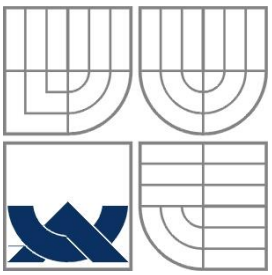
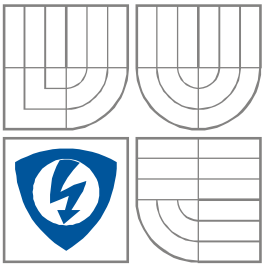


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ
ÚSTAV RADIOELEKTRONIKY

FACULTY OF ELECTRICAL ENGINEERING AND COMMUNICATION
DEPARTMENT OF RADIO ELECTRONICS

MODEL OF HTTP ADAPTIVE VIDEO STREAMING

Model adaptivního streamování videa přes HTTP

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE — Bc. DMYTRO KASIANENKO
AUTHOR

VEDOUCÍ PRÁCE — Ing. MARTIN SLANINA, Ph.D.
SUPERVISOR

BRNO, 2016

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA ELEKTROTECHNIKY A KOMUNIKAČNÍCH TECHNOLOGIÍ

Diplomová práce

magisterský navazující studijní obor **Elektronika a sdělovací technika**
Ústav radioelektroniky

Student: Bc. Dmytro Kasianenko ***ID:*** 171471
Ročník: 2 ***Akademický rok:*** 2015/16

NÁZEV TÉMATU:

Model adaptivního streamování videa přes HTTP

POKYNY PRO VYPRACOVÁNÍ:

Seznamte se s existujícími technikami pro adaptivní přenos videa s využitm HTTP protokolu. Vypracujte rešerši s důrazem na možnosti praktické implementace. Prostudujte koncepci simulátoru NS3 pro simulaci komunikačních systémů. V simulátoru simulujte libovolnou jednoduchou komunikaci s využitím HTTP protokolu.

Implementujte vybranou techniku přenosu videa tak, aby mohla být s využitím simulátoru NS3 simulována bezdrátová síť. Konrétní typ a nastavení sítě budou určeny vedoucím.

DOPORUČENÁ LITERATURA:

[1] LEDERER, S, MUELLER, C., TIMMERER, C. Dynamic adaptive streaming over HTTP dataset. In Proceedings of the ACM Multimedia Systems Conference 2012. Chapel Hill (North Carolina, USA), 2012.

[2] MULLER, C., TIMMERER, C. A VLC media player plugin enabling dynamic adaptive streaming over HTTP. In Proceedings of the ACM Multimedia 2011. Scottsdale (Arizona, USA), 2011.

Termín zadání: 8.2.2016 ***Termín odevzdání:*** 19.5.2016

Vedoucí práce: Ing. Martin Slanina, Ph.D.
Konzultant diplomové práce:

doc. Ing. Tomáš Kratochvíl, Ph.D., *předseda oborové rady*

ABSTRAKT

Klíčové úkoly této práce jsou vytvořit simulaci bezdrátové sítě v ns-3 a modelovat streamování videa přes tuto síť s použitím technologie MPEG-DASH. Práce popisuje nástroje a metody, využitelné pro kódování videa v souladu se standardem MPEG-DASH. V práci je vysvětleno, jaké nástroje jsou nutné pro spuštění video serveru, který používá MPEG-DASH technologii streamování videa. Pak je popsáno, jak vytvořit simulaci drátových a bezdrátových WiFi a LTE sítí. Je také popsáno spuštění video serveru a provázání se simulací, je demonstrováno, jaké uzly mohou být použity a jaká musí být vybrána vhodná délka segmentu.

KLÍČOVÁ SLOVA

Simulace, ns-3, MPEG-DASH, streamování, video, WiFi, LTE.

ABSTRACT

Key tasks of this work are to create the simulation of a wireless network in ns-3 and to stream a video over this network by using MPEG-DASH technology. It describes the tools and methods how to encode a video according to MPEG-DASH. It is explained what is needed to run video server, which uses MPEG-DASH technology to stream videos. Then, it is described how a simulation of wired and wireless WiFi and LTE networks is created. The simulation starts a server and shows how it works, how many nodes can be used and chooses suitable segment duration.

KEYWORDS

Simulation, ns-3, MPEG-DASH, streaming, video, WiFi, LTE.

KASIANENKO, D. *Model of HTTP adaptive video streaming*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, Ústav radioelektroniky, 2016. 50 s., 1 s. příloh. Diplomová práce. Vedoucí práce: Ing. Martin Slanina, Ph.D.

PROHLÁŠENÍ

Prohlašuji, že svou diplomovou práci na téma Model of http adaptive video streaming jsem vypracoval samostatně pod vedením diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce.

Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této diplomové práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a/nebo majetkových a jsem si plně vědom následků porušení ustanovení § 11 a následujících zákona č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon), ve znění pozdějších předpisů, včetně možných trestněprávních důsledků vyplývajících z ustanovení části druhé, hlavy VI. díl 4 Trestního zákoníku č. 40/2009 Sb.

V Brně dne ....

(podpis autora)

PODĚKOVÁNÍ

Děkuji vedoucímu diplomové práce Ing. Martinu Slaninovi, Ph.D. za účinnou metodickou, pedagogickou a odbornou pomoc a další cenné rady při zpracování mé diplomové práce.

V Brně dne ....

(podpis autora)

Faculty of Electrical Engineering
and Communication

Brno University of Technology
Technicka 12, CZ-61600 Brno, Czech Republic

http://www.six.feec.vutbr.cz

Experimentální část této diplomové práce byla realizována na výzkumné infrastruktuře vybudované v rámci projektu CZ.1.05/2.1.00/03.0072
Centrum senzorických, informačních a komunikačních systémů (SIX)
operačního programu Výzkum a vývoj pro inovace.

CONTENTS

LIST OF FIGURES

LIST OF TABLES

PREFACE

Today customers require a good video quality everywhere and anytime. With the growth of technologies for mobile high-speed internet access, companies like Apple and Microsoft created their own HTTP adaptive streaming technologies. Later on, the DASH Industry Forum created an open adaptive streaming standard MPEG-DASH, which consists a lot of benefits.

The main task of this work is to create a model of HTTP adaptive video streaming. MPEG-DASH was chosen as the best alternative. Network for this model is simulated in ns-3 and two virtual machines (VMs), created as Linux containers LXC, acting as client and server are used. At the server side, the virtual machine is running the *nginx* HTTP server, as well as the installation of all video encoding tools (x254, GPAC). After encode the video content and running the video server, the virtual machine with its simulated client is able to access the server, acquire the video and analyze it.

The first chapter of the thesis describes what software tools are required to put together a relevant toolchain and why, it shows how to encode video content and how to configure a HTTP server. The following three chapters describe how to create a simulation of a wired, WiFi and LTE network. Then, we analyze the adaptive streaming itself, show how the network requirements to provide appropriate quality of a video. The next chapter compares different length of DASH-segments. The last chapter shows how video adapts for different wireless network conditions.

1 INTRODUCTION

HTTP video streaming is a combination of simultaneous downloading and playing. A client gets video through HTTP, then saves in application's buffer. When enough data is downloaded (not whole video should be downloaded), video can be played from a buffer. Because data transmits using TCP, a client gets an uninterrupted copy of the video.

HTTP adaptive streaming (HAS) is based on classic HTTP video streaming, but can change a quality of a video to adapt it to the network conditions. To realize adaptation, the client measures network conditions and requests an appropriate quality of the media. On a server side, video divided into small segments with different representation/bitrate and each segment can be downloaded.

1.1 MPEG-DASH

Dynamic Adaptive Streaming over HTTP (DASH) [2] technology was developed under MPEG in cooperation with big companies like Microsoft, Adobe, Samsung, Qualcomm, Sony and many others. Standard is based on Adaptive HTTP streaming (AHS) and HAS, it is free, open, and made to work everywhere.

MPEG-DASH takes all benefits of previously created streaming systems Adobe Systems HTTP Dynamic Streaming, Apple Inc. HTTP Live Streaming (HLS) and Microsoft Smooth Streaming and add a lot of new, like HTML5 supports, agnostic to audio and video codecs, client failover and much more [3]. Moreover, DASH Industry Forum (IF) create the recommendations, guideline and software that everyone can use and easy implement.

According to DASH technology [4], a multimedia file splits into segments with different representations and stored on the server or the segments can be accessed via byte-range requests. In relation to available bandwidth and resources, client requests some quality level. If conditions become better client request for higher quality and vice versa, if conditions deteriorate (loaded network, distancing from the access point) it requests for worse quality. In this way, client gets varying quality of the media, which adapts to current conditions and it increases the experience of using such service.

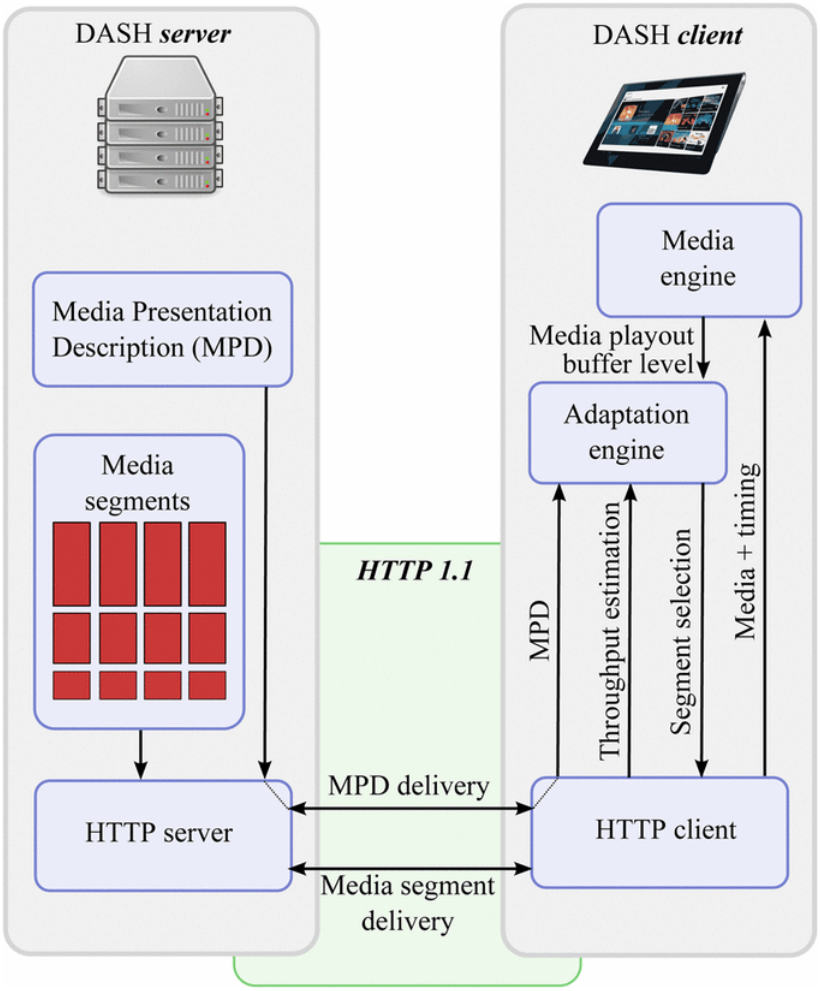


Figure 1.1 Principles of MPEG-DASH [4].

In Media Presentation Description (MPD) [5] provides information for the client about streaming service and segments. The server transmits the MPD file and media segments in different quality to the client. The client requests media segments in different representations based on information from MPD and available throughput and buffer level.

1.2 ns-3

The ns-3 [6] is a discrete-event network simulator created for research and education purposes. ns-3 is free software licensed under the GNU GPLv2. It carries well-documented core and elements, which are easy to use and debug.

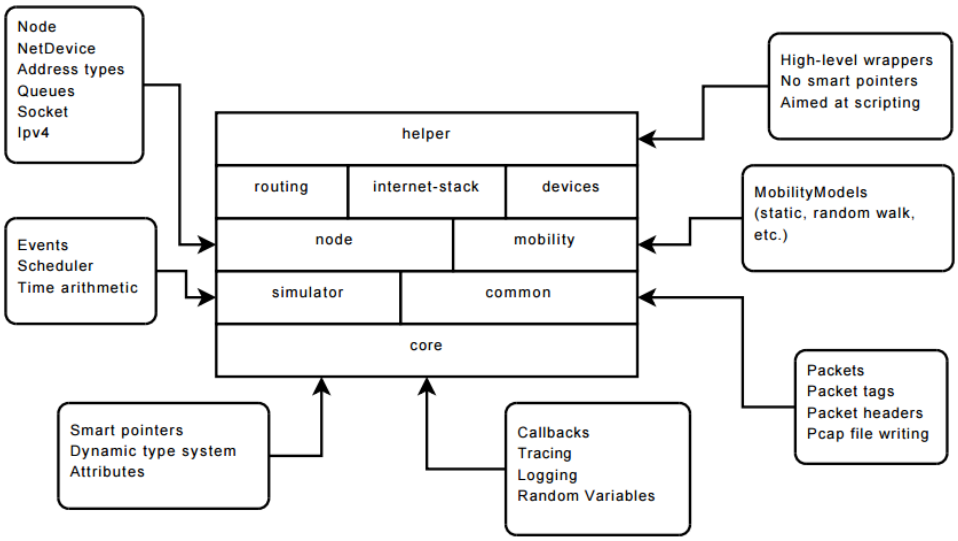


Figure 1.2 ns-3 modules [7].

In addition, the important goal of the project is to develop sufficiently realistic models, which allows a simulated network to interact in real time with real networks. Because it uses Linux networking architecture, like sockets and net devices, and ns-3 packets are stored in special packet buffers, packets can be easily sent on a real network interface.

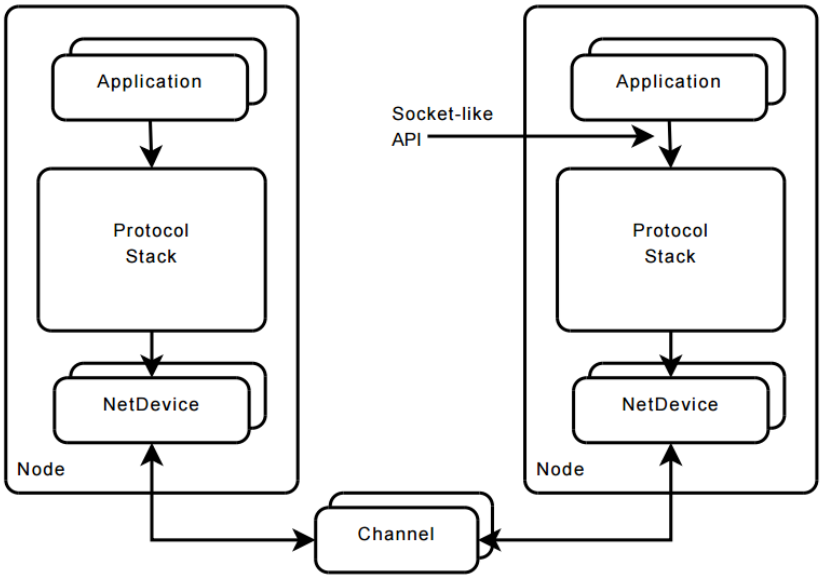


Figure 1.3 Basic model of ns-3 [7].

ns-3 focuses on IP and non-IP networks, however, common use is to create wireless network simulation, i.e. Wi-Fi, WiMAX, or LTE.

ns-3 is a C++ library which provides a set of network simulation models implemented as C++ objects and wrapped through Python [8]. In this work we use C++ to write a script, which sets a simulation using ns-3 libraries. Python can be used as an alternative. Any action can be scheduled in concrete time by use of a callback function.

2 PREPARATION

In this work is used the operation system Linux x86_64 (Linux Mint, based on Debian), because it is free and ns-3 and some of required software does not work or works incorrectly in other systems. For client and server the Linux Containers (LXC) [9] are used. It uses the same Linux kernel as the host machine but create an isolated Linux system. The biggest benefit of the containers is that it require much less resources than a full virtual machine. In the containers HTTP server and required software can be installed and connected to a simulated network through bridges.

At first, it is necessary to create two containers. It can be done using the following pattern: `lxc-create -n <container's name> -n <template name> -f <configuration file>`:

We will use Ubuntu OS on containers, because it has good repositories, so we shouldn't install some of required software form source code. With parameters, the required commands can be as follows:

```
$ sudo lxc-create -n left -t ubuntu -f src/tap-bridge/examples/lxc-
left.conf
$ sudo lxc-create -n right -t ubuntu -f src/tap-bridge/examples/lxc-
right.conf
```

Make sure that we have created containers:

```
$ sudo lxc-ls
```

We should see the containers just created:

```
$ left right
```

2.1 Install required dependencies

It is necessary to install required software before create the model, otherwise it will not work. All software used in this work is free and open.

Before starting the VMs, it is needed to run the following commands one by one [1] to add bridges and tunnels.

Firstly, add bridges to transmit packets in and out of the containers:

```
$ sudo brctl addbr br-left
$ sudo brctl addbr br-right
```

Then, create tap devices that will be used to get packets from the bridges into its process:

```
$ sudo tunctl -t tap-left
$ sudo tunctl -t tap-right
```

Set IP addresses of the tap devices and turn them on:

```
$ sudo ifconfig tap-left 0.0.0.0 promisc up
$ sudo ifconfig tap-right 0.0.0.0 promisc up
```

Add the tap devices to bridges and bring up:

```
$ sudo brctl addif br-left tap-left
$ sudo ifconfig br-left up
$ sudo brctl addif br-right tap-right
$ sudo ifconfig br-right up
```

Disable Ethernet filtering. If not, only STP and ARP traffic will be allowed to flow across bridges:

```
$ sudo mount -t cgroup cgroup /cgroup
$ cd /proc/sys/net/bridge
# sudo -s
# for f in bridge-nf-*; do echo 0 > $f; done;
# exit
$ cd -
```

We need to install many programs, so it is better to have internet access. Otherwise, software can be installed by copying source codes or binary packages to VM folder and installing manually.

In directory `/var/lib/lxc/left` in file *config* need to change `lxc.network.link = br-left` to `lxc.network.link = lxcbr0`. *lxcbr0* is automatically created bridge that allows to access internet form containers.

Then, start the left container, default login and password are “ubuntu”:

```
$ sudo lxc-start –n left
```

First, we need to install the recent version (later than 4.6) of the GCC compiler [10]. To do this, new repository should be added first:

```
$ sudo add-apt-repository ppa:ubuntu-toolchain-r/test
$ sudo apt-get update
```

Then, GCC and G++ can be installed using:

```
& sudo apt-get install gcc-4.9
$ sudo apt-get install g++-4.9
```

Also PCRE Library (Perl 5 Compatible Regular Expression Library) [11]:

```
$ sudo apt-get install libpcre3 libpcre3-dev
```

And OpenSSL library [12] required for HTTP server:

```
$ sudo apt-get install libssl-dev
```

Install *Checkinstall* [13], we will use it instead of *make install*. It makes easier installation and uninstallation of software form source codes:

```
$ sudo apt-get install checkinstall
```

2.1.1 HTTP server

The role of the HTTP server is to send client the MPD file and requested segments.

There are more than 30 web servers available today. The most popular: Apache HTTP Server, Barracuda Web Server, lighttpd, nginx, Oracle HTTP Server and many more. Some of them are free, some proprietary, each has its own benefits.

For HTTP based streaming there is no complicated logic needed on the server side. So, almost any of these servers are suitable.

We have selected to use the *nginx* [14] (engine x) - a HTTP and reverse proxy server, a mail proxy server, and a generic TCP proxy server. We will use it as a server for Dash.js player, which will be installed later.

Get *nginx*:

```
wget http://nginx.org/download/nginx-1.9.12.tar.gz
```

Unpack it:

```
$ tar xzvf nginx-1.9.12.tar.gz
```

And install:

```
$ cd nginx-1.9.12
$ ./configure
$ make
$ sudo checkintall
```

Now we have nginx installed.

2.1.2 Multimedia player

DASH-IF created its own player named Dash.js [15]. It is an open source and free MSE/EME player that also functions as the reference client for DASH-IF. We need it to play dash-encoded video streaming over HTTP.

For this player cURL [16], Node.js [17] and Grunt [18] are needed.

To install cURL, the following command can be used:

```
$ sudo apt-get install curl
```

Install Node.js:

```
$ curl -sL https://deb.nodesource.com/setup | sudo bash -
$ sudo apt-get install -y nodejs
```

Install grunt:

```
$ npm install -g grunt-cli
$ npm install grunt-template- jasmine-istanbul –save-dev
```

Install some other dependencies needed for running Dash.js player:

```
$ npm install grunt-contrib-connect grunt-contrib-watch grunt-contrib-
jshint grunt-contrib-uglify
```

And now get Dash.js player:

```
$ wget https://github.com/Dash-Industry-
Forum/dash.js/archive/development.zip
```

And unzip it:

```
$ unzip development.zip
```

Now we have the Dash.js player.

2.2 Encoding

Several segmenters and packagers are presents: eDASH Packager, Bento4, Rebaca MPEG DASH Segmenter. But we will use GPAC [19], an Open Source multimedia framework, a "Swiss knife" in work with multimedia. It provides a multimedia player Osmo4/MP4Client (which can be used to play encoded and segmented videos), a multimedia packager MP4Box (which we need) and some server tools included in MP4Box and MP42TS applications

Get the GPAC:

```
$ wget https://github.com/gpac/gpac/archive/master.zip
```

Unzip it:

```
$ unzip master.zip
```

Get the dependencies:

```
$ sudo apt-get install make pkg-config g++ zlib1g-dev libfreetype6-dev
libjpeg62-dev libpng12-dev libopenjpeg-dev libmad0-dev libfaad-dev libogg-
dev libvorbis-dev libtheora-dev liba52-0.7.4-dev libavcodec-dev
libavformat-dev libavutil-dev libswscale-dev libxv-dev x11proto-video-dev
libgl1-mesa-dev x11proto-gl-dev linux-sound-base libxvidcore-dev libssl-dev
libjack-dev libasound2-dev libpulse-dev libsdl1.2-dev dvb-apps libavcodec-
extra-53 libavdevice-dev libmozjs185-dev
```

Then install GPAC:

```
$ cd gpac-master
$ ./comfigure
$ make
$ sudo checkinstall
```

DASH-IF has the Guidelines for Implementation where they recommend to use AVC/H.264 video codec [20]. x264 [21], a free software library and application for encoding video streams into the H.264/MPEG-4 AVC compression format, proved itself as the fastest encoder, so we will use it.

Install x264:

```
$ sudo apt-get install x264
```

Now we are going to encode some example video *Example.mkv* in H.264/AVC [22] with 3 representations 2400, 1200 and 600 kbps and separate them into 4s long segments. Following command will create a 2400 kbps raw stream:

```
$ sudo x264 --output intermediate_2400k.264 --fps 24 --preset slow 1--
bitrate 2400 --vbv-maxrate 4800 --vbv-bufsize 9600 --min-keyint 96 --keyint
96 --scenecut 0 --no-scenecut --pass 1 --video-filter
"resize:width=1280,height=720" Example.mkv
```

The parameters are in the Table 2.1.

Table 2.1 The parameters of H.264/AVC encoding [22].

--output intermediate_2400k.264	The output filename. It creates a raw H.264/AVC stream.
--fps 24	Specifies the frames per second 24 in this case.
--preset slow	Tell x264 if it should try to be fast/slow to enhance compression.
--bitrate 2400	Target bitrate in kbps.
--vbv-maxrate 4800	Max local Video Buffering Verifier (VBV) bitrate. The maximum possible bitrate in the video. Better to set 2x *--bitrate*.
--vbv-bufsize 9600	Size of VBV buffer. Better to set 2x *--vbv-maxrate*.
--keyint 96	The keyframe interval, the maximum distance between I-frames. Because we will split the video into segments, at the beginning of each segment should be an I-frame. `Keyint` must be segment length in seconds multiplied with the frame rate. For 4 seconds segment: 4 sec. * 24 frames/seconds = 96 frames.
--min-keyint 96	Sets the minimum interval between I-frames. We need a constant segment length, so minimal keyframe interval must be the same as *–keyint*.
--no-scenecut	Disables scenecut.
--pass 1	Uses 1 pass, where writes the stats file only. Can be set to 2 to improve quality, but takes a long time.
--video-filter "resize:width=1280,height=720"	Change the resolution.
Example.mkv	Input video file.

Then, encodes two videos with bitsreams 1200 kbps and 600 kbps. It differs in *--bitrate*, *--vbv-maxrate* and *--vbv-bufsize*. For 1200 kbps:

```
$ sudo x264 --output intermediate_1200k.264 --fps 24 --preset slow --
bitrate 1200 --vbv-maxrate 2400 --vbv-bufsize 4800 --min-keyint 96 --keyint
96 --scenecut 0 --no-scenecut --pass 1 --video-filter
"resize:width=1280,height=720" Example.mkv
```

For 600 kbps:

```
$ sudo x264 --output intermediate_600k.264 --fps 24 --preset slow --bitrate
600 --vbv-maxrate 1200 --vbv-bufsize 2400 --min-keyint 96 --keyint 96 --
scenecut 0 --no-scenecut --pass 1 --video-filter
"resize:width=1280,height=720" Example.mkv
```

Now we add the previously created h264 raw videos to an *.mp4* container:

```
$ sudo MP4Box -add intermediate_2400k.264 -fps 24 output_2400k.mp4
$ sudo MP4Box -add intermediate_1200k.264 -fps 24 output_1200k.mp4$ sudo
MP4Box -add intermediate_600k.264 -fps 24 output_600k.mp4
```

The parameters are in the Table 2.2:

Table 2.2 The parameters for *.mp4* container [22].

intermediate_2400k.264	Input file we want to add into *.mp4* container.
-fps	The framerate. It must match the framerate used in the x264 encoding.
output_2400k.mp4	The output filename.

Create MPD-file and 4-seconds long segments:

```
$ sudo MP4Box -dash 4000 -frag 4000 -rap -segment-name segment_%s -url-
template -out dash.mpd output_2400k.mp4 output_1200k.mp4 output_600k.mp4
```

The parameters are in the Table 2.3:

Table 2.3 The parameters for segmentation [22].

-dash 4000	The segments duration in *ms*.
-frag 4000	The duration of subsegments in *ms*.
-rap	Forces segments to start random access points (RAP), i.e. keyframes.
-segment-name	The name of the segments. An increasing number and the file extension is added automatically.
-out	The name of *.mpd* file.
output_2400k.mp4 output_1200k.mp4 output_600k.mp4	The video files that should be segmented.

Now we have MPD file, init-file and a lot of segments. We can play it locally from container with MP4Client from GPAC:

```
$ export DISPLAY=:0
$ MP4Client dash.mpd
```

2.3 Configuring HTTP server

Put video files under directory `~/www/dash` and Dash.js player `~/www/dash.js.`

Now open file `/usr/local/nginx/conf/nginx.conf` in editor (full path from local computer `/var/lib/lxc/left/rootfs/usr/local/nginx/conf/nginx.conf`) and add the following simple configuration:

```
worker_processes auto;
error_log logs/error.log debug;
events {
    worker_connections 1024;
}
http {
    include mime.types;
    default_type application/octet-stream;
    sendfile on;
    keepalive_timeout  65;
    server {
      listen 80;
        location / {
            root /home/ubuntu/www;
        }
    }
}
```

The directives are listed in Table 2.4.

Table 2.4 Directives for nginx [23].

worker_processes *number*	Defines the number of worker processes. The value "*auto*" will try to autodetect it.
error_log *file level*	Configures logging. Several logs can be specified on the same level. The first parameter defines a file that will store the log. The second parameter determines the level of logging.
events *{ ... }*	There are situated the directives that works with connection processing.
worker_connections *number*	Defines the maximum number of connections that can be opened simultaneous.
http *{...}*[24]	Specifies the HTTP server directives.
include *file* | *mask*	Includes another *file*, or files matching the specified *mask*, into configuration.
default_type *mime-type*	Sets default MIME type for response.
keepalive_timeout *timeout*	Sets a how long server keeps alive client's connection.
server *{ ... }*	Sets configuration for a virtual server.
listen *address[:port]*	Sets the *address* and *port*, or the *path* for a UNIX-domain socket on which the server will accept requests. Both *address* and *port*, or only *address* or only *port* can be specified.
location *uri { ... }*	Sets configuration depending on a request URI.

Now we have the server configured.

Then start server on the left virtual machine (VM) with:

```
$ sudo /usr/local/nginx/sbin/nginx
```

And just for control run simple simulation of LAN with two nodes. Go to ns-3 directory and run:

```
$ /waf –run tap-csma-virtual-machine
```

On the right VM run a browser (i.e. google-chrome):

```
$ export DISPLAY=:0
$ google-chrome
```

And connect to `http://[ip adress of left VM]/dash.js/samples/dash-if-reference-player/index.html`.

In the address bar on the player's web-page put address of *.mpd* file. In our case it is `http://[ip adress of left VM]/dash/dash.mpd .`

2.4 DASH validator [25]

On [25] we can check the *.mpd* file for conformance of ISO/IEC 23009-1 MPEG-DASH MPD and Segments [26]. The Validator said that our *.mpd* file complies:

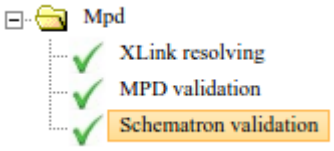


Figure 2.1 Validation results

3 CREATE LOADED WIRED LAN SIMULATION

In this part, we are going to write a C++ script, which will create a Local Area Network (LAN) simulation in ns-3 and interact with a real world. By default all user scripts are located in `/path/to/ns-3/scratch/` .

3.1 Configuring the devices and channel

We want to create a simulation than will be close to real word networks. We create a LAN with a variable number of nodes, which are sending packets at a random time to an echo server and echo server answers them.

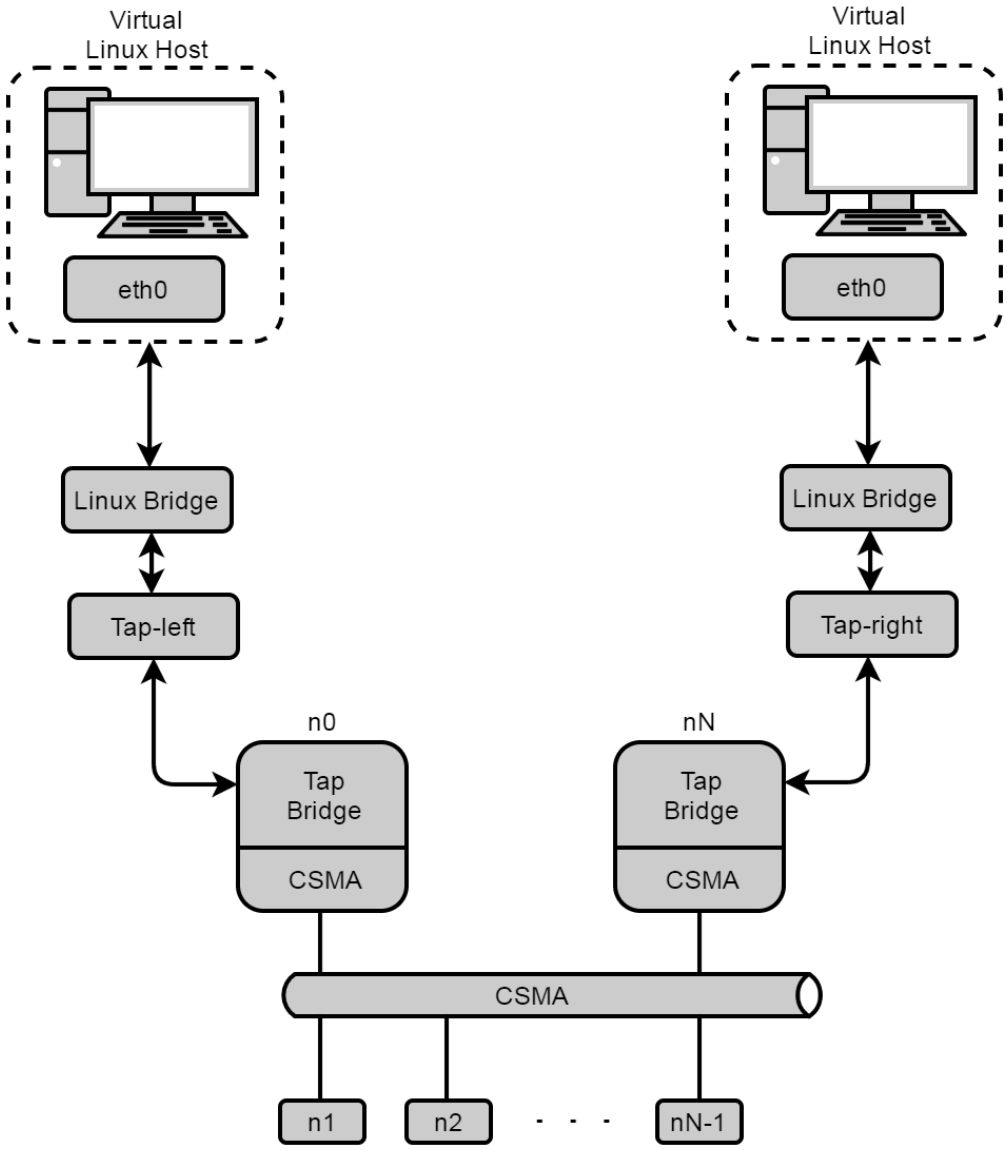


Figure 3.1 LAN configuration.

The code starts with a number of include statements [27].

First we need to include standard C++ Input/Output Stream [28].

```
#include <iostream>
```

Then, include ns-3 files. The *ns-3* has large modules and to simplify it provides a single include file that will load all necessary files for the module.

```
#include "ns3/core-module.h" //ns-3 core module
#include "ns3/network-module.h" //responsible for network devices, channels
and packets creation
#include "ns3/internet-module.h" // responsible for routing and protocol
implementations
#include "ns3/csma-module.h" // responsible for CSMA devices and channels
#include "ns3/tap-bridge-module.h" // responsible for tap bridges
#include "ns3/applications-module.h" // responsible for applications like
UdpClientServer
```

The *ns-3* project is implemented in a C++ namespace called *ns3*. This groups are *ns-3*-related declarations and situated outside the global namespace. The C++ can add the *ns-3* namespace into the current (global) declarative region [27].

A namespace declaration:

```
using namespace ns3;
```

Define a Log component with a specific name:

```
NS_LOG_COMPONENT_DEFINE ("TapCsmaVirtualMachine");
```

The declaration of the main function of program:

```
int main (int argc, char *argv[])
```

Another way we can change *ns-3* scripts without editing and building is via *command line arguments* [29]. The *ns-3* provide a mechanism to parse command line arguments and automatically set local and global variables based on those arguments.

```
uint32_t nNodes = 15; //number of nodes
bool verbose = false; // on/of logging

CommandLine cmd;
cmd.AddValue ("nNodes", "Number of \"extra\" CSMA nodes/devices", nNodes);
cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);
cmd.Parse (argc, argv);
```

The next lines of the script are used to enable two logging components that are built into the echo client and echo server applications [27]:

```
if (verbose)
{
      LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
      LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
}
```

We are interacting with the outside world. This means we have to interact in real-time and we have to use the real-time simulator and take the time to calculate checksums [30].

```
GlobalValue::Bind ("SimulatorImplementationType", StringValue
("ns3::RealtimeSimulatorImpl"));
GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));
```

3.1.1 The left side

The next lines of code will create the *ns-3 Node* objects that will represent the computers in the simulation:

```
nNodes +=2; //2 is for our virtual machines
NodeContainer nodes;
nodes.Create (nNodes);
```

The *NodeContainer* topology helper is used to create, manage and access nodes. The second line above declares a *NodeContainer* with the name *nodes*. The third line calls the *Create* method and asks the container to create *nNodes* nodes.

We will use a *CsmaHelper* to configure and connect *ns-3* csma devices and csma channel:

```
CsmaHelper csma; //instantiates a CsmaHelper object
csma.SetChannelAttribute ("DataRate", StringValue ("5Mbps")); //tells the
CsmaHelper to use the value “5Mbps” as the “DataRate” when it creates a
csma device
csma.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (6560)));
NetDeviceContainer devices = csma.Install (nodes); //tells the CsmaHelper
to use the value “2ms” as the value of the propagation delay of every csma
channel
```

We will need to have a list of all of the *NetDevice* objects that are created, so we use a *NetDeviceContainer* to hold them.

```
NetDeviceContainer devices = csma.Install (nodes);
```

This will finish configuring the devices and channel.

3.1.2 Internet stack

Install protocol stacks (TCP, UDP, IP, etc.) on our nodes:

```
InternetStackHelper stack; //creating Internet Stack
stack.Install (nodes); //install Internet Stack on each of the nodes in the
node container.
```

Next we need to associate the devices on our nodes with IP addresses. Create an address helper object and tell it to allocate IP addresses. The first parameter is “Network”, the second - “Network Mask”. The third parameter is “Initial address” and is it optional.

```
Ipv4AddressHelper address;
address.SetBase ("10.1.1.0", "255.255.255.0");
```

Make the actual address assignment, using an *Ipv4Interface* object:

```
Ipv4InterfaceContainer interfaces;
interfaces = address.Assign (devices);
```

Write IP addresses of the left and right tap-devices to the command window:

```
std::cout << "IP of tap-left (node 0) is: " << interfaces.GetAddress(0) <<
std::endl;
std::cout << "IP of tap-right (node " << nNodes-1<< ") is: " <<
interfaces.GetAddress(nNodes-1) << std::endl;
```

3.2 Applications

In this part, we add server and client applications. They will be communicate each other and generate traffic.

3.2.1 UdpEchoServerHelper

Another important part of the ns-3 system is the *Application* [31]. In this script we use two applications called *UdpEchoServerApplication* and *UdpEchoClientApplication.*

Set up a UDP echo server application:

```
UdpEchoServerHelper echoServer (9); //creating the echo server on port 9
ApplicationContainer serverApps = echoServer.Install (nodes.Get (1)); //
install server application on the node 1
serverApps.Start (Seconds (1.0));
serverApps.Stop (Seconds (600.0));
```

UdpEchoServerHelper is an object used to create the actual applications. The result of *nodes.Get(1)* returns a smart pointer to a node object - *Ptr<Node>*. *echoServer.Install* is going to install a *UdpEchoServerApplication* on the node with index number 1 of the *NodeContainer*.

Applications require a “start” and optional “stop” time to generate traffic. The echo server application will enable itself at one second into the simulation and disable itself at ten minutes into the simulation.

3.2.2 UdpEchoClientHelper

We are going to create echo client applications, which would send packets to the echo server. Therefore, we need to use random numbers to send packets in random time to bring closer network model to real life.

ns-3 contains a built-in pseudo-random number generator (PRNG). A class *ns3::RngSeedManager* provides an API to control the seeding and run number behavior. This seeding and substream state setting must be called before any random variables are created [32]:

```
RngSeedManager::SetSeed (3); // Changes seed from default of 1 to 3
RngSeedManager::SetRun (7);  // Changes run number from default of 1 to 7
```

Create the object (our random number) from the class *RandomVariableStream*:

```
Ptr<UniformRandomVariable> rand = CreateObject<UniformRandomVariable>();
```

Then set up the maximum and the minimum for object:

```
rand->SetAttribute( "Min", DoubleValue( 10 ) );
rand->SetAttribute( "Max", DoubleValue( 1000 ) );
```

Similar to the echo server, create echo clients. Two attributes are “RemoteAddress” and “RemotePort” of the remote echo server:

```
UdpEchoClientHelper echoClient (interfaces.GetAddress (1), 9); //echo
client application will send packets to node 1 port 9
```

Next, create client applications, set the maximum number of packets allowed to send during simulation, the interval between sending packets, the size of the packets and install those applications on the nodes. It is needed to use a loop to set all nodes a random packet interval:

```
ApplicationContainer clientApps;
for ( uint32_t  i=0; i<nNodes; i++ )//add aplications to nodes
{
      echoClient.SetAttribute ("MaxPackets", UintegerValue (50000));
//50000 should be enough for 10 minutes simulation
      echoClient.SetAttribute ("Interval", TimeValue (MilliSeconds (rand-
>GetInteger()))); //random interval
      echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
      clientApps = (echoClient.Install (nodes.Get(i)));
}
```

Set start and stop time for echo client applications:

```
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (600.0));
```

3.3 Tap briges

Now attach VMs to the tap-devices. There is exist a few *TapBridge* modes, but only *UseLocal* mode is suitable. In that mode *TapBridge* is going to use an existing tap device previously created and configured by the user. *ns-3* is used only to provide simulated networks for those devices [33].

```
// Use the TapBridgeHelper to connect to the pre-configured tap devices on
the left-side
TapBridgeHelper tapBridge;
tapBridge.SetAttribute ("Mode", StringValue ("UseLocal"));
tapBridge.SetAttribute ("DeviceName", StringValue ("tap-left"));
tapBridge.Install (nodes.Get (0), devices.Get (0));

// Connect the right side tap to the right side CSMA device

tapBridge.SetAttribute ("DeviceName", StringValue ("tap-right"));
tapBridge.Install (nodes.Get (nNodes-1), devices.Get (nNodes-1));
```

At the end run the simulator for ten minutes:

```
Simulator::Stop (Seconds (600.));
Simulator::Run ();
Simulator::Destroy ();
```

3.4 Run the simulation

This part describes how to use command line arguments to run scripts with some parameters without touching script's code.

3.4.1 Command line arguments

To just run the simulation we can use this in the *ns-3* directory:

```
$ ./waf --run myscript
```

But we have added a command line arguments and ns-3 can tell how to use them [34]:

```
$ ./waf --run "myscript --PtintHelp"
```

It will shows the following:

```
myscript [Program Arguments] [General Arguments]
Program Arguments:
    --nNodes:   Number of "extra" CSMA nodes/devices [15]
    --verbose:  Tell echo applications to log if true [false]
General Arguments:
    --PrintGlobals:              Print the list of globals.
    --PrintGroups:               Print the list of groups.
    --PrintGroup=[group]:        Print all TypeIds of group.
    --PrintTypeIds:              Print all TypeIds.
    --PrintAttributes=[typeid]:  Print all attributes of typeid.
    --PrintHelp:                 Print this help message.
```

In the section “Program Arguments” there are arguments that we have added in script. We can run the script with different number of CSMA nodes without changing the script:

```
$ ./waf --run “myscript –nNodes=5”
```

Or turn on logging:

```
$ ./waf --run “myscript –verbose=true”
```

Even change speed of the CSMA channel:

```
$ ./waf --run "myscript –ns3::CsmaChannel::DataRate=10Mbps"
```

3.4.2 Test running

In the *config* file to the left VM change back bridge to `lxc.network.link = br-left` to get connection to ns-3 network.

Start the VMs with:

```
$ sudo lxc-start -n left
$ sudo lxc-start -n left
```

According to the configuration, the left VM must have IP address 10.1.1.1 to communicate. Therefore, we need to configure VM's interface eth0:

```
$ sudo ifconfig eth0 10.1.1.1 netmask 255.255.255.0
```

For example, we have 5 extra nodes in the network, so the right VM must have IP address 10.1.1.7 :

```
$ sudo ifconfig eth0 10.1.1.7 netmask 255.255.255.0
```

Let run a simulation with 5 extra nodes a turn on logging:

```
$ sudo ./waf --run "myscript --nNodes=5 –verbose=true"
```

In the command window we will see:

```
…
At time 1.484s client sent 1024 bytes to 10.1.1.2 port 9
At time 1.49372s server received 1024 bytes from 10.1.1.4 port 49153
At time 1.49372s server sent 1024 bytes to 10.1.1.4 port 49153
At time 1.495s client received 1024 bytes from 10.1.1.2 port 9
…
```

As we can see, a client sends 1024 bytes to the server and the server sends them back.

VMs can ping each other:

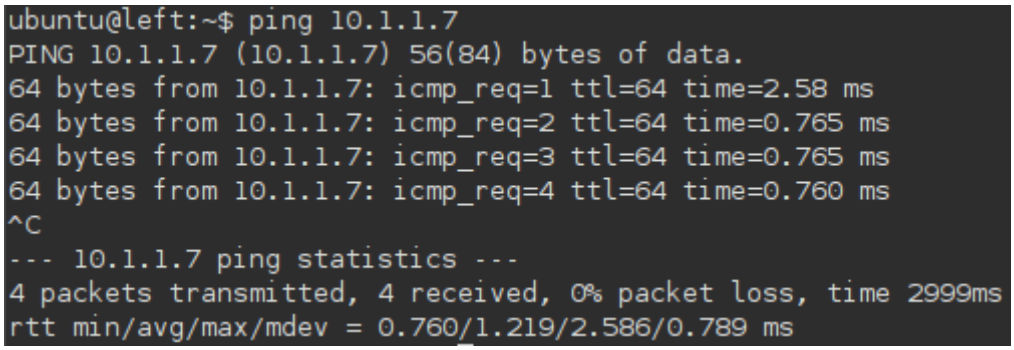


Figure 3.2 Pinging the right VM.

The system works perfectly. Directly from the command line we can change the number of the nodes, the size of packets, the seed and the run number for the random interval, so every run of the simulation can be the same or completely different.

4 CREATE WIFI SIMULATION

Based on the simulation of a wired network build a wireless network. WiFi simulation [35] almost the same, except few moments.

The biggest problem is that tap-devices do not work well with anything except CSMA. So, if we create only WiFi network and connect tap-devices directly to the WiFi-nodes it will works strangely. Assume we connect tap-left to an STA (station) and tap-right to the AP (Access Point). From the AP we can ping all nodes but tap-left, from STA we cannot ping any device. Therefore, we need to add CSMA links to the both ends of the system and connect them to WiFi network through P2P links.

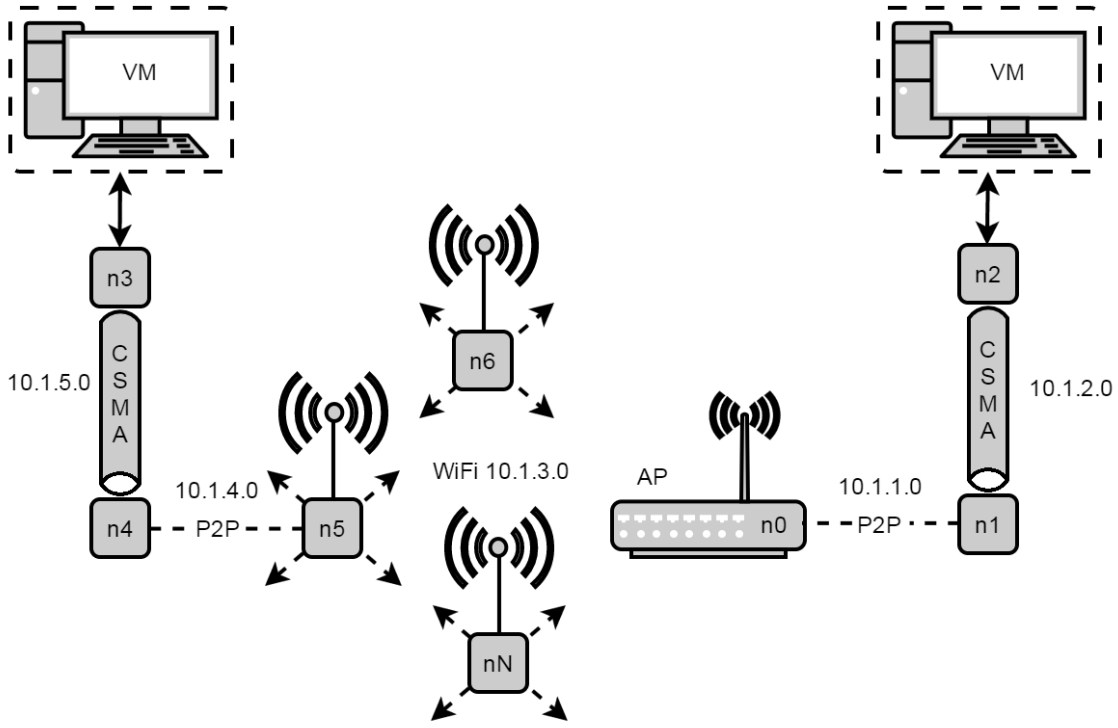


Figure 4.1 The configuration of WiFi network.

First, add necessary modules:

```
#include "ns3/core-module.h"
#include "ns3/csma-module.h"
#include "ns3/wifi-module.h" //responsible for WiFi STAs and APs, MAC
layer, propagation models and control algorithms
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/mobility-module.h" // to add mobility to STA nodes
#include "ns3/tap-bridge-module.h"
#include "ns3/internet-module.h"
#include "ns3/point-to-point-module.h" //responsible for P2P nodes and
channels
```

Add command line arguments to choose number of nodes and to turn on/off application logging to the command window:

```
bool verbose = false;
uint32_t nWifi = 3;

CommandLine cmd;
cmd.AddValue ("nWifi", "Number of wifi STA devices", nWifi);
cmd.AddValue ("verbose", "Tell echo applications to log if true", verbose);
cmd.Parse (argc,argv);

if (verbose)
{
LogComponentEnable ("UdpEchoClientApplication", LOG_LEVEL_INFO);
LogComponentEnable ("UdpEchoServerApplication", LOG_LEVEL_INFO);
      }
```

For the real-time simulation, same as in the LAN-simulation add to script next lines:

```
GlobalValue::Bind ("SimulatorImplementationType", StringValue
("ns3::RealtimeSimulatorImpl"));
GlobalValue::Bind ("ChecksumEnabled", BooleanValue (true));
```

4.1 Right side

Create P2P nodes to join WiFi and LAN networks:

```
NodeContainer p2pNodesRight;
p2pNodesRight.Create (2);
```

Because we want to simulate only WiFi network, P2P and CSMA links must bring minimal influence in the simulation. Thus, we make this links with minimal delay and maximal speed. 100 Gbps and 1 nanosecond should be enough:

```
PointToPointHelper pointToPointRight;
pointToPointRight.SetDeviceAttribute ("DataRate", StringValue ("100Gbps"));
pointToPointRight.SetChannelAttribute ("Delay", TimeValue(NanoSeconds
(1)));
```

And install this devices:

```
NetDeviceContainer p2pDevicesRight;
p2pDevicesRight = pointToPointRight.Install (p2pNodesRight);
```

For the CSMA we need to take one node from the P2P and create one node:

```
NodeContainer csmaNodesRight;
csmaNodesRight.Add (p2pNodesRight.Get (1));
csmaNodesRight.Create (1);
```

Then the same as for the P2P link:

```
CsmaHelper csmaRight;
csmaRight.SetChannelAttribute ("DataRate", StringValue ("100Gbps"));
csmaRight.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (1)));
NetDeviceContainer csmaDevicesRight;
csmaDevicesRight = csmaRight.Install (csmaNodesRight);
```

4.2 WiFi nodes

Next important step is to create the WiFi STA and AP nodes and add mobility to STA nodes.

Create STA nodes:

```
NodeContainer wifiStaNodes;
wifiStaNodes.Create (nWifi);
```

Take P2P node with index 1 as the node for access point:

```
NodeContainer wifiApNode = p2pNodesRight.Get (0);
```

Then constructs the interconnection channel between these WiFi nodes. Configure the PHY and channel helpers [36]:

```
YansWifiChannelHelper channel = YansWifiChannelHelper::Default ();
YansWifiPhyHelper phy = YansWifiPhyHelper::Default ();
phy.SetChannel (channel.Create ());
```

We use the default PHY layer configuration and channel models [35]

Confugure the MAC layer. We use a *NqosWifiMacHelper* object because we will work with non-QoS MAC and set AARF algorithm as rate control algorithm [37]:

```
WifiHelper wifi = WifiHelper::Default ();
wifi.SetRemoteStationManager ("ns3::AarfWifiManager");
NqosWifiMacHelper mac = NqosWifiMacHelper::Default ();
```

Next, we set the type of MAC for STA, the SSID of the network and make sure that probe requests will not be sent:

```
Ssid ssid = Ssid ("ns-3-ssid");
mac.SetType ("ns3::StaWifiMac",
             "Ssid", SsidValue (ssid),
             "ActiveProbing", BooleanValue (false));
```

Install WiFi devices on the nodes:

```
NetDeviceContainer staDevices;
staDevices = wifi.Install (phy, mac, wifiStaNodes);
```

Similar, set MAC parameters of the AP. Also, set beacon generating each 2.5 seconds.

```
mac.SetType ("ns3::ApWifiMac",
             "Ssid", SsidValue (ssid),
             "BeaconGeneration", BooleanValue (true),
             "BeaconInterval", TimeValue (Seconds (2.5)));
```

Create the AP which shares the same and channel as the STAs:

```
NetDeviceContainer apDevice;
apDevice = wifi.Install (phy, mac, wifiApNode);
```

4.3 Moblility

Now, we are going to add mobility to STA nodes to be mobile, random move inside. First, we initialize a MobilityHelper object and set parameters of the two-dimensional grid to place STAs [38]:

```
MobilityHelper mobility;
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "MinX", DoubleValue (0.0), //start point
                               "MinY", DoubleValue (0.0),
                               "DeltaX", DoubleValue (5.0), //space between
objects
                               "DeltaY", DoubleValue (10.0),
                               GridWidth", UintegerValue (3), //the number of
objects layed out on a line
                               LayoutType", StringValue ("RowFirst"));
```

We need to set the borders and tell STAs to move in a random direction at a random speed:

```
mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel", Bounds",
RectangleValue (Rectangle (-50, 50, -50, 50)));
```

Install mobility models on the STA nodes:

```
mobility.Install (wifiStaNodes);
```

Put AP in fixed position [39]:

```
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (wifiApNode);
```

4.4 Left side

The same way like we did on the right side, we create the CSMA and the P2P links on the left side, but connect index 2 STA node with P2P node:

```
//create P2P nodes
NodeContainer p2pNodesLeft;
p2pNodesLeft.Add (wifiStaNodes.Get (2)); //take second WiFi node as P2P to
connect networks
p2pNodesLeft.Create (1);

//P2P channel
PointToPointHelper pointToPointLeft;
pointToPointLeft.SetDeviceAttribute ("DataRate", StringValue ("100Gbps"));
pointToPointLeft.SetChannelAttribute ("Delay", TimeValue (NanoSeconds
(1)));

//create P2P devices and install them on P2P nodes
NetDeviceContainer p2pDevicesLeft;
p2pDevicesLeft = pointToPointLeft.Install (p2pNodesLeft);

//create CSMA nodes
NodeContainer csmaNodesLeft;
csmaNodesLeft.Add (p2pNodesLeft.Get (1)); //take P2P node as CSMA to
connect networks
csmaNodesLeft.Create (1);

//create CSMA channel
CsmaHelper csmaLeft;
csmaLeft.SetChannelAttribute ("DataRate", StringValue ("100Gbps"));
csmaLeft.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (1)));

//create CSMA devices and install them on CSMA nodes
NetDeviceContainer csmaDevicesLeft;
csmaDevicesLeft = csmaLeft.Install (csmaNodesLeft);
```

4.5 Internet stack and address assigning

Then we need to install the protocol stacks on the devices:

```
InternetStackHelper stack;
stack.Install (csmaNodesRight);
stack.Install (wifiApNode);
stack.Install (wifiStaNodes);
stack.Install (csmaNodesLeft);
```

As in the LAN simulation, we are going to assign IP addresses to our device interfaces using *Ipv4AddressHelper* and write them to the command window:

```
Ipv4AddressHelper address;

//for right P2P link
address.SetBase ("10.1.1.0", "255.255.255.0"); //set address base
Ipv4InterfaceContainer p2pInterfacesRight; //create address container with
address base
p2pInterfacesRight = address.Assign (p2pDevicesRight); //assign addresses
from the container to the interfaces

//write IP addresses to the command window
std::cout << "IP of P2P right 0 is: " << p2pInterfacesRight.GetAddress(0)
<< std::endl;
std::cout << "IP of P2P right 1 is: " << p2pInterfacesRight.GetAddress(1)
<< std::endl;

//for right CSMA link
address.SetBase ("10.1.2.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfacesRight;
csmaInterfacesRight = address.Assign (csmaDevicesRight);
std::cout << "IP of CSMA right 0 is: " << csmaInterfacesRight.GetAddress(0)
<< std::endl;
std::cout << "IP of CSMA right 1 is: " << csmaInterfacesRight.GetAddress(1)
<< std::endl;

//for WiFi devices
address.SetBase ("10.1.3.0", "255.255.255.0");
address.Assign (staDevices);
address.Assign (apDevice);

//for left P2P link
address.SetBase ("10.1.4.0", "255.255.255.0");
Ipv4InterfaceContainer p2pInterfacesLeft;
p2pInterfacesLeft = address.Assign (p2pDevicesLeft);
std::cout << "IP of P2P left 0 is: " << p2pInterfacesLeft.GetAddress(0) <<
std::endl;
std::cout << "IP of P2P left 1 is: " << p2pInterfacesLeft.GetAddress(1) <<
std::endl;

//for left CSMA link
address.SetBase ("10.1.5.0", "255.255.255.0");
Ipv4InterfaceContainer csmaInterfacesLeft;
csmaInterfacesLeft = address.Assign (csmaDevicesLeft);
std::cout << "IP of CSMA left 0 is: " << csmaInterfacesLeft.GetAddress(0)
<< std::endl;
std::cout << "IP of CSMA left 1 is: " << csmaInterfacesLeft.GetAddress(1)
<< std::endl;
```

4.6 Applications

Similar to the LAN simulation, we put echo server on the AP:

```
UdpEchoServerHelper echoServer (9); //echo port
ApplicationContainer serverApps = echoServer.Install (wifiApNode);
//install echo server
serverApps.Start (Seconds (1.0)); //start time in seconds
serverApps.Stop (Seconds (600.0)); //stop time in seconds
```

Create the random number between 10 and 1000:

```
RngSeedManager::SetSeed (3); //Change seed to 3
RngSeedManager::SetRun (7); //Change run number to 7
Ptr<UniformRandomVariable> rand = CreateObject<UniformRandomVariable> ();
//create random variable object
rand->SetAttribute( "Min", DoubleValue( 10 ) ); //set minimum of the
variable
rand->SetAttribute( "Max", DoubleValue( 1000 ) ); //set maximum
```

Put the echo client on all STAs to generate traffic and make them generate it randomly:

```
UdpEchoClientHelper echoClient (p2pInterfacesRight.GetAddress(0), 9);
//tell echo client to send packets to AP, port 9
ApplicationContainer clientApps; //create client's application
for ( uint32_t  i=0; i<nWifi; i++ ){
echoClient.SetAttribute ("MaxPackets", UintegerValue (50000));
echoClient.SetAttribute ("Interval", TimeValue (MilliSeconds (rand-
>GetInteger())));  //random interval
echoClient.SetAttribute ("PacketSize", UintegerValue (1024));
clientApps = (echoClient.Install (wifiStaNodes.Get(i))); //install
application on the nodes
}
clientApps.Start (Seconds (2.0));
clientApps.Stop (Seconds (600.0));
```

4.7 Install Tap bridges

We will connect to the both ends of our system, i.e. CSMA nodes with index 1 on the left and on the right side:

```
TapBridgeHelper tapBridge; //create tap bridge
tapBridge.SetAttribute ("Mode", StringValue ("UseLocal")); //set mode
tapBridge.SetAttribute ("DeviceName", StringValue ("tap-left"));//set name
to connect
tapBridge.Install (csmaNodesLeft.Get(1), csmaDevicesLeft.Get(1)); //install
tap bridge on the device 1 on the node 1

tapBridge.SetAttribute ("Mode", StringValue ("UseLocal"));
tapBridge.SetAttribute ("DeviceName", StringValue ("tap-right"));
tapBridge.Install (csmaNodesRight.Get (1), csmaDevicesRight.Get (1));
```

We have an internetwork, so we need to enable automatic internetwork routing:

```
Ipv4GlobalRoutingHelper::PopulateRoutingTables ();
```

Then stop simulator after 10 minutes:

```
Simulator::Stop (Seconds (600.0));
```

And run the simulation:

```
Simulator::Run ();
Simulator::Destroy ();
```

5 CREATE LTE SIMULATION

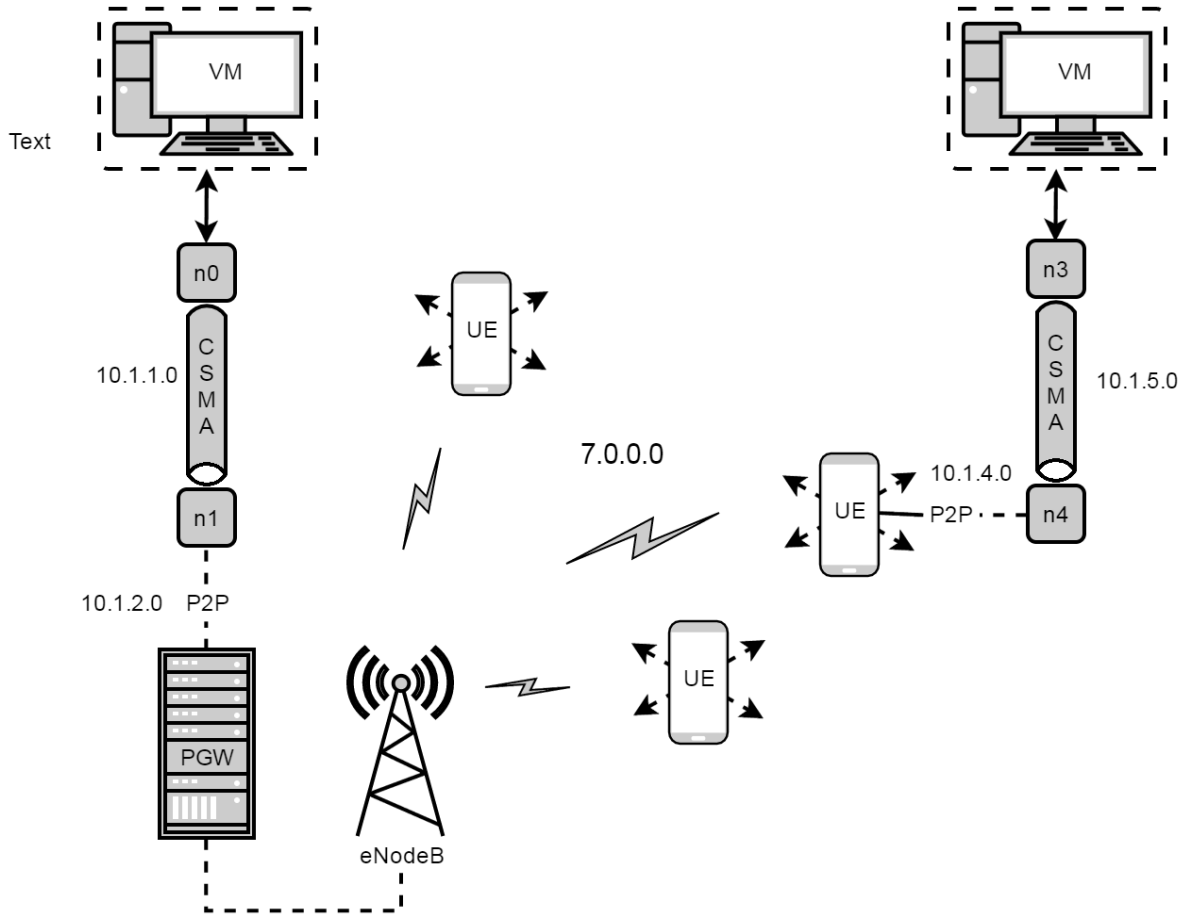


Figure 5.1 The configuration of LTE network.

In the LTE [40] simulation we need to add a lot of modules:

```
#include "ns3/core-module.h"
#include "ns3/point-to-point-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/csma-module.h"
#include "ns3/tap-bridge-module.h"
#include "ns3/internet-module.h"
#include "ns3/lte-module.h" //LTE core module
#include "ns3/lte-helper.h" //creation and configuration of LTE
#include "ns3/epc-helper.h" //Evolved Packet Core module (EPC)
#include "ns3/ipv4-static-routing.h" //for static routing, ns-3 does not
support dynamic routing for LTE
#include "ns3/ipv4-list-routing.h" // a prioritized list of routing
protocols
#include "ns3/ipv4-routing-table-entry.h" //a record of an IPv4 routing
table
#include "ns3/ipv4-static-routing-helper.h" //creation of static routing
#include "ns3/ipv4-list-routing-helper.h" //creation of a prioritized list
```

Create a PDN Gateway (PGW) as a Gateway to our VM with video streaming server.

```
Ptr<LteHelper> lteHelper = CreateObject<LteHelper> (); //create LTE helper
object
Ptr<PointToPointEpcHelper>  epcHelper = CreateObject<PointToPointEpcHelper>
(); //create EPC helper object
lteHelper->SetEpcHelper (epcHelper);
Ptr<Node> pgw = epcHelper->GetPgwNode (); //PGW node
```

Then, create the left part nodes according to topology, same as before:

```
//P2P nodes
NodeContainer p2pNodesLeft;
p2pNodesLeft.Add (pgw); //connect P2P and PGW
p2pNodesLeft.Create (1);
//P2P channel
PointToPointHelper pointToPointLeft;
pointToPointLeft.SetDeviceAttribute ("DataRate", StringValue ("100Gbps"));
pointToPointLeft.SetChannelAttribute ("Delay", TimeValue(NanoSeconds (1)));
//P2P devices
NetDeviceContainer p2pDevicesLeft;
p2pDevicesLeft = pointToPointLeft.Install (p2pNodesLeft);
//CSMA nodes
NodeContainer csmaNodesLeft;
csmaNodesLeft.Create (1);
csmaNodesLeft.Add (p2pNodesLeft.Get (1));
//CSMA channel
CsmaHelper csmaLeft;
csmaLeft.SetChannelAttribute ("DataRate", StringValue ("100Gbps"));
csmaLeft.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (1000)));
//CSMA devices
NetDeviceContainer csmaDevicesLeft;
csmaDevicesLeft = csmaLeft.Install (csmaNodesLeft);
```

Create eNodeB and User Equipment (UE) nodes:

```
NodeContainer enbNodes;
enbNodes.Create(1);
NodeContainer ueNodes;
ueNodes.Create(numberOfNodes);
```

Add Random Walk Mobility model on UEs:

```
MobilityHelper mobility;
mobility.SetPositionAllocator ("ns3::GridPositionAllocator",
                               "MinX", DoubleValue (0.0),
                               "MinY", DoubleValue (0.0),
                               "DeltaX", DoubleValue (5.0),
                               "DeltaY", DoubleValue (10.0),
                               "GridWidth", UintegerValue (3),
                               "LayoutType", StringValue ("RowFirst"));
mobility.SetMobilityModel ("ns3::RandomWalk2dMobilityModel", "Bounds",
RectangleValue (Rectangle (-50, 50, -50, 50)));
mobility.Install (ueNodes);
```

And Constant Position Mobility model on eNodeB:

```
mobility.SetMobilityModel ("ns3::ConstantPositionMobilityModel");
mobility.Install (enbNodes);
```

Install LTE devices to the nodes:

```
NetDeviceContainer enbLteDevs = lteHelper->InstallEnbDevice (enbNodes);
NetDeviceContainer ueLteDevs = lteHelper->InstallUeDevice (ueNodes);
```

The right part nodes, connect VM to UE number 0:

```
NodeContainer p2pNodesRight;
p2pNodesRight.Add (ueNodes.Get (0));
p2pNodesRight.Create (1);
PointToPointHelper pointToPointRight;
pointToPointRight.SetDeviceAttribute ("DataRate", StringValue ("100Gbps"));
pointToPointRight.SetChannelAttribute ("Delay", TimeValue (NanoSeconds
(1)));
NetDeviceContainer p2pDevicesRight;
p2pDevicesRight = pointToPointRight.Install (p2pNodesRight);
NodeContainer csmaNodesRight;
csmaNodesRight.Create (1);
csmaNodesRight.Add (p2pNodesRight.Get(1));
CsmaHelper csmaRight;
csmaRight.SetChannelAttribute ("DataRate", StringValue ("100Gbps"));
csmaRight.SetChannelAttribute ("Delay", TimeValue (NanoSeconds (1)));
NetDeviceContainer csmaDevicesRight;
csmaDevicesRight = csmaRight.Install (csmaNodesRight);
```

Install Internet stack on the nodes, PGW and eNodeB already have it:

```
InternetStackHelper stack;
stack.Install (csmaNodesLeft);
stack.Install (csmaNodesRight);
stack.Install (ueNodes);
Ipv4AddressHelper address;
```

Assigning addresses to P2P and CSMA nodes is the same. However, IP address and default gateway of UEs are set by the network.

First, write IP addresses of the both interfaces of PGW to the command window:

```
Ptr<Node> refPgw = epcHelper->GetPgwNode();
Ptr<Ipv4> ipv4p = refPgw->GetObject<Ipv4> ();
Ptr<Ipv4> ipv4p = refPgw->GetObject<Ipv4> ();
Ipv4Address addr1p = ipv4p->GetAddress (1, 0).GetLocal ();
std::cout << "PGW if 1: " << addr1p << std::endl;
Ipv4Address addr2p = ipv4p->GetAddress (2, 0).GetLocal ();
std::cout << "PGW if 2: " <<  addr2p << std::endl;
```

Next, assign addresses to UE devices using EPC Helper [41]:

```
Ipv4InterfaceContainer ueIpIface;
ueIpIface = epcHelper->AssignUeIpv4Address (NetDeviceContainer
(ueLteDevs));
//just write IP adresses of UEs
for (uint32_t i=0; i<numberOfNodes; i++)
std::cout << "IP of UE" << i <<": " << ueIpIface.GetAddress(i) <<
std::endl;
```

Set default gateway to UEs automatically by the network using EPC Helper:

```
Ipv4StaticRoutingHelper ipv4RoutingHelper;
for (uint32_t u = 0; u < numberOfNodes; u++)
{
      Ptr<Node> ueNode = ueNodes.Get (u);
      // Set the default gateway for the UE
      Ptr<Ipv4StaticRouting> ueStaticRouting =
ipv4RoutingHelper.GetStaticRouting (ueNode->GetObject<Ipv4> ());
      ueStaticRouting->SetDefaultRoute (epcHelper-
>GetUeDefaultGatewayAddress (), 1);
}
```

And attach UEs to eNodeB:

```
      lteHelper->Attach (ueLteDevs, enbLteDevs.Get (0));
```

Then, we assign IP addresses to the right part of the network and connect VMs to the network via *tap-bridges*. Everything is the same as we done before.

In the end, we need to add routes for packets. Due to ns-3 does not support automatic configuration for LTE as in WiFi network, we need to add it manually. We are going to use static routing [42]. Template is:

```
AddNetworkRouteTo (remoteNetwork, remoteNetworkMask, interface, metric
(default is 0))
```

We will use interfaces 1 or 2 according to the network topology, interface 0 is lopback. For CSMA node 1:

```
Ptr<Ipv4StaticRouting> csmaStaticRoutingLeft =
ipv4RoutingHelper.GetStaticRouting (csmaNodesLeft.Get(1)->GetObject<Ipv4>
());

//to UE
csmaStaticRoutingLeft->AddNetworkRouteTo (Ipv4Address ("7.0.0.0"), Ipv4Mask
("255.0.0.0"), 2);

//to p2p left
csmaStaticRoutingLeft->AddNetworkRouteTo (Ipv4Address ("10.1.2.0"),
Ipv4Mask ("255.255.255.0"), Ipv4Address ("10.1.2.1"), 2);

//to p2p right
csmaStaticRoutingLeft->AddNetworkRouteTo (Ipv4Address ("10.1.4.0"),
Ipv4Mask ("255.255.255.0"), Ipv4Address ("10.1.2.1"), 2);

//to csma right
csmaStaticRoutingLeft->AddNetworkRouteTo (Ipv4Address ("10.1.5.0"),
Ipv4Mask ("255.255.255.0"), 2);
```

And continue with adding routes to all nodes.

6 ANALYSIS OF ADAPTIVE STREAMING

In this part we analyzing how video quality changes with the different number of nodes in wired and wireless network. In part number 8 shows how video adapts to network conditions.

6.1 Wired network

Before running the simulation we need VMs running.

The name of ns-3 script with the wired network that we created is *tap-csma-virtual-machine-n-nodes* and it is situated in *scratch* directory. For beginning, run script with 0 extra nodes:

```
$ sudo ./waf --run "scratch/tap-csma-virtual-machine-n-nodes –nNodes=0"
```

When we run it, it will show IP addresses of VMs:

```
IP of tap-left (node 0) is: 10.1.1.1
IP of tap-right (node 1) is: 10.1.1.2
```

We need to assign them to Ethernet devices of VMs. On the left VM:

```
$ sudo ifconfig eth0 10.1.1.1 netmask 255.255.255.0
```

On the right VM:

```
$ sudo ifconfig eth0 10.1.1.2 netmask 255.255.255.0
```

Next, for video streaming start *nginx* server on the left VM:

```
$ sudo /usr/local/nginx/sbin/nginx
```

Then, on the right VM run a browser:

```
$ export DISPLAY=:0
$ google-chrome
```

You can use another bowser, but be sure that it supports Media Source Extensions (MSE) API set. Check it here [43]. Chrome browser from version 23 supports DASH.

Also, it is better to open Chrome in development mode (F12) and check “Disable cache”, so it does not save video parts in a cache .

In the address bar type address to DASH JavaScript Player :

```
http://10.1.1.1/dash.js/samples/dash-if-reference-player/index.html
```

On the player's web page in the manifest address bar put address to *.mpd* file and click “Load”:

```
http://10.1.1.1/dash/dash.mpd
```

On the right side of the player are situated the parameters of the video:

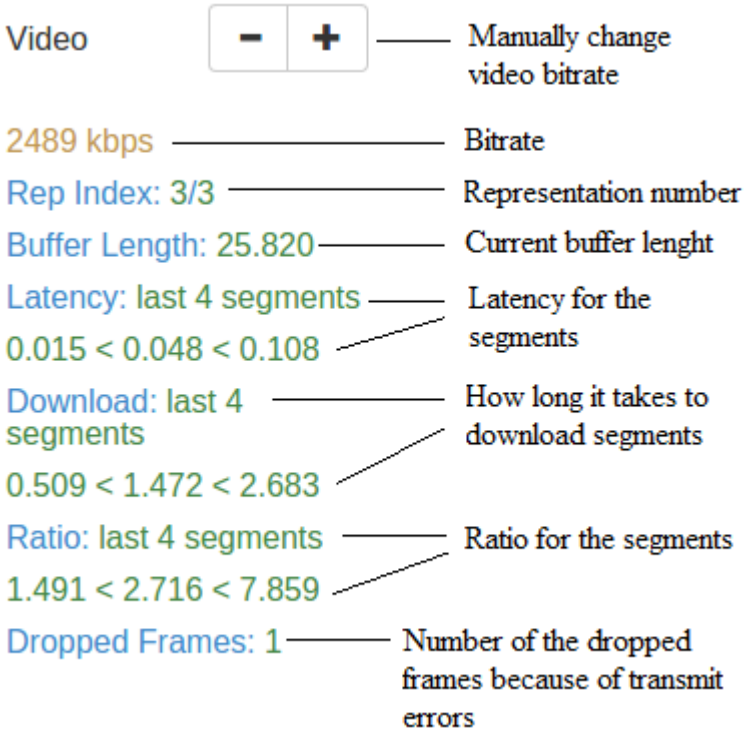


Figure 6.1 The video parameters.

Because we have almost point-to-point network and nothing affects it, we can see that bitrate is always maximum. Only after rewinding the video, it can be a little pause and a quality drop.

Now we can simulate the wired network with the different number of nodes. Dependence of the average quality and the number of the nodes in the network given below:

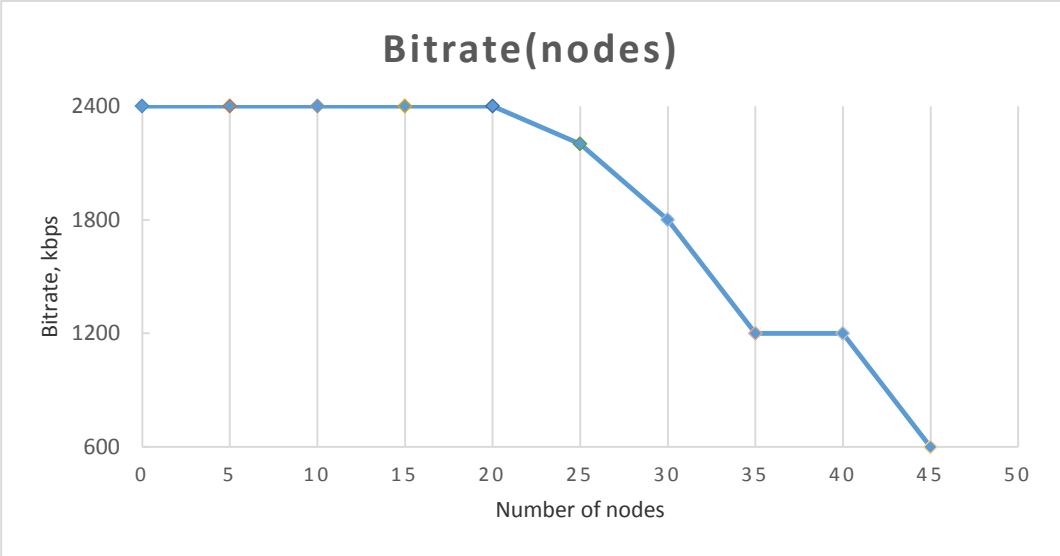


Figure 6.2 Dependence of the average quality and the number of the nodes in wired network.

When the buffer empties, quality of the video changes to lower. With an increasing number of nodes, delays after rewinds also increase.

Up to 40 nodes in the network the quality is great, but after 50 nodes network is overloaded and it only takes 60-80 sec. to download the player's web page and it almost impossible to watch the video because of delays.

6.2 Wireless network

In WiFi simulation because of nodes situated in different networks we need to add a default gateway to the interface configuration. After simulation starts we can see IP addresses of the nodes in the network:

```
IP of P2P right 0 is: 10.1.1.1
IP of P2P right 1 is: 10.1.1.2
IP of CSMA right 0 (tap-right) is: 10.1.2.1
IP of CSMA right 1 (dg right) is: 10.1.2.2
IP of P2P left 0 is: 10.1.4.1
IP of P2P left 1 is: 10.1.4.2
IP of CSMA left 0 (tap-left) is: 10.1.5.1
IP of CSMA left 1 (dg left)is: 10.1.5.2
```

The left VM is connected to CSMA left 1, so to configure its IP address and default gateway enter following:

```
$ sudo ifconfig eth0 10.1.5.2 netmask 255.255.255.0
$ sudo route add default gw 10.1.5.1 eth0
```

The right VM is connected to CSMA right 1, so to configure its IP address and default gateway enter following:

```
$ sudo ifconfig eth0 10.1.2.2 netmask 255.255.255.0
$ sudo route add default gw 10.1.2.1 eth0
```

Also, IP address of the DASH player changes to 10.1.5.2, so we need to put in address bar:

```
http://10.1.5.2/dash.js/samples/dash-if-reference-player/index.html
```

And for .mpd file:

```
http://10.1.5.2/dash/dash.mpd
```

To run the simulation with 3 nodes:

```
$ sudo ./waf --run "scratch/tap-wifi-virtual-machine-n-nodes --nWifi=3"
```

Graph is given below shows dependence between the quality and the number of nodes in the WiFi network:

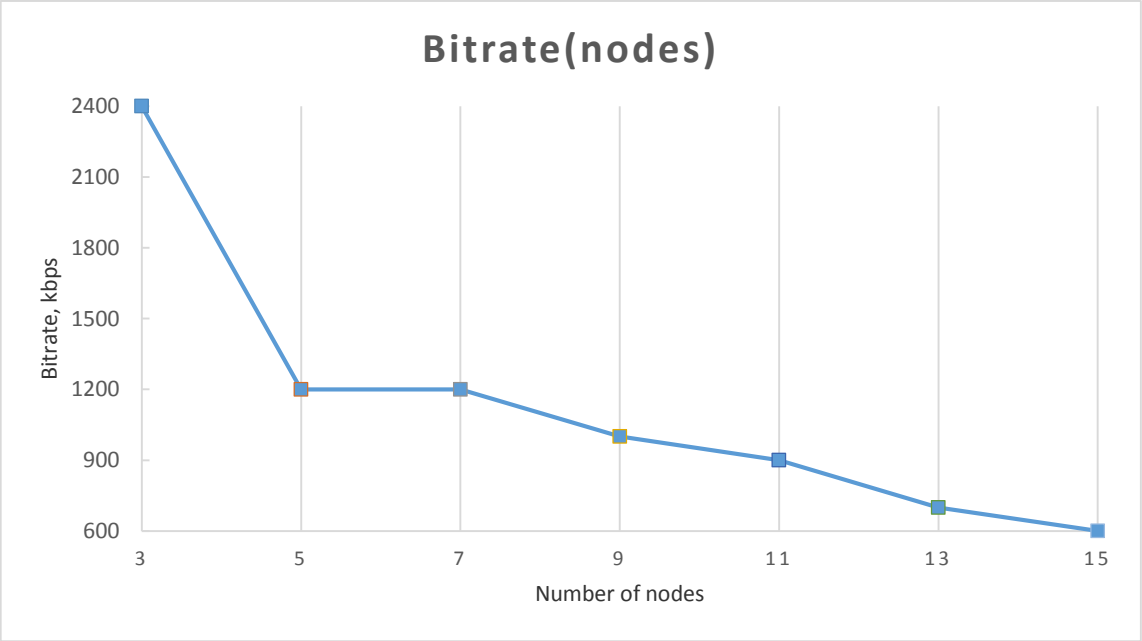


Figure 6.3 Dependence between the average quality and the number of nodes in the WiFi network.

After 10 nodes appear delays while watching a video. With an increasing number of STAs, the number of short delays also increases, then delays become longer. Because ns-3 supports up to 19 STAs we cannot get the moment when it is impossible to watch the video.

7 PERFORMANCE FOR VIDEOS WITH DIFFERENT SEGMENT DURATION

We have a video with 4s segments and are going to create 1s, 10s and 15s long segments. We need to re-encode video because the *keyint* should match the desired segment length in seconds multiplied with the frame rate. For 10s long segment: 10 seconds * 24 frames/seconds = 240 frames.

For example, re-encoding for 10s segments:

```
$ cd ~
$ mkdir dash-10s
$ cd dash
$ sudo x264 --output intermediate_2400k.264 --fps 24 --preset slow --
bitrate 2400 --vbv-maxrate 4800 --vbv-bufsize 9600 --min-keyint  --keyint
240 --scenecut 0 --no-scenecut --pass 1 --video-filter "resize:width=1280,
height=720" Example.mkv

//repeat this step for 1200 and 600 kbps

$ sudo MP4Box -dash 10000 -frag 10000 -rap -segment-name segment_%s -url-
template -out ../dash-10s/dash.mpd output_2400k.mp4 output_1200k.mp4
output_600k.mp4
$ sudo MP4Box -dash 10000 -frag 10000 -rap -segment-name segment_%s -url-
template -out ../dash-10s/dash.mpd output_2400k.mp4 output_1200k.mp4
output_600k.mp4
```

Also, we repeat all steps for all segment durations.

Now we have different segments and can compare them. We will use the wireless network with 10 STAs because delays appear irregularly and we can see how different segment duration affects on user experience.

1s long segments

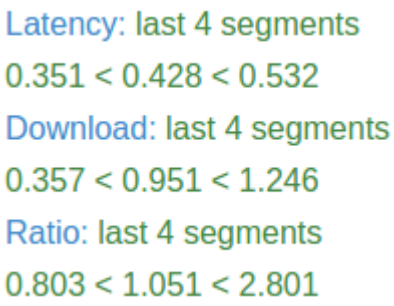


Figure 7.1 Video parameters for 1s segment.

As we can see from the “Download” part of the picture, it can take a more than second to download 1s segment. Therefore, while watching a video we will get a huge amount of small delays, but fast rewind. From user experience, it is very unpleasant to watch such kind of video.

4s long segments

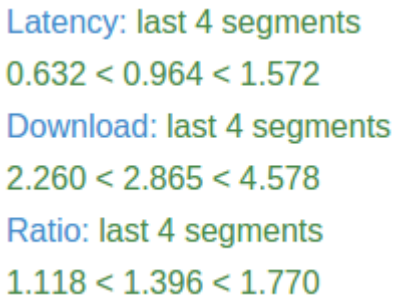


Figure 7.2 Video parameters for 4s segment.

As well as before, we can see many short delays, but they appears rarely.

10s long segments

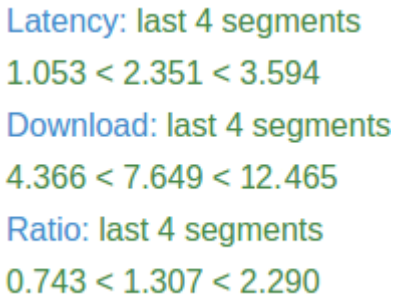


Figure 7.3 Video parameters for 10s segment.

Longer delays while rewinding the video, rare longer delays.

15s long segments

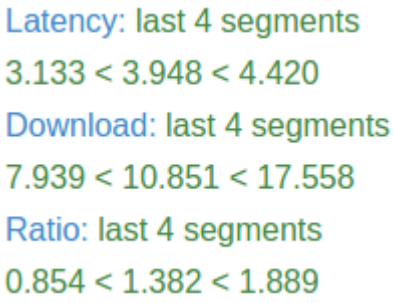


Figure 7.4 The video parameters for 15s segment.

There are rare delays, but they are too long. And there are really long delays when rewinding.

For wireless video streaming is better to use 8-10s long segments. It is a compromise between the amount of delays and their duration. For universal usage (wireless and wired networks), it is better to use 5-8s long segments.

8 MPEG-DASH IN WORK

We can take one of the free videos form DASH-IF [44] or encode own using method given above in part 2.3.

For example, download DASH-encoded video named “Elephants Dream” and look how it adapts for three WiFi network settings: 4, 8 and 12 nodes. Video has 4 representations: 100, 175, 250 and 500 kbps.

This part is tested in system created in virtual machine running Ubuntu 14.04 with LXDE desktop environment [45] for less computer resource usage. Because VM has fewer resources than system created directly on the computer, average video quality will be less.

For 4 nodes is the network prevalent quality is 500 kbps. Changes of the video bitrate are shown below.

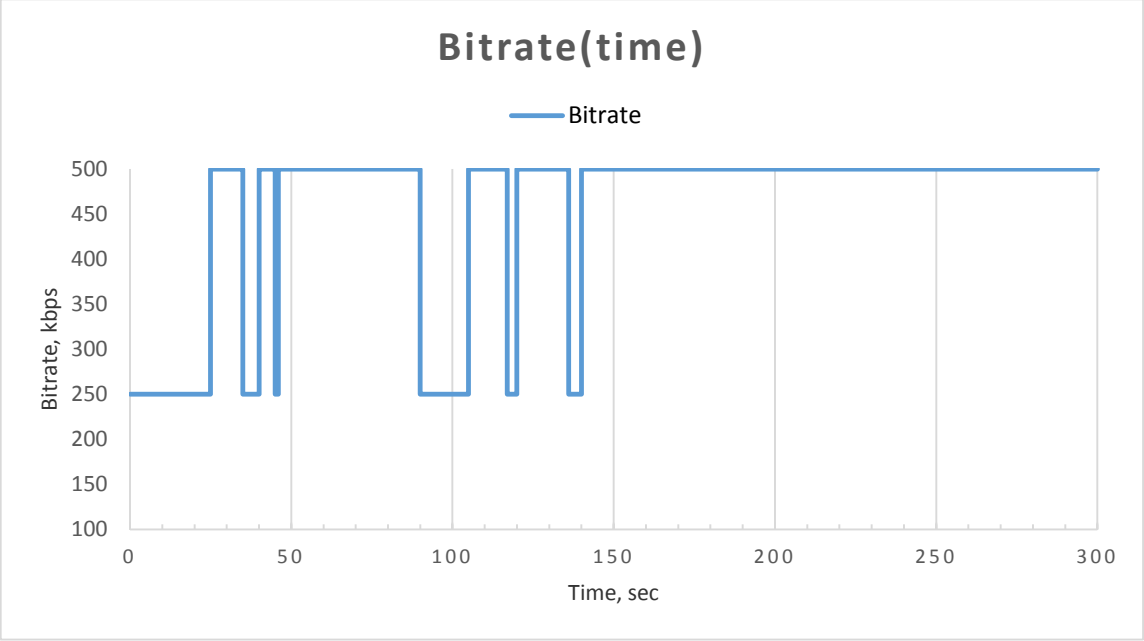


Figure 8.1 Bitrate changes for 4 nodes in the network.

For 8 nodes in the network quality hops between 250 and 500 kbps, what is shown below.

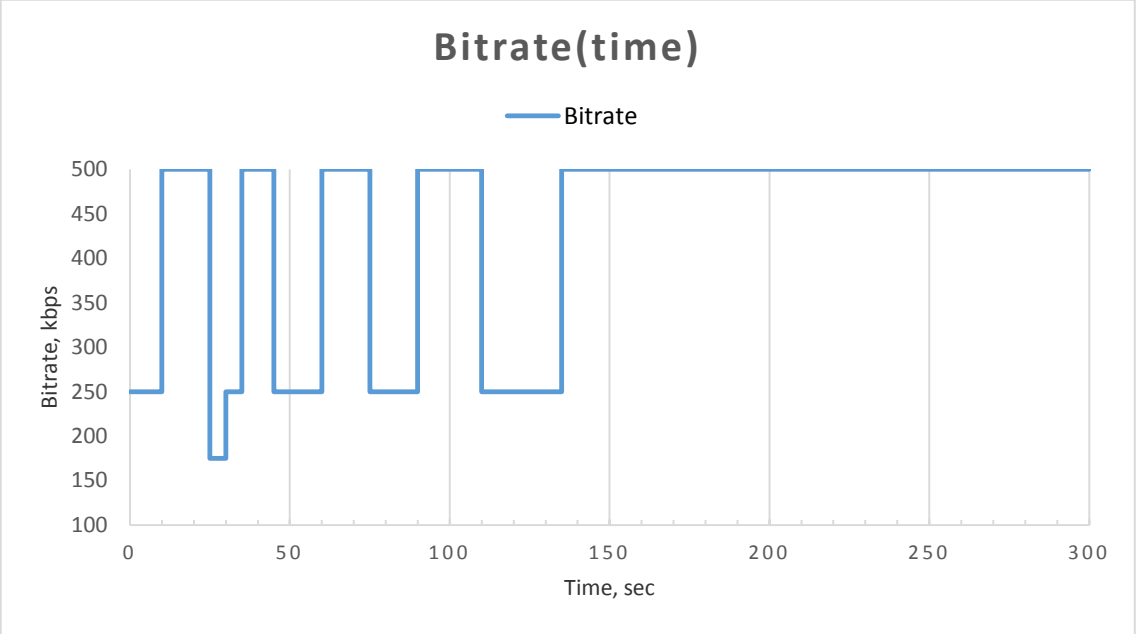


Figure 8.2 Bitrate changes for 8 nodes in the network.

For 12 nodes in the network quality jumps between 100 and 500 kbps. When buffer has enough data, player tries to get 500 kbps quality, then buffer quickly becomes empty and quality drops back to 100 kbps.

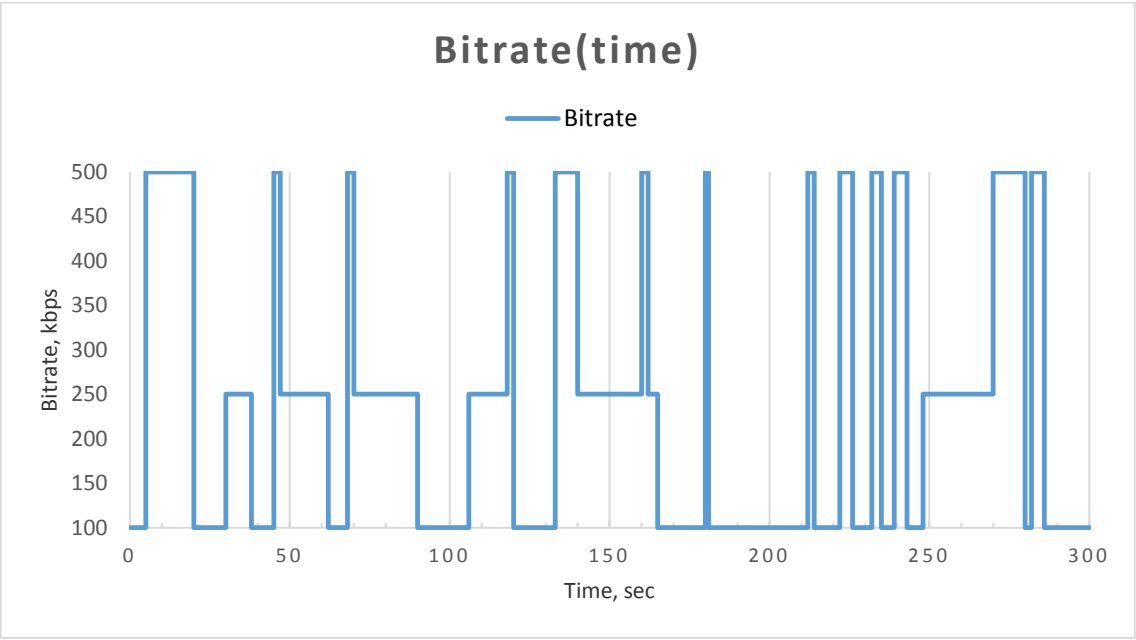


Figure 8.3 Bitrate changes for 12 nodes in the network.

As we can see from the figures 8.1 - 8.3, bitrate adaptation works well. Player's logic might be better because in last case quality hops between extremes (100-500 kbps) when it can be in the middle (175-250 kbps). Than can reduce a little user experience. However, technology always improves, e.g. DASH-IF reference player changes it version from 1.5.1 to 2.1.1 within six months while doing this master's thesis.

9 CONCLUSION

Model of HTTP adaptive video streaming was successfully created. Firstly, was created a wired and a wireless network, which implementation is close to real life and is very scalable, almost everything can be changed from command line: the number of nodes, speed of the channels, the length and the quantity of packets etc.

Then, a video was encoded according to the modern MPEG-DASH technology. Ran a video server on VM, so the client can watch and analyze videos streaming through simulated network. From Figure 6.2, in wired network with 5 Mbps channel and up to 45 nodes video can be watched. From Figure 6.3, in the wireless network can be used up to 10 nodes before delays appear.

For both wireless and wired networks it is better to use 5-8s long segments. However, for wireless video streaming is better to a little longer segments - 8-10s, because of it nature.

In the part number 8 is shown how video adapts for network conditions. It works well, so client can get video almost everywhere, despite network conditions. It significantly increases user experience of using video service.

Created system can be used not only for purposes described in this work, but for any other research goals, where the scalable network can be useful.

REFERENCES

[1] KASIANENKO, D. *Model adaptivního streamování videa přes HTTP*. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Ústav radioelektroniky, 2016. 20 s., 1 s. příloh. Semestrální práce. Vedoucí práce: Ing. Martin Slanina, Ph.D

[2] DASH Industry Forum. (2016). [online] Dashif.org. Available at: http://dashif.org/about/ [Accessed 1 May 2016].

[3] MUELLER, C. (2015). MPEG-DASH vs. Apple HLS vs. Microsoft Smooth Streaming vs. Adobe HDS. [online] Bitmovin. Available at: https://bitmovin.com/mpeg-dash-vs-apple-hls-vs-microsoft-smooth-streaming-vs-adobe-hds/ [Accessed 1 May 2016].

[4] SEUFERT, M. , EGGER, S. ; SLANINA, M. ; ZINNER, T. ; HOBFELD, T. ; TRAN-GIA, P. *A Survey on Quality of Experience of HTTP Adaptive Streaming.* [online]. Wurzburg, Germany: Inst. of Comput. Sci., Univ. of Wurzburg, Available at: http://ieeexplore.ieee.org/xpl/articleDetails.jsp?tp=&arnumber=6913491 [Accessed 1 May 2016]

[5] ISO/IEC 23009-1:2014 - Dynamic adaptive streaming over HTTP (DASH) (2016). [online] ISO. Available at: http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=65274 [Accessed 1 May 2016].

[6] Nsnam.org. (2016). What is ns-3 [online] Available at: https://www.nsnam.org/overview/what-is-ns-3/ [Accessed 8 May 2016].Key Technologies. [online] Available at: https://www.nsnam.org/overview/key-technologies/ [Accessed 8 May 2016].

[7] LACAGE, M. *Experimentation with ns-3*, (2009) [online] Available at: http://typo3.trilogy-project.eu/fileadmin/publications/Other/Lacage-NS3.pdf [Accessed 8 May 2016].

[8] Nsnam.org. (2016). Key Technologies. [online] Available at: https://www.nsnam.org/overview/key-technologies/ [Accessed 15 May 2016].

[9] Linux Containers. (2016). [online] Linuxcontainers.org. Available at: https://linuxcontainers.org/ [Accessed 1 May 2016].

[10] GCC, the GNU Compiler Collection. (2016). [online] Gcc.gnu.org. Available at: https://gcc.gnu.org/ [Accessed 6 Mar. 2016].

[11] PCRE - Perl Compatible Regular Expressions. (2016). [online] Pcre.org. Available at: http://www.pcre.org/ [Accessed 6 Mar. 2016].

[12] OpenSSL Foundation. (2016).*OpenSSL*. [online] Openssl.org. Available at: https://www.openssl.org/ [Accessed 6 Mar. 2016].

[13] CheckInstall. (2016). [online] Asic-linux.com.mx. Available at: http://asic-linux.com.mx/~izto/checkinstall/ [Accessed 6 Mar. 2016].

[14] nginx. (2016). [online] Nginx.org. Available at: http://nginx.org/ [Accessed 6 Mar. 2016].

[15] Dash-Industry-Forum/dash.js. (2016). [online] GitHub. Available at: https://github.com/Dash-Industry-Forum/dash.js/wiki [Accessed 6 Mar. 2016].

[16] curl and libcurl. (2016). [online] Curl.haxx.se. Available at: https://curl.haxx.se/ [Accessed 6 Mar. 2016].

[17] Asynchronous event driven framework. (2016). *Node.js*. [online] Nodejs.org. Available at: https://nodejs.org/ [Accessed 6 Mar. 2016].

[18] Grunt: The JavaScript Task Runner. (2016). [online] Gruntjs.com. Available at: http://gruntjs.com/ [Accessed 6 Mar. 2016].

[19] GPAC. (2016). [online] Gpac.wp.mines-telecom.fr. Available at: https://gpac.wp.mines-telecom.fr/home/ [Accessed 6 Mar. 2016].

[20] Guidelines for Implementation: DASH-IF Interoperability Points V3.2 [online] Available at: http://dashif.org/wp-content/uploads/2015/12/DASH-IF-IOP-v3.2.pdf [Accessed 15 May 2016].

[21] VideoLAN - x264, H.264/AVC encoder. (2016). [online] Videolan.org. Available at: http://www.videolan.org/developers/x264.html [Accessed 6 Mar. 2016].

[22] MPEG-DASH content generation using MP4Box and x264. [online] Available at: http://www.dash-player.com/blog/2014/11/mpeg-dash-content-generation-using-mp4box-and-x264/ [Accessed 6 Mar 2016].

[23] Core functionality. (2016). [online] Nginx.org. Available at: http://nginx.org/en/docs/ngx_core_module.html [Accessed 6 Mar. 2016].

[24] Module ngx_http_core_module. (2016). [online] Nginx.org. Available at: http://nginx.org/en/docs/http/ngx_http_core_module.html#http [Accessed 6 Mar. 2016].

[25] DASH Valitador (2016). [online] Available at: http://dashif.org/conformance.html [Accessed 6 Mar. 2016].

[26] ISO/IEC 23009-1:2014 - Information technology -- Dynamic adaptive streaming over HTTP (DASH) -- Part 1: Media presentation description and segment formats . (2016). [online] Iso.org. Available at: http://www.iso.org/iso/home/store/catalogue_ics/catalogue_detail_ics.htm?csnumber=65274 [Accessed 26 Mar. 2016].

[27] Conceptual Overview — Tutorial. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/docs/tutorial/html/conceptual-overview.html#a-first-ns-3-script [Accessed 13 Mar. 2016].

[28] iostream - C++ Reference. (2016). [online] Cplusplus.com. Available at: http://www.cplusplus.com/reference/istream/iostream/ [Accessed 13 Mar. 2016].ns-3:

[29] ns3::CommandLine Class Reference. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/doxygen/classns3_1_1_command_line.html [Accessed 13 Mar. 2016].

[30] RealTime — Manual. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/docs/manual/html/realtime.html [Accessed 13 Mar. 2016].

[31] ns-3: ns3::Application Class Reference. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/doxygen/classns3_1_1_application.html [Accessed 13 Mar. 2016].

[32] Random Variables — Manual. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/docs/manual/html/random-variables.html [Accessed 13 Mar. 2016].

[33] NS-3: Tap Bridge Model. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/docs/release/3.9/doxygen/group___tap_bridge_model.html [Accessed 13 Mar. 2016].

[34] Tweaking — Tutorial. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/docs/tutorial/html/tweaking.html#using-command-line-arguments [Accessed 20 Mar. 2016].

[35] Building Topologies — Tutorial. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/docs/tutorial/html/building-topologies.html#building-a-wireless-network-topology [Accessed 20 Mar. 2016].

[36] ns-3: ns3::YansWifiChannelHelper Class Reference. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/doxygen/classns3_1_1_yans_wifi_channel_helper.html#a9a7fb65955fd9eb046600b89292e8062 [Accessed 20 Mar. 2016].

[37] ns-3: ns3::AarfWifiManager Class Reference. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/doxygen/classns3_1_1_aarf_wifi_manager.html#details [Accessed 20 Mar. 2016].

[38] ns-3: ns3::GridPositionAllocator Class Reference. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/doxygen/classns3_1_1_grid_position_allocator.html [Accessed 20 Mar. 2016].

[39] ns-3: ns3::ConstantPositionMobilityModel Class Reference. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/doxygen/classns3_1_1_constant_position_mobility_model.html [Accessed 20 Mar. 2016].

[40] LTE Module — Model Library. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/docs/models/html/lte.html [Accessed 3 Apr. 2016].

[41] ns-3: ns3::EpcHelper Class Reference. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/doxygen/classns3_1_1_epc_helper.html [Accessed 3 Apr. 2016].

[42] ns-3: ns3::Ipv4StaticRouting Class Reference. (2016). [online] Nsnam.org. Available at: https://www.nsnam.org/doxygen/classns3_1_1_ipv4_static_routing.html [Accessed 3 Apr. 2016].

[43] MSE support in browsers (2015). [online] Caniuse.com. Available at: http://caniuse.com/#search=mse [Accessed 26 Mar. 2016].

[44] Dashif.org. (2016). Test vectors | DASH Industry Forum. [online] Available at: http://dashif.org/test-vectors/ [Accessed 9 May 2016].

[45] Lxde.org. (2016). LXDE.org | Lightweight X11 Desktop Environment. [online] Available at: http://lxde.org/ [Accessed 9 May 2016].

GLOSSARY

AARF	Adaptive Auto Rate Fallback.
AP	Access Point.
API	Application Programming Interface.
CSMA	Carrier Sense Multiple Access.
DASH	Dynamic Adaptive Streaming over HTTP.
EME	Encrypted Media Extensions.
eNodeB	Evolved Node B.
EPC	Evolved Packet Core.
GCC	GNU Compiler Collection.
HTTP	HyperText Transfer Protocol.
IF	Industry Forum.
IP	Internet Protocol.
LAN	Local Area Network.
LTE	Long-Term Evolution.
LXC	Linux Containers.
MAC	Media Access Control.
MIME	Multipurpose Internet Mail Extensions.
MPD	Media Presentation Description.
MSE	Media Source Extensions.
P2P	Point-to-Point.
PCRE	Perl 5 Compatible Regular Expression Library.
PDN	Packet Data Network.
PGW	PDN Gateway.
PRNG	Pseudo-Random Number Generator.
QoS	Quality of Service.
SSID	Service Set Identifier.
SSL	Secure Sockets Layer.
STA	Station.
TCP	Transmission Control Protocol.
UDP	User Datagram Protocol.
UE	User Equipment.
URI	Uniform Resource Identifier.

URL	Uniform Resource Locator.
VM	Virtual Machine.
VBV	Video Buffering Verifier.

LIST OF ANNEXES

A 3 DVDs

System named “test” splits into 3 DVDs attached to this work. Also, electronic version of this work is on first DVD. After decompressing system can be run in VirtualBox. User name and password are “testuser”. On desktop situated file “readme” where can be found step-by-step instruction how to run the simulation and locations of required files.