**

The train contains Passenger communication units (PCU). PCUs can be used by passengers to communicate with a driver in case of an accident or an emergency. Driver answers the calls from the driver's cabin or ground operation control center (OCC). The system is informed which PCU made the call via CIP (Common industrial protocol) and the voice is transmitted using VoIP SIP protocol.



## **3 PROTOCOLS**

In this chapter the protocols used in the JMeter load testing will be shortly described.

### **3.1 TCP**

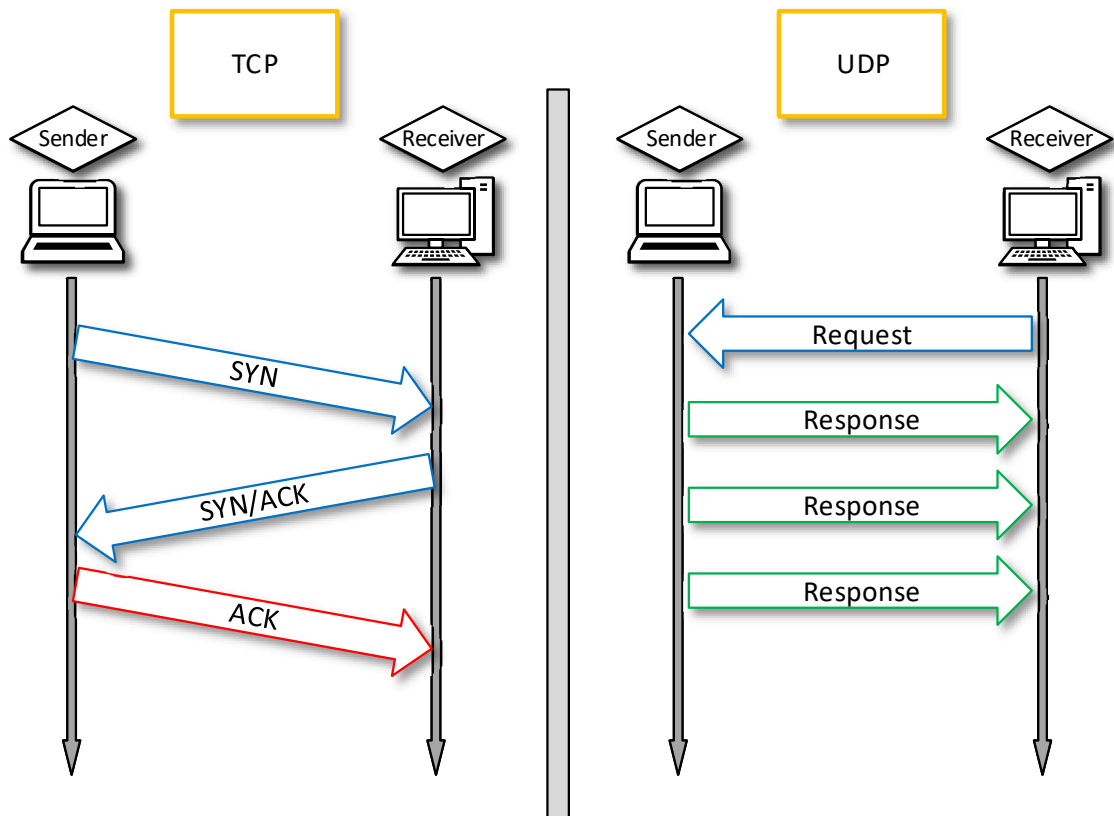
TCP (Transmission Control Protocol) is a very common internet protocol. It provides a reliable and error checked transmission between applications. When transferring data via TCP the connection has to be established first. To initiate a connection, the TCP uses a three way handshake process. First the device that wants to create a connection sends a SYN (synchronization) packet to target device. The target Answers with a SYN/ACK (acknowledgement) packet to tell that SYN packet was received. At last the first device send his own ACK packet to the target device to tell that SYN/ACK packet was received correctly. After this last packet is received by the target, transmission connection is established. Many protocols like HTTP, FTP and SSH are built on the TCP.

### **3.2 HTTP**

HTTP (Hyper Text Transfer protocol) is a commonly used protocol in a web environment. It is an application layer protocol and is used to transfer information between web server and a client. HTTP protocol supports nine different methods but it depends on the web server configuration which of them are applicable. The most commonly used methods are GET and HEAD requests. When using the Get method a client opens a TCP (Transmission control protocol) connection to the web server and requests specified information from the web server. The web server answer to the request with the information. Nowadays the HTTP is mostly used only in local web services. Public web services mostly use HTTPS (Hyper Text Transfer Protocol Secure) because the data transmitted through HTTP is not secured.

### 3.3 UDP

UDP (User Datagram Protocol) is a connectionless communication protocol and does not require a packed transfer confirmation from the receiver. The UDP functions on the transport layer. It is lighter protocol than TPC but is less reliable because the success of transmit is not confirmed or error checked. It is often used in real time systems and when the same data needs to be transferred to multiple targets for example streaming media or voice over IP applications. Also the DNS (Domain Name System) uses UDP.

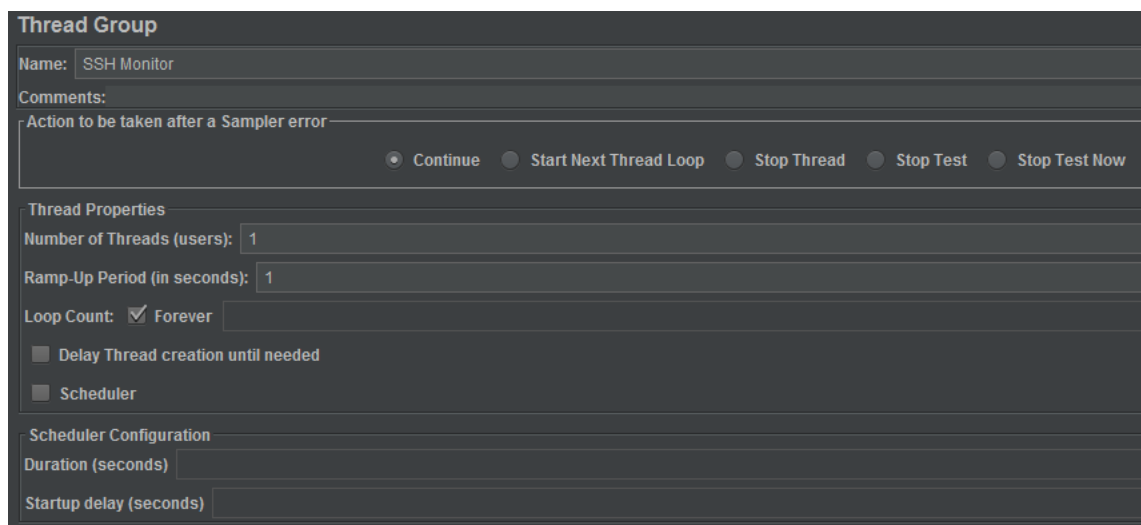


Picture 1 TCP vs. UDP

## 4 JMETER

Apache JMeter is an open source Java application. Originally it was created to test web applications but since it was born it has expanded a lot. JMeter works on a protocol level. It can be used to simulate the load of numerous protocols and also to measure the causations of the load tests. For example a test could be sending a HTTP request to a web page and then measure how long it took for web server to answer the request.

A basic way to run tests in JMeter is to create a test plan and a thread group. For better understanding the thread group can be thought like a one test. One test can be executed as many times and with as many users as wanted. Under a thread group a sampler can be added and results inspected with a listener. Of course there are also other elements that can be added to test like pre and post processors, config elements, timers and many others.

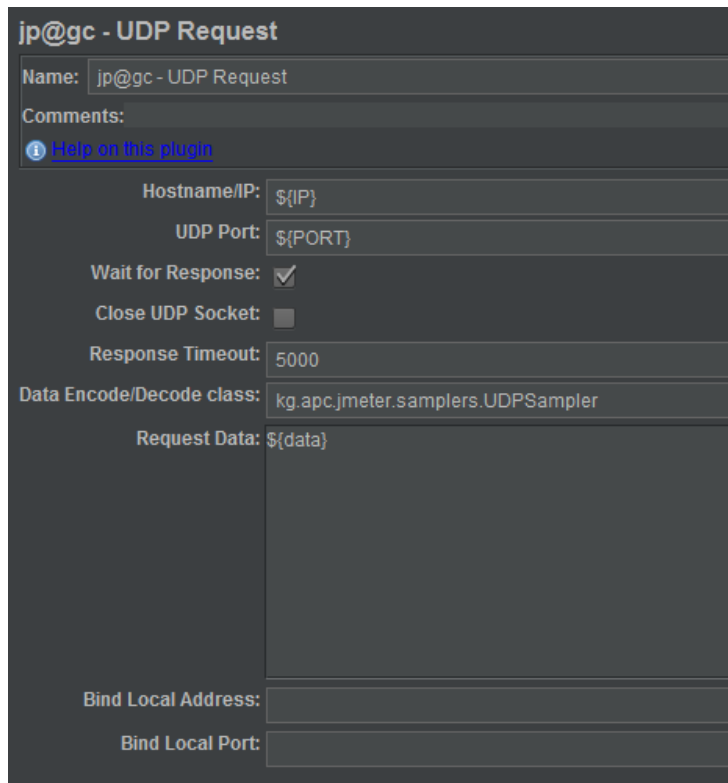


Picture 2 Thread Group

### 4.1 Samplers

Samplers are the ones that causes the load in the tests. There are many different sampler by default and many more can be easily added with a plugin manager. A sampler can be for example a UDP packet sender. The user defines the ad-

dress, port and the message to be send to the target system. Thread group defines how many times the UDP sampler is executed and how many of them are executed at the same time.



The image shows the configuration window for the 'UDP Request' sampler in JMeter. The window title is 'jp@gc - UDP Request'. It contains several fields and checkboxes:

- Name:** jp@gc - UDP Request
- Comments:** (empty)
- Hostname/IP:** \${IP}
- UDP Port:** \${PORT}
- Wait for Response:**
- Close UDP Socket:**
- Response Timeout:** 5000
- Data Encode/Decode class:** kg.apc.jmeter.samplers.UDPSampler
- Request Data:** \${data}
- Bind Local Address:** (empty)
- Bind Local Port:** (empty)

Picture 3 UDP Request Sampler

## 4.2 Listeners

Listeners are used to inspect the effects caused by samplers. With listeners it is possible to create different graphs and tables about the test results.

## 5 COLLECTING METRICS

Before executing tests there has to be a way figured out to inspect the causations of the load tests to the system. In this project the SSHMonitor was used to collect metrics from the target test system. SSHMonitor is not installed in the basic version of the JMeter. It has to be installed via Plugins Manager. The SSHMonitor was chosen because it does not require any agents or software to be installed or copied to the target system.

SSHMonitor is used to send command-line commands via SSH to the target system. It creates a SSH connection to the target system, executes a specified command and returns the value to JMeter. The commands executed in the target system have to return a numeric value because the JMeter uses these replies to draw a line graph.

SSHMonitor is not very user friendly. The user has to know exactly what values to inspect at the target system and how to get these values via command-line commands. In this project the target system was a Linux based so basic knowledge of Linux commands was required.

In this project the systems overall health needed to be inspected while performing load simulation tests. Following metrics were selected to be collected for inspecting the overall health of the system:

- CPU usage
- RAM consumption
- Network usage

### 5.1 CPU usage

For the CPU usage, three different values were collected from the system. 1min, 5min and 15min average usage. By default the Linux tracks the average CPU usage in the file “/proc/loadavg”.

```
1.78 1.17 0.83 3/746 30075
```

Picture 4 Loadavg

The file loadavg shows the CPU load as combined value for all the cores. As the load value of one core ranges between 0 -1, and in our system there are four cores in the CPU, the range of the load value is 0 – 4. So being in our case 0 equals 0% load and 4 equals 100% load to the CPU. To receive only the values by one SSH command the values have to be extracted from the file with following command:

```
cat /proc/loadavg | cut -d' ' -f1
```

Where “cat” opens the file “/proc/loadavg”, “cut -d' ’” defines the space to be a separator between the fields and “-f1” selects the column one. By using same command and only changing the number after “-f” all 1min, 5min and 15min CPU average loads can be collected.

## 5.2 Memory consumption

Here the word memory stands for a random access memory (RAM). In the Linux system information about memory consumption can be shown by writing “free” to the command-line.

```
root@cu:~# free
              total        used         free       shared    buffers     cached
Mem:          4020984      2415868      1605116       123756     215720     608032
-/+ buffers/cache:  1592116      2428868
Swap:           0             0             0
root@cu:~# █
```

Picture 5 free - command

From there the interesting columns are free memory and cached memory. Both are actually free memory but cached memory is “borrowed” by the hard drive disk for faster accessing and more responsive usage. The Linux system caches part of the memory that applications do not currently need. Cached memory will be freed if the applications run on the system need it.

To scrape the required information, in this case free memory, the following command is used:

```
free | grep Mem | awk '{print $4}'
```

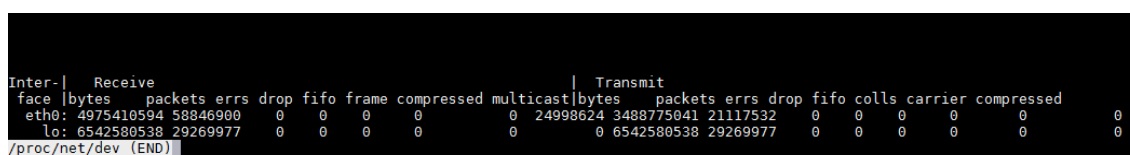
Where first “free” is opened, line where Mem is written is selected with “grep Mem” and from that line the column 4 is printed with “awk '{print \$4}'”.

Same goes for the cached memory only by changing column number to 7.

### 5.3 Network usage

When doing load simulation it is interesting to see how much network traffic the load causes at the exact moment. As for the default Linux do not calculate the network usage in or between specific time period, this was by far the most difficult to scrape with a one SSH command.

By default Linux logs how many bytes have been transmitted and received for each network interface since it was turned on. Linux system tracks that information in the file `/proc/net/dev`. From there it is possible to calculate the bytes going through.



```
Inter- | Receive | Transmit
face | bytes  packets errs drop fifo frame compressed multicast | bytes  packets errs drop fifo colls carrier compressed
eth0: 4975410594 58846900 0 0 0 0 0 0 24998624 3488775041 21117532 0 0 0 0 0 0
lo: 6542580538 29269977 0 0 0 0 0 0 0 6542580538 29269977 0 0 0 0 0 0
/proc/net/dev (END)
```

Picture 6 `/proc/net/dev`

At first a one value has to be scraped and saved to a variable, then wait a period of time and scrape the value again. Then calculate the subtraction of those values and divide it by the chosen time period. In order to complete the task in one line command, multiple commands had to be executed after another. Command separations in Linux command-line can be done with a semicolon (;).

```
i=$(cat /proc/net/dev |grep eth0 |awk '{print$10}'); sleep 5; a=$(cat /proc/net/dev |grep eth0 |awk '{print$10}'); echo "$((((($a - $i) / 5) / 1000))"
```

Where at first a variable "i" is created and a value from "/proc/net/dev" is scraped by selecting a line where is eth0 with "grep eth0" and from that line a column 10 with "awk '{print\$10}'". After that the command waits five seconds; "sleep 5". Then the command does the same thing again but this time saving the value to variable "a". After both "i" and "a" variables are saved the command prints the result of a final calculation. In the calculation the variables are first subtracted by each other  $(\$a - \$i)$  and the divided by 5. At last the result is divided by 1000 because it makes more sense to track transmitted kilobytes than bytes. As a result of this calculation, the last five second average of kilobytes transmitted in a one second is returned. Other averages could be easily calculated as well but in this case five second average is good enough.

For the received network traffic the command stays same except the values are scraped from a column 2.

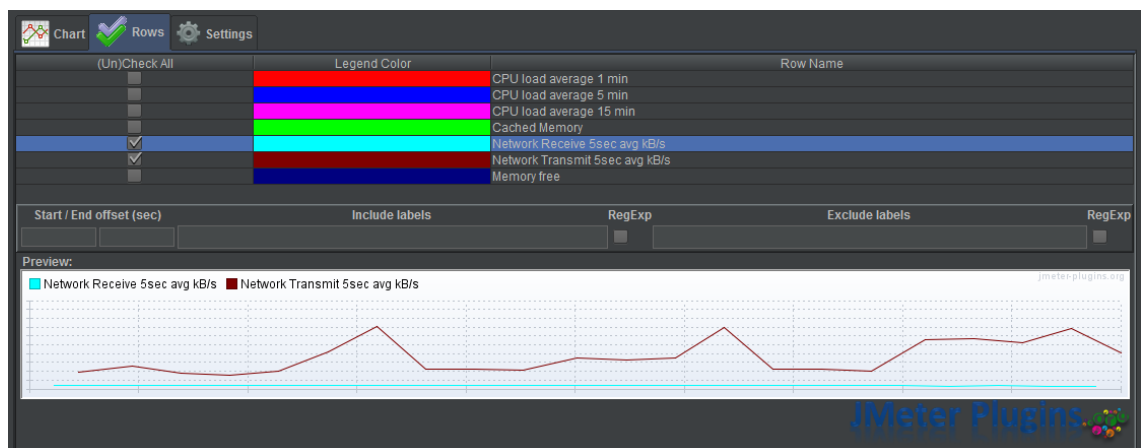


## 6 LOAD SIMULATION

This chapter presents how the actual load simulation and stress tests were done.

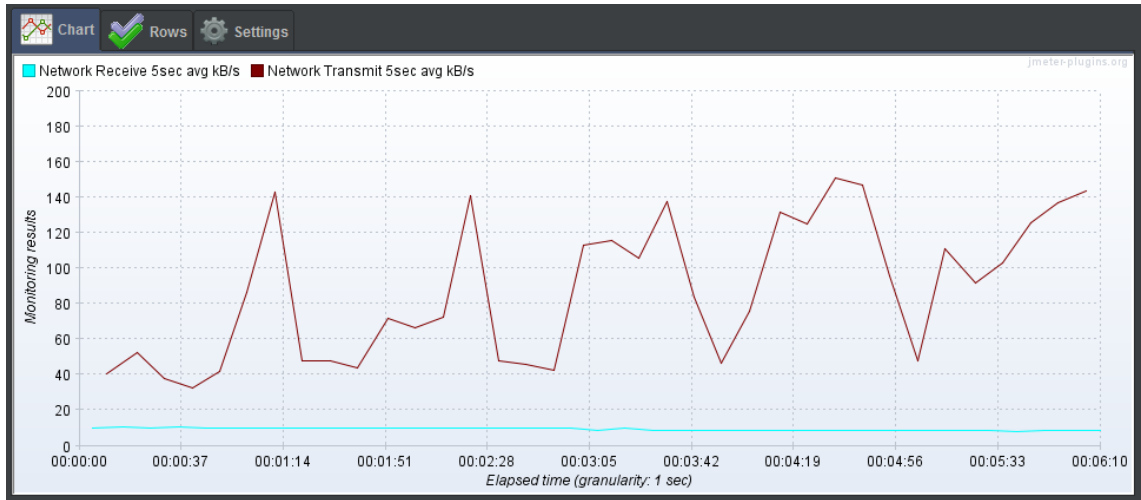
### 6.1 Idle state

Before initiating any kind of stress test to the system it has to be know how the system functions on an idle state so the stress test results can be compared to system default metrics. For collecting these metrics the system was left in the idle state on an off-route mode and everything else than SSHMonitor was disabled from the JMeter test plan. To have a better look at the specific metrics other lines can be hidden from the SSHMon graph.

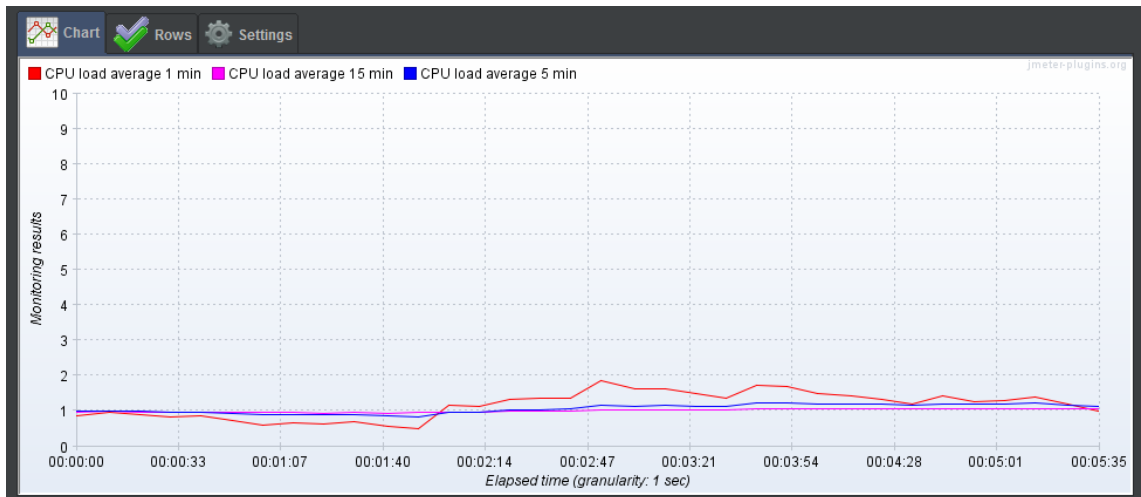


Picture 7 SSHMonitor rows

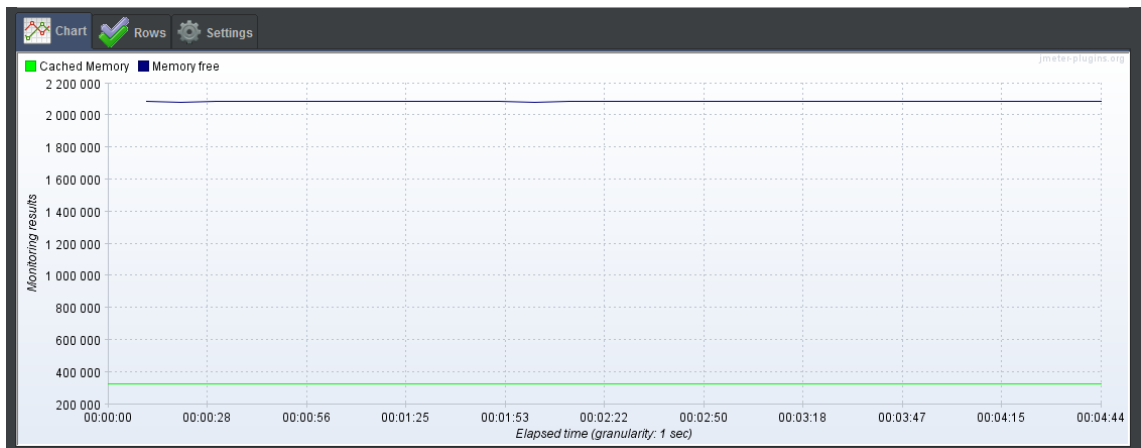
SSHMonitor graphs can be reviewed at the JMeter user interface or they can be saved as .csv file. Following screenshots will present the collected metrics when the system is in the idle state. Metrics show that there are no memory leaking (picture 9) and that CPU usage is quite low, about 25% of the maximum (picture 8). Received network traffic stays stable at 10kB/s and the transmitted network traffic varies between 30 and 150 kB/s (picture 7).



Picture 8 Network usage – idle state



Picture 9 CPU load average - idle state



Picture 10 Free and cached memory - idle state

## 6.2 Simulating HTTP requests

Since the JMeter was originally created for web application testing, was the HTTP load simulation the most convenient way to start. In JMeter the HTTP load can be caused through HTTP request sampler. It can be made manually by creating a HTTP request with a GET method to request a single web site but more realistic way is to record a browser while it is requesting the web site. By setting a localhost with a port defined for JMeter as a proxy to a browser, JMeter can automatically record the HTTP requests that the browser is sending. Those requests recorded with the browser can be re-used in the JMeter test plan.

The image shows the 'HTTP Request' configuration window in JMeter. The 'Name' field is set to 'HTTP Request'. Under the 'Web Server' tab, the 'Protocol [http]' is 'http', 'Server Name or IP' is '\${ip}', and 'Port Number' is '3020'. In the 'HTTP Request' section, the 'Method' is set to 'GET' (highlighted with a blue box), 'Path' is empty, and 'Content encoding' is empty. There are several checkboxes: 'Redirect Automatically' (unchecked), 'Follow Redirects' (checked), 'Use KeepAlive' (checked), 'Use multipart/form-data for POST' (unchecked), and 'Browser-compatible headers' (unchecked). At the bottom, the 'Parameters' tab is selected, showing a table for 'Send Parameters With the Request' with columns 'Name', 'Value', 'Encode?', and 'Include Equals?'.

Picture 11 HTTP request

Web server can be given as a domain name or an IP address. Path defines what content will be requested under given web server. If it is left blank, the index page will be requested.

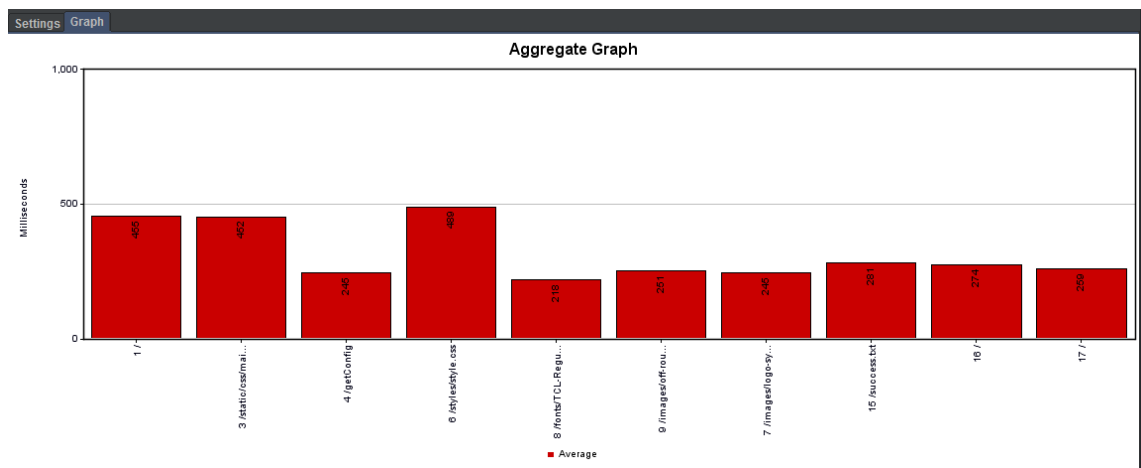
In the project this test was executed, there will be 12 TFT-displays requesting a page from one web server. That for the number of thread groups was set to 12. HTTP requests for the test plan were created by recording the requests made with a browser. To inspect the amount of data and time that different HTTP request required, an aggregate graph listener was added to the test plan. When the HTTP request test plan was ready, the train information system was set to on-route state with a driving simulator and then the JMeter test build was started.

From the aggregate listeners results it can be seen that server's average re-  
sponding time is quite long, over 400 milliseconds for some requests. Still re-  
gardless of the response time the functionality stayed good and there were no  
errors in any requests.

Label	# Samples	Average	Median	90% Line	95% Line	99% Line	Min	Max	Error %	Throughput	Received KB/sec	Sent KB/sec
1 /	653	454	453	532	561	643	54	766	0.00%	3.8/sec	2.24	1.29
3 /static/css/main.56bce2df.css	650	451	447	525	557	689	50	846	0.00%	3.8/sec	2.61	1.24
4 /getConfig	649	245	245	294	314	359	30	447	0.00%	3.8/sec	0.87	1.24
6 /styles/style.css	646	486	482	571	597	668	85	787	0.00%	3.8/sec	14.83	1.19
8 /fonts/TCL-Regular.ott	645	217	216	264	289	343	41	384	0.00%	3.8/sec	157.52	1.57
9 /images/off-route.png	645	251	247	305	337	401	61	508	0.00%	3.8/sec	562.48	1.21
7 /images/logo-syrial.png	644	244	240	292	309	380	84	478	0.00%	3.8/sec	28.84	1.22
15 /success.txt	642	279	273	332	351	456	173	856	0.00%	3.8/sec	1.41	1.17
16 /	642	274	268	324	341	399	188	847	0.00%	3.8/sec	2.90	1.70
17 /	641	257	253	306	324	451	158	548	0.00%	3.8/sec	2.91	1.70
TOTAL	6457	316	272	488	522	601	30	856	0.00%	37.7/sec	774.28	13.50

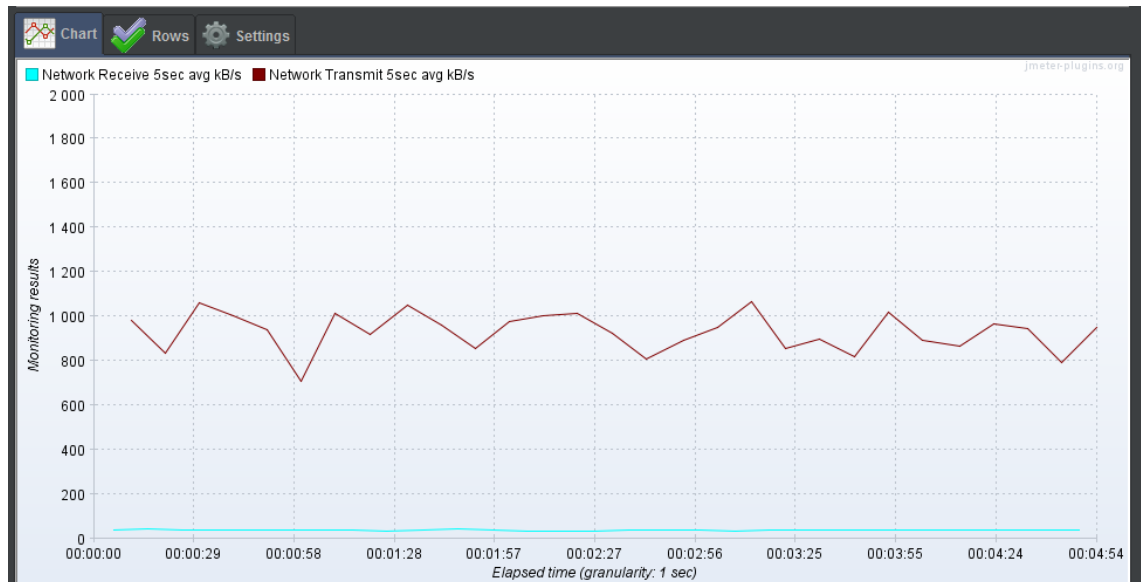
Picture 12 Aggregate graph listener table

Visual presentation of the request response times:



Picture 13 Aggregate graph listener - visual presentation

When inspecting the metrics of the target device with the SSHMonitor, the only  
big difference in the idle state happened in the network transmission. Other dif-  
ferences were only marginal. The conclusion is that the web server is not using  
too much resources even in the real train environment where all the displays are  
connected to the system.



Picture 14 Network usage while HTTP request simulation

### 6.3 Stressing the system with UDP data

UDP does not require any kind of session initiation between the sender and the receiver. It can be used to send additional traffic to target system and then inspect how much data the system can handle before its purposed functionalities start to be affected. In JMeter this can be done by using a UDP request sampler. It could be used to send also real data and set to wait for responses from the target system but in this project the UDP request sampler was simply used to send random binaries to system to fill its bandwidth.

Hostname/IP:	\${IP}
UDP Port:	\${PORT}
Wait for Response:	<input type="checkbox"/>
Close UDP Socket:	<input type="checkbox"/>
Response Timeout:	1000
Data Encode/Decode class:	
Request Data:	0101010
Bind Local Address:	
Bind Local Port:	

Picture 15 UDP request sampler

**Hostname:** IP or a hostname of a target system where the UDP packets will be sent.

**UDP Port:** Port of the target system that will be used.

**Wait for Response:** If this is selected the sampler will wait for responses from the target system and give an error if the response is not received.

**Close UDP Socket:** If this is selected the connection socket will be closed and a new socket will be opened for each sample.

**Response Timeout:** This does not matter if Wait for Response bullet is not selected.

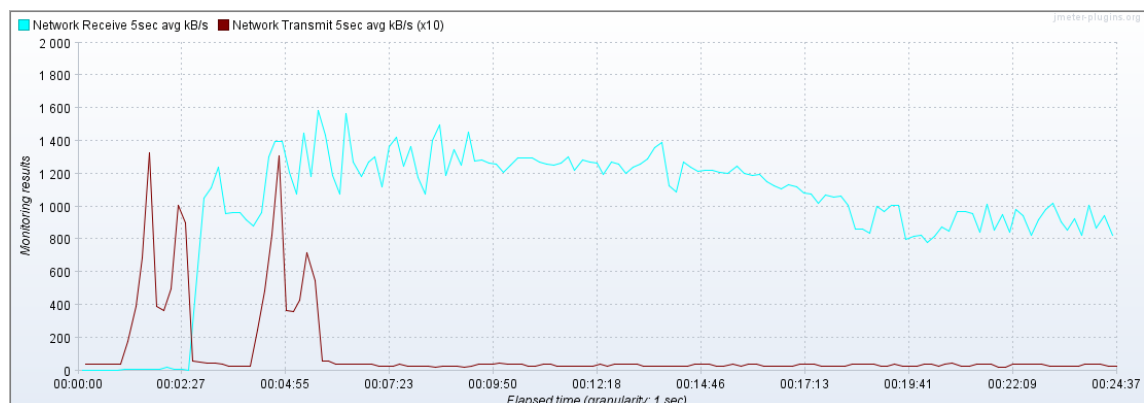
**Data Encode/Decode class:** Java class that will encode/decode the data sent to a target system. For example if request data is given in a plain text, the encode/decode class can be used to translate the data to binaries.

**Request Data:** Data can be given in any format depending which kind of encode/decode class is used. Data can also be read from a file.

**Bind local address:** Sender's local area network IP

**Bind local port:** If a specific port is wanted to be used to send out the UDP traffic it is given here.

At first the TPC was tested how it handles the UDP disturbance without any other stressing at the same time. In the train information system the TPC has to be able to handle a lot of data at the same time because it handles most of operations in the system. In the next graph it is shown how the network metrics look like while one thread of UDP binaries were sent to the TPC. The amount of the UDP data was not limited in this point so on thread sent as many samples as it could. The amount of samples sent is also dependent about the host PC that is sending the samples.



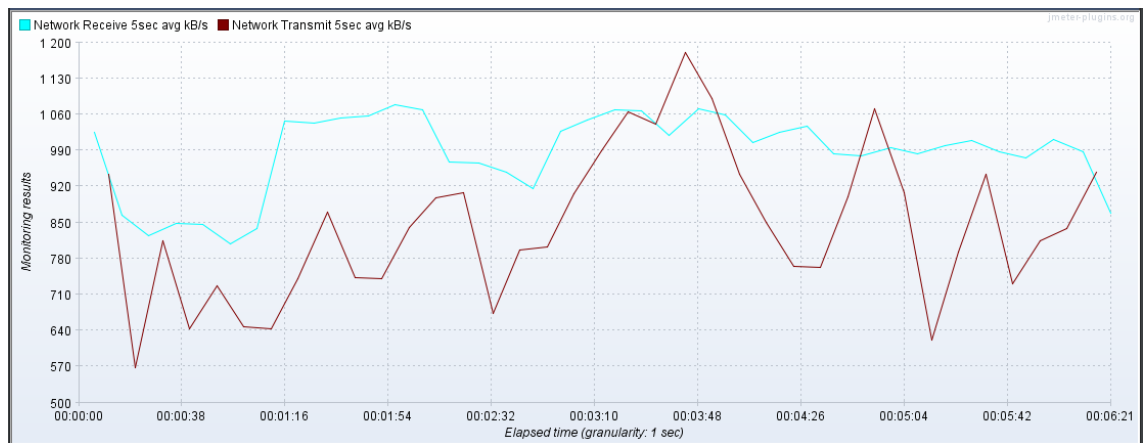
Picture 16 UDP samples to the TPC

In the graph the network transmit line makes two spikes in the beginning. Those spikes represents driving a short mission. First one is without the UDP disturbance and the second one with the UDP stressing. Network received- line shown clearly when the UDP sampler was set on. Driving a mission went perfectly fine even when the UDP sampling was on and any weird misbehaviors were not observed. Network received- line decreases gradually when the time goes further. This happens because the host PC can't keep up the amount of samples sent. Also the free memory was gradually decreasing but in the opposite hand the cached memory which is also "free" memory was gradually increasing. In the graph the change seems bigger than it really is. In the 20 minute test the variation range was about 60 megabytes which is about 3% of the free memory.

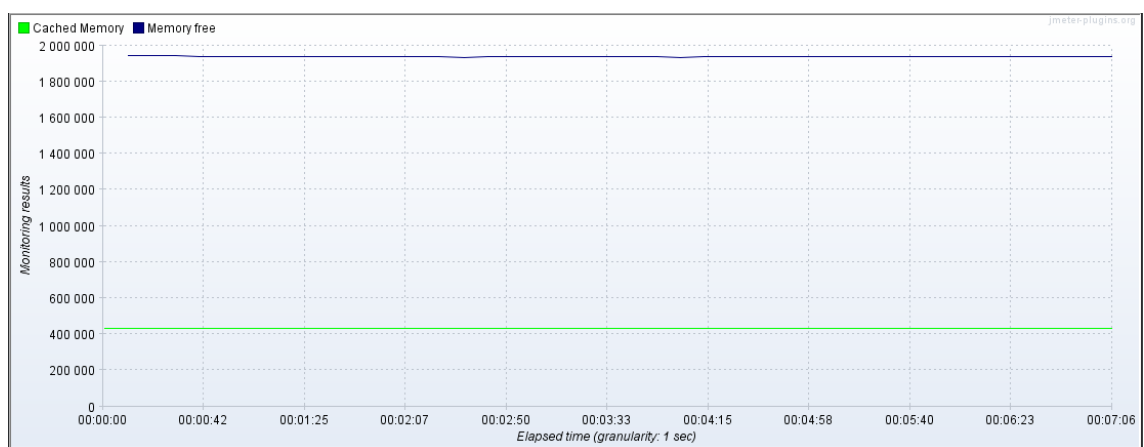


Picture 17 TPC free memory while receiving UDP data

Next it was time to add the full train http request simulation to the game. In this test the TPC had to be able to handle high amount data to both ways. Both HTTP – request test and the UDP tests were enabled from the test plan, test system was set to drive a mission and the test plan was set on. While driving a mission all automatic announcements, route information and other parts of the system were still functioning correctly and any weird behaviors were not observed. Also CPU usage and free/cached memory stayed stable during the short test.



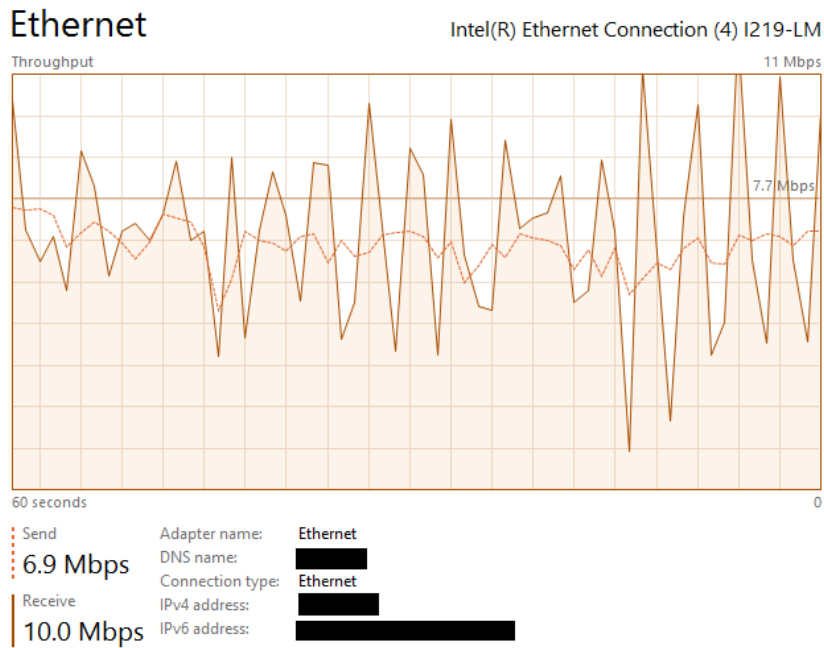
Picture 18 Network usage during HTTP+UDP test



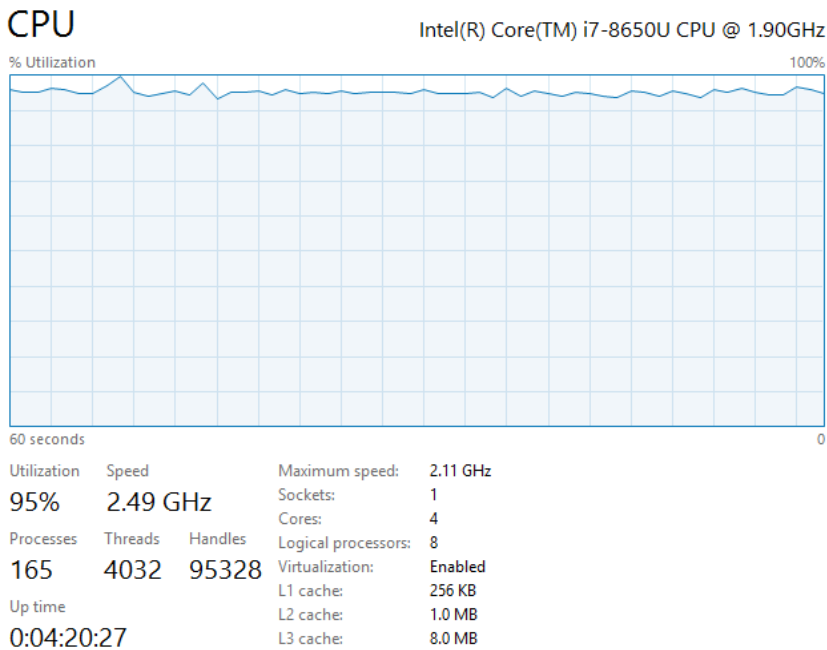
Picture 19 Free and Cached Memory during HTTP+UDP test

It is possible that in this test the limitations of the JMeter graphical version were reached. The host PC reached almost 100% CPU usage during the test. In the following graphs it can be seen how much data the host PC transmitted and received and how high CPU usage was during the test.





Picture 20 JMeter host PC received/transmitted



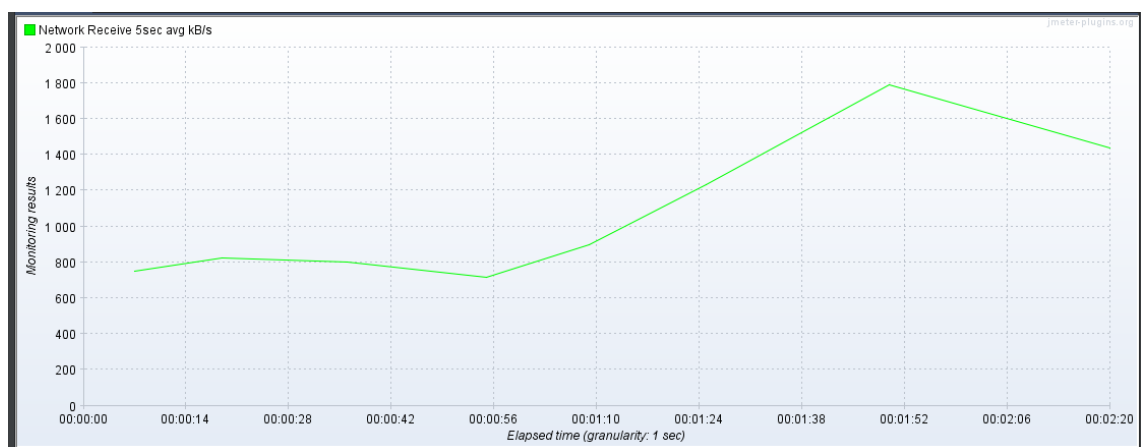
Picture 21 JMeter host PC CPU usage

### 6.4 UDP to IPAMP

The TPC is not an only device in the train information system that has to be able to deal with high amounts of data. IPAMP is responsible of streaming the audio to loud speaker. If there is a problem in the IPAMP the train passengers may not

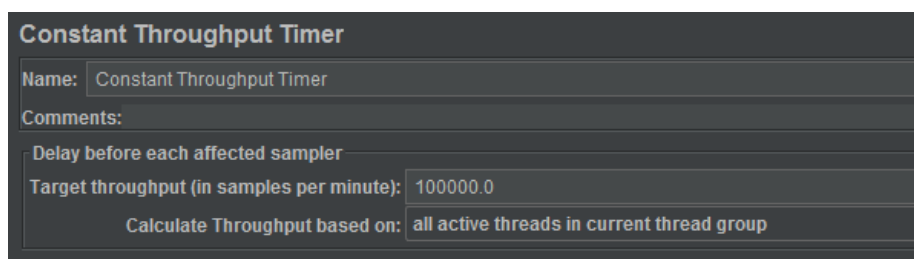
receive important information announcements regarding to their journey. IPAMP uses SIP (Session Initiation Protocol) protocol and RTP (Real-time Transport Protocol) protocols for emergency- and cabin to cabin calls and UDP for announcements to stream the audio. Since it is not possible to send RTP data with JMeter the load to IPAMP bandwidth was simulated as random binary UDP.

At first the train was set on a mission and then the UDP sampler was set on with one unlimited thread. When the UDP sampler was set on, the automatic route announcements started to become suppressed from the end and the manual public announcement had a huge delay. Also the frequency of responses received by SSHMonitor decreased so the metrics collecting became unreliable.



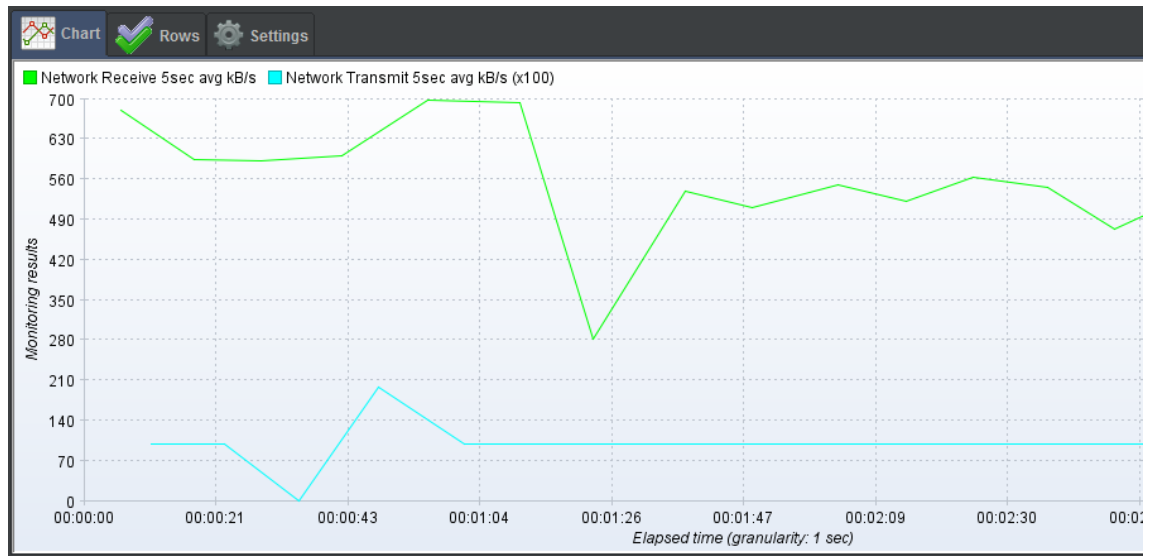
Picture 22 IPAMP UDP received

At this point it was clear that IPAMP can't handle the same amount of data than the TPC can. Of course this is understandable, IPAMP has less powerful hardware and is does not have to be able to handle the same amount of data. To get something out from the UDP testing with IPAMP, some limitations needed to be added to the test. Constant throughput timer was a choice to go and it was added to test plan.



Picture 23 Constant Throughput Timer

The constant throughput timer allows the threads in the test plan to send only specific amount of samples per minute. With limited amount of samples it was possible to clarify the bandwidth boundaries of IPAMP when it is still capable of streaming high quality audio without any suppressions or delays. Next graph shows the network usage when there were first 500k samples per minute sent and then the amount was decreased to 400k samples per minute.



Picture 24 Samples 500k to 400k

While sending 500k samples per minute there could be still a little noise observed but when the samples were reduced to 400k samples per minute, the quality of announcements was very good.

## 7 DELIBERATION

This thesis presents only a scratch what it is possible to do, test and simulate with the JMeter. The field of performance testing is wide and the JMeter is not an only tool that would the trick. However it is one of the best that is free to use and widely documented. It gets regular software updates and new plugins for use appear all the time. But still it is not very user friendly and in the varying environment it takes some time to configure and implement the test plans to new or other current projects.

There results from this project are good and quite reliable but to be even more reliable the tests should have been let running for longer times. Maybe some excel graphs could have been added as attachments for better visualization. Metrics collecting was a huge success as far as the devices were able to answer the SSH queries. Graphs got from the SSH monitor are clear and reliable.

For more intense stress testing the graphical user interface of the JMeter should not be used. Running the Java program requires quite a lot resources from the host computer. Decentralized stress testing is also possible with JMeter and maybe the best and only option for real stress testing with JMeter.

When this thesis project was planned, also simulating of JMS (Java Message Service) messages was included. This turned out not to be so simple because of Teleste's in-house railcomm service that differs a lot from a regular JMS service. This could have been a whole new topic for another thesis.

To encapsulate the topic, I would say that JMeter will be great help in the future to observe the overall functionality and metrics of our passenger information systems with or without added load or stressing.

## SOURCES

What is cached memory?

<https://www.linuxatemyram.com/>

SSH Monitor

<https://github.com/tilln/jmeter-sshmon>

How to use JMeter

<https://jmeter.apache.org/>

UDP request

[https://jmeter-plugins.org/wiki/dns\\_test\\_using\\_jmeter/](https://jmeter-plugins.org/wiki/dns_test_using_jmeter/)

UDP and TCP protocols

<https://www.howtogeek.com/190014/htg-explains-what-is-the-difference-between-tcp-and-udp/>

What is HTTP?

[https://www.w3schools.com/whatis/whatis\\_http.asp](https://www.w3schools.com/whatis/whatis_http.asp)

Linux commands

<https://www.geeksforgeeks.org/linux-commands/>

**APPENDICES**

## Attachment 1. Train passenger information system network

