Comparative study of metrics for IPTV transport in the access network

Patrik Björkqvist

Department of Electrical and Information Technology
Lund University

Advisors:
Jens A Andersson at *EIT*
Stefan Höst at *EIT*
Daniel Cederholm at *Ericsson AB*

November 22, 2011

Printed in Sweden
E-huset, Lund, 2011

Abstract

Last few years, services such as *Internet Protocol Television* (IPTV) and *Voice over Internet Protocol* (VoIP) have been in focus. Network operators see a huge potential in streaming real-time media over IP-networks and this lays the ground for so called paid services such as *Video on Demand* (VOD) as well as free services like YouTube and SVT Play. IPTV delivers content over a closed infrastructure that receives media from a particular provider using a *set-top-box*, or STB for short, in the home of customers. Services such as *Internet TV* (SVT Play, TV3 Play and YouTube for example), have the same look and feel as IPTV but are delivered over the open infrastructure of the internet, relying on the best-effort channel. The latter is called *over-the-top* or OTT , referring to the media going around the STB rather than through it. The benefit of such a service can be put into one word, flexibility, watch what you want, when you want and where you want. But these type of services have some drawbacks. Problems such as packet loss, delay and jitter in the network greatly affects the quality for the end-user.

A significant problem for OTT-traffic in the access network, when dealing with DSL copper transmission, are the old telephone cables used for the last mile to customers, this part of the network was not built for this kind of traffic and suffers from external disturbances in form of $50Hz$ power lines, crosstalk and radio frequency interferences. This paper aims to evaluate and compare different methods for classifying network traffic and how *Quality-of-Service* (QoS) parameters (packet loss, delay, jitter, out-of-sequence) on the IP level affect *Quality-of-Experience* (QoE) from a head-end to a receiver in a home network connected to an access network. A simulated IPTV transmission is done in a simulated network with and without an external disturbance signal injected to a pair in the same cable.

Acknowledgements

First of all, I want to show my respect and gratitude to my supervisor Jens A. Andersson and my examiner Stefan Höst, both working in the department of Electrical and Information Technology at Lund University of Technology. It has been a privilege to work with you during this time, thanks for your support and knowledge. I also want to show my respect and gratitude to Ericsson Research in Kista and especially Daniel Cederholm who, despite his regular work, had the time to give comments and feedback throughout the process. It has been a fantastic experience to see and get this close to your work at Ericsson AB, and it has really inspired me to continue on the same path in my professional career. Last but not least, I thank my family, without your support and help I would not have made this journey, a special thanks to my daughter Emely who put up with my long days at school and late evenings at home.

Terminology

Throughout this paper a set of terminology is used. An explanation of the most common terms are given below:

- Internet Protocol TeleVision (IPTV) - IPTV delivers video content over a closed secure infrastructure that can only receive content from the IPTV provider' s channels. IPTV focuses primarily on the TV in the living room with high image quality.
- Internet TV - Internet TV, which has the same look and feel as IPTV, is delivered over the open best-effort channel as OTT-traffic. Internet-TV is usually delivered to the PC or some other device using peer-to-peer technology.
- Over-The-Top (OTT) - Is referring to traffic in an IP-network not going through an end customers set-top box.
- Quality of Experience (QoE) - Quality of Experience is a subjective measure of a customer's experience with a service (web browsing, phone call, TV).
- Quality of Service (QoS) - Quality of Service is a set of parameters (jitter, delay, latency and packet-loss) indicating the status of a traffic flow handled by a network.
- ADSL - Asymmetric Digital Subscriber Line, part of the xDSL family and is a technique for transmitting information over copper cables.
- DSLAM - Digital Subscriber Line Access Multiplexer is a network device and connects multiple customer DSL interfaces to a high-speed communication channel.
- DMT - Discrete Multi Tone is a method of separating a DSL signal into 256 (in ADSL) sub-channels or so called tones.

- QAM - Quadrature Amplitude Modulation - a method of combining two amplitude modulated signals into one signal.
- PAM - Pulse Amplitude Modulation is a form of signal modulation where the information is placed in the amplitude.
- FDM - Frequency Division Multiplexing, is a form of signal multiplexing and involves assigning non-overlapping frequencies to different users.
- RTP - Real time Transport Protocol, used for data transmissions in realtime such as audio and video.
- RTCP - Real time Transport Control Protocol provides control information for an RTP flow.
- RTSP - Real Time Streaming Protocol, used for establish and synchronize media transmissions using UDP or RTP.
- IP - Internet Protocol is the communication protocol used for relaying data packets over an network such as Internet.
- UDP - User Datagram Packet, is a connectionless protocol used for sending datagram over an IP-network.
- TCP - Transport Control Protocol, a connection oriented protocol used for reliable delivery of data streams.

Table of Contents

List of Figures

List of Tables

Chapter 1

Chapter Overview

Chapter 1 describes the structure of this thesis and briefly explains the content in each chapter. Chapter 2 gives an introduction for this paper and explains the motivation of this research. In Chapter 3, a theoretical background of the PSTN and cable theory is given, this Chapter also briefly explains some of the disturbances that appear in the network and especially in cables used for xDSL transmissions. This is followed by classification methods, probing techniques and the subject of mapping QoS parameters to expected QoE for end-users in Chapter 4. In Chapter 5, a walkthrough is done regarding the work of the signal generator made in C# used to simulate a disturbance signal in the frequency range of ADSL. Finally in Chapter 6 the thesis ends with conclusions and some ideas for future work.

Chapter 2

Introduction and Motivation

Time sensitive media, such as video and audio, is steadily increasing on the internet. What was once divided into separate telephone-, television- and computer-networks are today more or less combined and run on the IP-network we call the internet. One huge problem for both end-customers and operators for different services is that the IP-network was not built for time sensitive media. When it comes to audio/video traffic it can be divided into two categories, the traffic going through a set-top-box in the home of end-users, and the traffic going around the end-users set-top-box. The traffic to the set-top-box is most often a service paid for by end-customers and it has its own logical channel and is not the focus of this report. But the traffic running over-the-top in the so called best-effort channel is trickier.

The QoS parameters (e.g. delay, jitter, packet-loss) are connected to the quality of the traffic and in the end QoE for customers. In an attempt to investigate and understand how external disturbances affect QoS parameters in xDSL lines and how to classify network traffic, this thesis is made in collaboration between the department of Electrical and Information Technology in Lund and Ericsson Research in Stockholm. Studies in recent years have been made on different types of disturbances affecting the IP-traffic in the access network, one effect discovered is the impact of radio interference from distant radio stations transmitting in the medium frequency range, also called *medium wave*. The old telephone cables used for xDSL traffic are sensitive for external disturbances and can therefore have a great impact on QoE. In the attempt of maintaining or even improve QoE for customers the QoS parameters for that particular network traffic must be analyzed, traffic classification is a way of respond to this challenge. This thesis aims to theoretically analyze how to probe and classify networks and the traffic flowing within it, in the future with an enormous amount of time sensitive media the classification and control of that flow will be of great importance. The network operators must be able to provide benefits for the traffic of paying customers as well as limit the traffic that is free of charge.

To be able to evaluate both QoS and QoE correlation and probing a simulated ADSL network is set-up in the lab in Lund, a controlled disturbance signal is generated to affect the ADSL line in a controlled way.

Chapter 3
Background

3.1 Related Work

3.1.1 Network Architecture

The *Public Switched Telephone Network* (PSTN) is a world wide circuit switched network delivering telephone services to the public and is the part most think of when talking in terms of telephone systems. Private networks such as *Private Branch Exchange* (PBX) are not a part of the PSTN although they most often are connected to PSTN. The network can be divided into two major parts:

- **Core Network** - This is the central part of the PSTN and provides various services to all who are connected to the core by the access network, the core network is also referred to as the Backbone network. The main function is to route traffic across the PSTN and to provide a path from different sub-networks.
- **Access Network** - The access network is the part which connects subscribers to their service provider, that is the cable between a customer and the telephone exchange server. This part is most often old copper wires and is known as *The last mile*. This is also the part of interest in this thesis since it is here a lot of the problems on hand arises.

3.1.2 Cable Theory

Unshielded twisted-pair cable is the most common type of cable used today in telecommunication. It is mostly associated with voice transmissions but handles frequencies well in the range of $100kHz$ and $200MHz$ and is well suited for data traffic as well. The telephone line consists of two conductors, usually copper, each with its own insulation. In the early days of

telecommunications, two parallel flat wires were used but suffered a lot from electromagnetic interference. This is because the wire closest to the source of noise receives more interference than the one further away, this results in an uneven load on the cable and distorts the signal as depicted in Figure 3.1. In the twisted cable, the distance to the noise is alternated between the two wires making the total effect to zero, this can be seen in Figure 3.2. This solution does not eliminate the impact of noise, but reduces it. There are five major disturbances where inductive noise and wave interference are of special importance in this thesis:

1. Galvanic disruption - When two circuits use a common cable, the influence of galvanic shocks can occur. This is often a common reference or power supply wires. By current or voltage fluctuations in the first circuit, the second one is affected.

2. Capacitive interference - Is caused by a system's electrical field, which acts as a source of interference. A typical example of a capacitive disturbance are two, over a longer distance, parallel buried cables. These acts as two opposing horizontal capacitor plates and creates a short circuit for high frequency signals.

3. Inductive noise - The reason for an inductive interference is the magnetic field formed around a current, which also penetrates the adjacent conductors. A power change leads to a change in the magnetic field, after which a voltage is induced in the conductor between neighboring cables.

4. Wave interference - Means the transfer of conducted waves or impulses to the adjacent lines, with the right frequency of the wavelength compared to the cable. The whole cable can start to act as a receiving antenna if no countermeasures are taken.

3.1.3 Digital Subscriber Line

The *Digital Subscriber Line* or DSL for short, is a family of technologies that provides high-speed transmissions of data over the existing PSTN cables. Commonly used techniques in Sweden are ADSL2, ADSL2+ and VDSL2 is in progress, other techniques are ADSL, SDSL and VDSL. They are most often referred to as xDSL as a whole. In this thesis we refer to the first two techniques of ADSL which stands for *Asymmetric Digital Subscriber Line*. It provides higher bit rate in the downstream than the upstream, from the customers point of view. ADSL divides the bandwidth of a twisted-pair cable into three bands as seen in Figure 3.3. The first band, normally

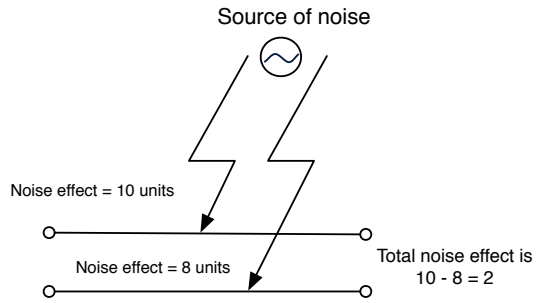


Figure 3.1: Effects on un-twisted pair cable.

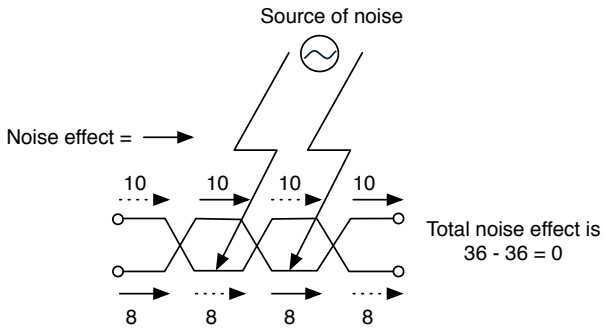


Figure 3.2: Effects on twisted pair cable.

between 0 and 25 kHz, is used for regular telephone service and only uses around 4 kHz, the rest is used as guard band. The second band is between 25 kHz to 200 kHz and is used for the upstream, the third and last band is used for the downstream and is between 250 kHz and 1.1 MHz.

The modulation used in ADSL is called *Discrete Multi Tone* (DMT) and combines QAM and FDM. ADSL and ADSL2 each have 256 sub channels while ADSL2+ has 512 sub channels and extends the down band to 2.2 MHz. The bandwidth is divided into bins of 4.3125 kHz and each sub channel within a specific frequency range will be responsible for either up- or down-stream data as seen in Figure 3.4. In DMT a technique called *bit-swapping* is used, changes in the channel gain or noise spectrum are tracked by a DMT-based DSL modem and moves bits between sub-channels in order to maintain the target bit error rate (BER). Typically bit-swapping retains the same data rate but redistributes energy so that best (and lowest neces-

sary power) bit distribution is maintained. Bit swapping allow up to at least a 14 dB range of noise variation and is used when single tones are affected, however, some noises may exceed this range and then re-initialization is necessary.

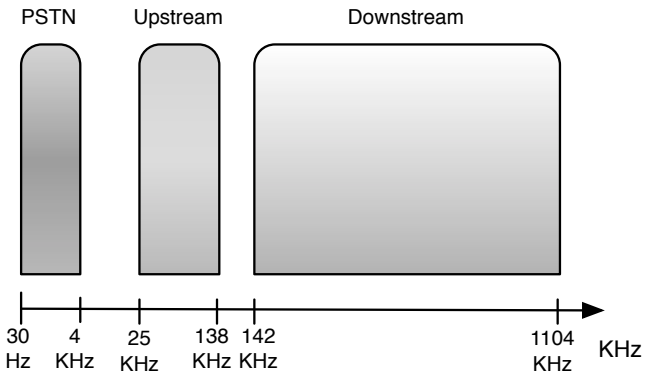


Figure 3.3: ADSL frequencies.

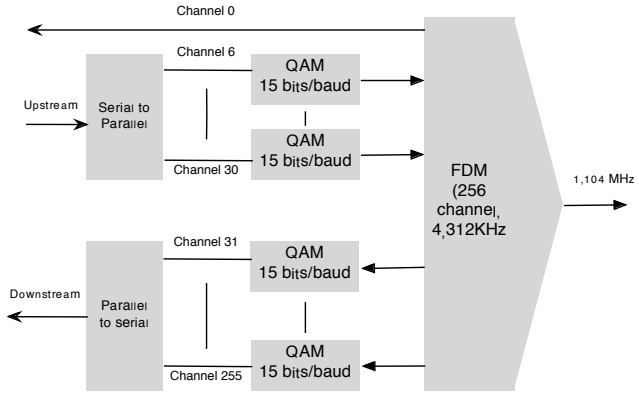


Figure 3.4: Structure of DMT.

3.1.4 Radio Transmission

Radio technology is out of the scope of this thesis but to understand how electromagnetic waves can disturb cables such as the ones used for ADSL

traffic, a short introduction is made. Electromagnetic radiation is a form of energy with wave-like behavior when it travels through space and consists of both an electric and magnetic field component, which oscillate in phase perpendicular to each other and perpendicular to the direction of energy propagation. Radio waves are in the range of 3 kHz to 300 GHz as seen in table 3.1. Problem arise when radio waves start to interfere with electronic equipment and cables both far away and near the transmitter. Due to atmospheric changes radio waves can sometimes travel far distances so both nearby transmitters as well as distant transmitters can interfere.

3.1.5 RTP Protocol

RTP stands for *Real time Transport Protocol* and is widely used for real time multimedia data streams such as audio and video and provide end-to-end delivery services. RTP is used in conjunction with *RTP Control Protocol* (RTCP), while RTP is used to carry the actual media stream (e.g. audio/video), RTCP is used to monitor transmission statistics and QoS. When these protocols are used in conjunction, RTP is originated and received on even port numbers and RTCP on the next higher odd port number. RTP is also used in conjunction with other protocols such as *RTSP*. RTSP, or *Real Time Streaming Protocol*, is a network control protocol used for controlling streaming media servers for entertainment and communications systems. The transmission of streaming data itself is not a task of the RTSP protocol, thats the job of RTP. The following differentiates between these protocols:

- **Real Time Streaming Protocol(RTSP)** - Is the control protocol for the delivery of multimedia data over any IP-network, it typically uses TCP and has very similar operations as HTTP. RTSP is used by the client to communicate information, such as the media file requested, type of application on client side, if unicast or multicast, TCP or UDP, to the server. Also control commands such as *SETUP* and *PLAY* are sent by this protocol.

- **Real-time Transport Protocol (RTP)** - RTP is the protocol used for the actual transport and delivery of real time data, such as audio/video. UDP is used as the layer 4 mechanism for delivery, TCP however, can also be used in situations where packet loss is higher. The RTP flow is unidirectional when delivering data, that is from the server to the client. As mentioned before, the source port is always even when sending the UDP data and the port number is dynamically allocated. The destination port is chosen by the client and communicated over the RTSP connection.

Frequency	Wavelength	Designation	Abbreviation
3 - 30 Hz	10^5 km - 10^4 km	Extremely Low Frequency	ELF
30 - 300 Hz	10^4 km - 10^3 km	Super Low Frequency	SLF
300 - 3000 Hz	10^3 km - 100 km	Ultra Low Frequency	ULF
3 - 30 kHz	100 km - 10 km	Very Low Frequency	VLF
30 kHz - 300 kHz	10 km - 1km	Low Frequency	LF
300 kHz - 3 MHz	1 km - 100 m	Medium Frequency	MF
3 MHz - 30 MHz	100 m - 10 m	High Frequency	HF
30 MHz - 300 MHz	10 m - 1 m	Very High Frequency	VHF
300 MHz - 3 GHz	1 m - 10 cm	Ultra High Frequency	UHF
3 GHz - 30 GHz	10 cm - 1 cm	Super High Frequency	SHF
30 GHz - 300 GHz	1 cm - 1 mm	Extremely High Frequency	EHF

Table 3.1: Radio Spectrum

- **Real-time Control Protocol (RTCP)** - RTCP is a complimentary protocol for RTP and is a bidirectional UDP mechanism to communicate QoS back to the server. The RTCP UDP communication always uses the next source port up from the one used by the RTP stream, and thereby always odd.

A picture of the basics in the process of RTP, RTCP and RTSP can be seen in 3.5 and are explained in the following steps:

1. Client establishes a TCP connection to the server, usually on port 554 which is a well known port for RTSP
2. Client then commences a series of RTSP header commands describing to the server details of the session requirements, such as supported RTSP versions, transport for the data and port information. This is passed using the *DESCRIBE* and *SETUP* headers.
3. When the previous step is done the client will issue a *PLAY* command to tell the server to commence delivery of the RTP data.
4. When the client decides to close the stream of data, a *TEARDOWN* command is issued making the server to cease the RTP stream with that particular session.

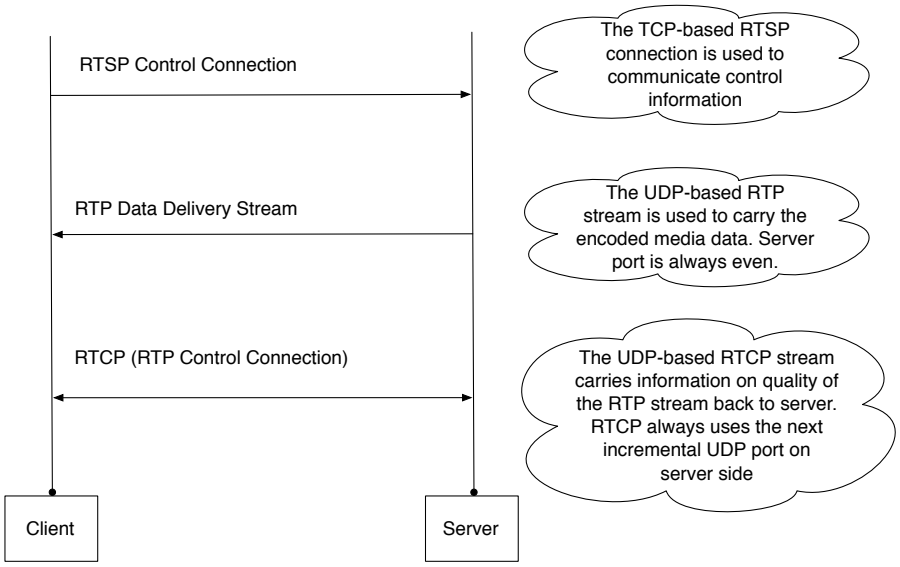


Figure 3.5: RTP, RTCP and RTSP in real-time streaming.

Chapter 4
Theory

The main goal here is not to investigate how operators can, shall or will handle the process of quality, but for simplicity a flowchart called *Process of Active Experience and Service assurance* (PAXSa) is shown in Figure 4.1. This is only done to give a survey over this broad topic. As seen in previous chapter, multimedia traffic on the internet can use the old PSTN network and suffers from disturbances. Because of this it is important for operators to be able to perform measurements on the network and based on the outcome take action. Probing can be done in many ways and depends on the reason for doing it. Active probing can give QoS parameters for an end-to-end link while passive probing typically gives QoS parameters at a point in the network. One reason for probing can be that operators wish to prevent problems by continuously probing the desired network and take countermeasures based on the result, a second reason can be customers complaining of poor quality for different services.

4.1 Probing Methods

Many proposals have been made for service providers to measure a particular path in the network and determine the quality of that path. Active and passive measurements are two fundamental techniques of measuring QoS parameters such as jitter, delay and packet loss in an IP-network. These techniques differ quite considerably in the procedure. Active probing injects traffic on the network which is then observed, while passive probing passively examines the traffic that is already flowing on the net. It is important to highlight that in active probing it is the injected traffic that actually carries the QoS parameters, while in passive probing, the QoS parameters are taken from the network traffic.

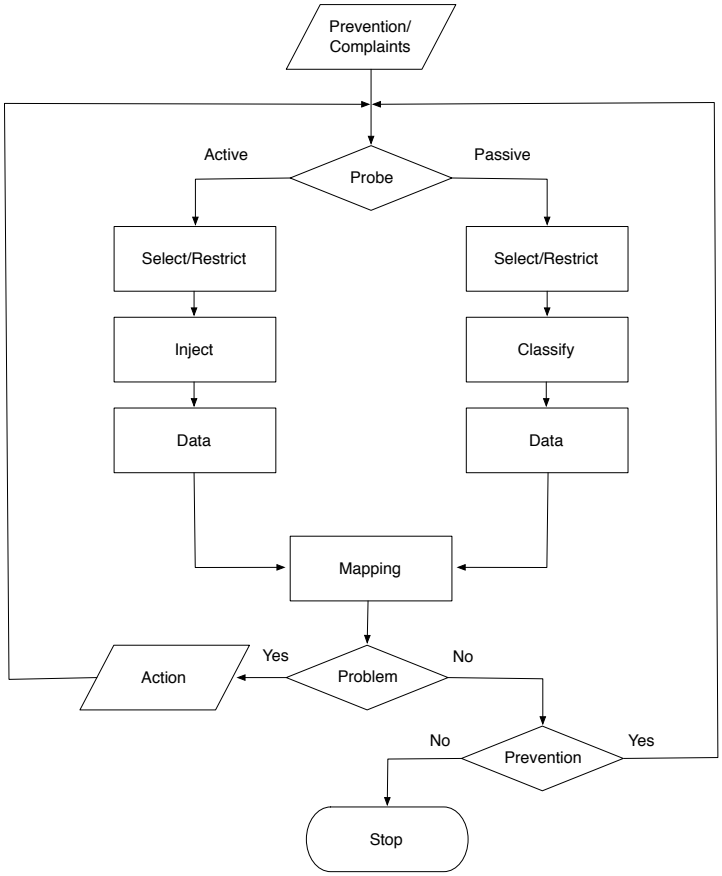


Figure 4.1: Process of Active Experience and Service assurance

4.1.1 Passive Probing

The very first step in the passive approach in PAXSa is *Select/Restrict* and is a pre-step to passive probing. In order to monitor the network in a passive way it is prone to determine which traffic to actually observe, if a particular application or protocol is to be monitored or maybe its enough to monitor some part of the traffic as a whole. The work of actually separating the desirable traffic from the undesirable is called classification and is explained in detail in Chapter 4.2. Passive measurement is a means of tracking performance and behavior of traffic streams in the network by monitoring the traffic without modifying it. The level of detail of the information collected depends on the traffic volume, how the metrics are being processed and of course what metrics are of interest. The passive approach has the bene-

fit of not affecting network traffic in any way and can be of importance if implemented in an already highly loaded network, the passive approach is also used for measurements in a point rather then end-to-end. This can of course be a problem if one does not have access to network devices and/or links. Typical QoS parameters in passive probing can be:

- Packet/Bit rates
- Inter-arrival time
- Queue levels in buffers (Used as an indicator of packet loss and delay)

There exists two common passive probing techniques, port mirroring and network tapping.

Port mirroring, or *Switched Port Analyzer* (SPAN), is the more active approach in the sense that the network device physically has a duty to copy packets from a certain port or ports and forward them not only to the destination port, but also to a mirror port. Since traffic in both directions is copied, problems like buffer overflow and dropped packets may occur when multiple ports are mirrored to one port, making time-sensitive measurements difficult to perform. This process takes some resources from the switch, such as CPU, making the workload lead to reduced switching performance. A basic model of this technique can be seen in Figure 4.2.

Network Tapping is fully passive and is directly inserted onto a link. A network-tap split or copy the signals from both channels and retransmit the data to a monitoring device. Unless an aggregation tap is used, a tap has one tap-port per direction. This means that the network device needs dual interfaces in order to capture a full-duplex line. A basic model of this technique can be seen in Figure 4.3. Since taps split/copy the traffic without interfering, all anomalies are also copied, making it a preferable solution when it comes to troubleshooting. The copper tap does not *split* the signal, it regenerates it. Regenerating the signal means the signal gets amplified to a level where it can be received by the monitor device, this also means that copper-taps need power. When it comes to fibre taps, there are two things to consider, first the *split ratio* of the tap and second the *light source* of the tap. A fibre tap´s split ratio is mainly determined by the sensitivity of the receiver, transmitter strength and cabling. When a fibre is tapped, the link suffers from insertion loss due to the split light beam. The tap however is non-powered. Which technique to use, Mirror or Tap, depends on the reason for executing a passive probe in the first place. The network tap will most likely be more expensive since it requires hardware to be installed, but this solution however give full access of a full duplex link with the ability to investigate physical errors. The port mirror

has the benefit of copying intra-switch traffic but is not the best choice if time sensitive parameters are to be measured. A summery of the techniques pros. and cons. are displayed in Table 4.1

	Network TAP	Port Mirror
Pros	Eliminates the risk of dropped packets	Low Cost
	Monitoring device receives all packets, including physical errors	Remotely configurable from any system connected to the switch
	Provides full visibility into full-duplex networks	Able to copy intra-switch traffic
Cons	Analysis device may need dual receive capture interface if using full-duplex TAP	Can not handle utilized full-duplex links without dropping packets.
	Additional cost with purchase of TAP hardware	Filters out physical layer errors
	Can not monitor intra-switch traffic	Burden placed on a switch´s CPU to copy all data

Table 4.1: Pros and Cons for Port Mirroring and Network TAP

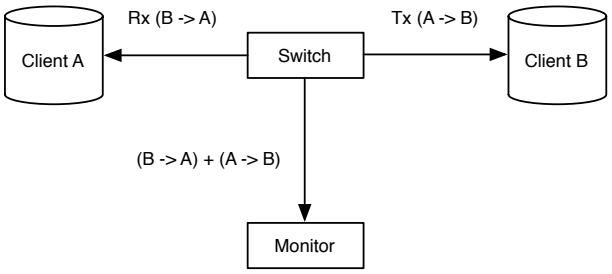


Figure 4.2: Concept of Port Mirroring.

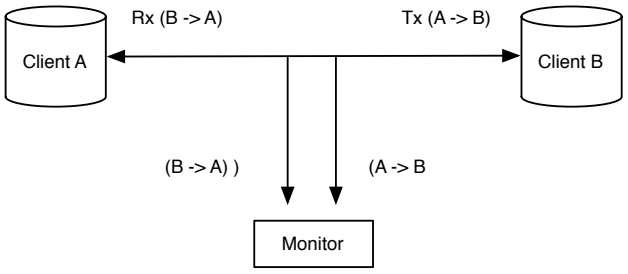


Figure 4.3: Concept of Network Tapping.

In [9] a passive technique of detecting and diagnosing link-level anomalies is suggested to cover as many IP flows as possible in a network. A directed graph $G = (V,E)$, where V is the set of nodes (e.g. routers, switches etc.), and E is the set of bidirectional edges (e.g communication links) is used to model the network. The infrastructure consists of passive monitoring devices, placed at different points in the network, and a *Network Operations Center* (NOC). A variety of passive technologies are available to observe network traffic, both on the link and on the link interface, such as *Network Taps* and *Port mirroring*.

- **Detecting anomalies** - As mentioned above, a tap on a path p can be used to detect packet loss and/or delays for links on this particular path. Let s and t be two taps at either end of the path p, at predefined time intervals both s and t send the number of packets passed to NOC. If the difference exceeds some specified threshold, we can conclude that packets are being lost somewhere on the path. One other alternative is to let s and t send samples of the observed traffic to the NOC and then inference of packet loss can be made if the discrepancy is large. By associating timestamps with packets it is possible to, in a similar way as above, detect delays for the path p. As long as every link belongs to at least one monitored path, it is possible to detect link-level anomalies. In order to reduce communication overhead between probes (e.g passive monitoring devices), it is beneficial to monitor as few paths as possible while every link is covered by at least one monitored path. In Figure 4.4, an example of a Service Provider network is depicted, nodes a, b, c and d are so called edge nodes which are connected to some customer networks. Now suppose that there are two bidirectional communication paths, $p_1 = \langle a, v_1, v_2, b\rangle$ and $p_2 = \langle c, v_1, v_2, d\rangle$. Monitoring taps are placed at nodes a, b, c and d, so the network traffic on the above two paths can be monitored.

- **Diagnosing anomalies** - While paths p_1 and p_2 are enough for detecting anomalies on the links, they are not enough to determine *which* link is having problem. Say for example that path p_1 reports an anomaly but not path p_2, then either of the links $\langle v_2, b\rangle$ or $\langle a, v_1\rangle$ could be the cause of the anomaly. $\langle v_1, v_2\rangle$ is not among the suspects since p_2 is not reporting any anomalies. Passive probes at the edge nodes are required to ensure all paths are monitored and all anomalies are detected. For diagnosing however, this may not be enough and additional probes may have to be placed within the core network. In a simplified way we assume that *a path only reports an anomaly if and only if there is an anomalous link in this path*, second we also assume

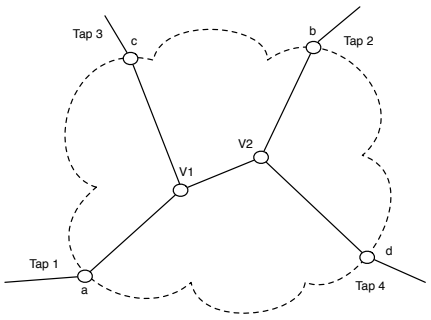


Figure 4.4: Example of Service Provider network.

that *a network anomaly is caused by a single link*. These assumptions can be relaxed according to the authors of [9]. By introducing path $p_3 = \langle a, v_1, v_2, d\rangle$ the path set $Q = \{p_1, p_2, p_3\}$ now can distinguish between all link pairs. If however no communication is in progress over p_3, we are back to the same problem, another way to solve the problem is as mentioned above to place an additional tap/probe in the path segment between the undistinguished link pairs and break each path into two new ones. By placing a tap on link $\{v_1, v_2\}$ the path p_1 splits into path $p_3 = \langle a, v_1, v_2\rangle$ and $p_4 = \langle v_2, b\rangle$, and p_2 splits into $p_5 = \langle c, v_1, v_2\rangle$ and $p_6 = \langle v_2, d\rangle$. The new paths p_3 and p_5 contain just one of the links in the pairs $\{\langle a, v_1\rangle, \langle v_2, b\rangle\}$ and $\{\langle c, v_1\rangle, \langle v_2, d\rangle\}$. By selecting the subset $Q = \{p_1, p_2, p_3, p_5\}$ or $Q = \{p_3, p_4, p_5, p_6\}$ anomaly diagnosis can once again be made.

4.1.2 Active Probing

Active probing is another way of measuring the performance of the network and involves inserting traffic into the network path to be measured. In active probing, the probes actually carry the information regarding jitter, delay and packet loss. The sole purpose of probe packets is to provide insight into how the real network traffic is treated within the network. Active probing has the disadvantage of burdening the network with more traffic, on the other hand it has the benefit of being able to monitor the link from end-to-end without the need of access to network stations in between. According to the model of PAXSa, *Select/Restrict* is the first step in the active option and refers to select and restrict what probes to use. Probes may be generic (specific only to the protocol, not the application), or customized to an expected application. Other choices that have to be made are probe-structure

(e.g. packet-pair, packet-trains), departure distribution (e.g. Poisson) and if probes are to be pre-planned or actively chosen during the process. How to make these selections differs from situation to situation and will not be completely explained in this thesis. However, it is the opinion of the author of this report that there exist three major probe-structures and are mainly used for gathering data such as delay, loss and jitter:

- **Packet Trains** - One of the first models of a traffic flow was created by [4] and is called *packet train* model. It is here defined as a burst of packets arriving from the same source. If the spacing between two packets exceeds some pre-defined inter-train gap, they are said to belong to different trains.
- **Packet-Pair** - The packet-pair is defined as two packets of the same size traveling from the same source to the same destination with a predefined inter packet time.
- **Back-to-Back** - Is defined as a packet-pair but back to back, no time space between packets.

An interesting approach is taken in [5], it is shown that the approach explained can reduce the time and number of probes needed to localize a problem compared with pre-planned probing. The pre-planned approach is rather straight forward technique but suffers from some limitations. Probes are computed offline and need to be able to diagnose all possible anomalies, making the probe-set very large in some cases. Another disadvantage is that the probes run periodically at scheduled intervals, there will be delays in detecting a problem if it happens in between two probes. In this approach, an initial set of probes is selected offline and is run periodically, this is to detect any anomalies in the network. If a problem is accounted additional probes are selected online for the purpose of gathering more information, this step is repeated until the problem is determined. This allows fewer probes to be used than in the case when the set must be determined in advance, an outline of the system is seen in Figure 4.5. The process explains how to select initial probes, analyze probe data and select additional probes.

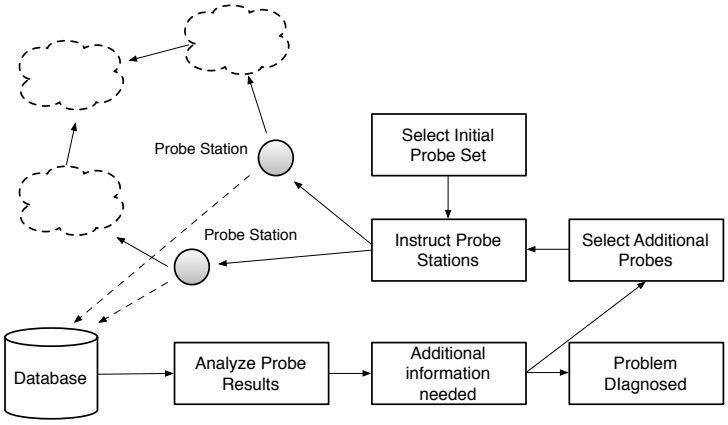


Figure 4.5: Example of an active probing system

- **Selecting Initial Probes** - A dependency matrix can be used in the relationship between available probes and the nodes with a problem needed to be detected. The probes are the rows and the nodes are the columns and a nonzero entry in the matrix where a probe tests the occurrence of a problem in that node. For example, consider the network in Figure 4.6. Now suppose one probe is being sent along the path of N_1, N_2 and N_5 and one probe is sent along the path of N_1, N_3 and N_6. A dependency matrix is shown next to the graph where probes are indexed by start and end nodes. If N_2 is down, then the probe of the first path fails but not the second one, if instead N_5 is down once again the first path fails but not the second one. These two failures result in the same signal because their columns in the matrix are the same. Any problem whose column is unique generates a unique signal making not only problem detecting possible, but also problem diagnosing. The task of **problem detection** however, is to find the smallest set of probes that no matter what problem occurs, there will be a probe failing. This can also be stated as finding the smallest set of probes (e.g rows) such that each column has a nonzero entry.

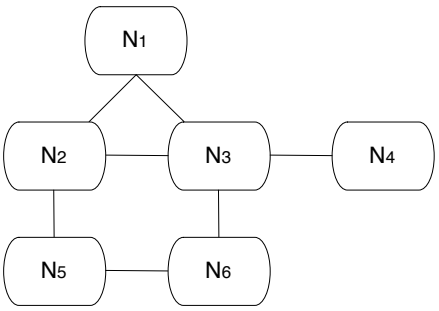
- **Analyzing Probe Results** - When a problem occurs, the probe results must be analyzed. Diagnosis can be seen as the task of finding the *most likely* state to all network components given by probe results. If we assume a prior probability of fault equal to $p_i = P(X_i = 0)$ for all nodes, we then wish to find a vector $x^* = \arg\max_{x_1,\ldots,x_n} \prod_{j=1}^{n} p_j$

matching to those constraints imposed by observed probes. The problem can also be seen as a constraint satisfaction rather then optimization, if there is a unique solution satisfying the constraints.

- **Selecting Additional Probes** - At every stage, additional probes must be selected according to the previous probe results. For each probe the following can be computed:

  1. Likelihood of the probe succeeding or failing, which depends on inferences drawn about the probability of different network states.
  2. Additional information about the network as a result of sending that probe and receiving a successful or failed result.

  By using this strategy one can compute the expected information gain of each node and which node to send to next, to maximize the expected information gain. Once the *next to send*-probe is selected, it is sent and once again inferences are made about the network state,if necessary additional probes are sent again.



	N_1	N_2	N_3	N_4	N_5	N_6
P_{15}	1	1	0	0	1	0
P_{16}	1	0	1	0	0	1

Figure 4.6: Example of a dependency matrix

Both [6] and [7] show that longer probes (packet trains) result in better estimates regarding bandwidth (data rate) in the measured network. In [6], the authors come to the conclusion that shorter probe trains introduce bias errors, while the work of [7] points out a problem regarding bandwidth estimation with probe trains in the length of three to seven packets. The problem occurs when switching from looking at the order of sent packets

to the order of arrived packets, a possible reason for this can depend on reordering of packets. In active probing, there are several parameters to consider such as packet length, departure time and inter-packet gap, even how the network treats the train is important. If the purpose is to estimate bandwidth, a train in the range of seven to fifteen packets with a Poisson departure is often suggested, but if the purpose is to investigate how the network treats a certain application or a certain class of traffic, the train should have similar properties as that particular application or traffic. Most importantly, the packet trains will interfere with the rest of the traffic, so called cross-traffic. It is important to understand how the interaction between cross-traffic and packet-trains works and will be discussed in the next section.

Cross Traffic Effects

When the packet train traverses the network path, the dispersion between successive probe packets will change. This is due to limited link capacity, interactions with other packets traversing the same path (so called cross-traffic packets) and disturbances. In [8], a model is used to describe and analyze how packet trains are affected by cross-traffic when traversing a network. Three major effects are identified as *Chain patterns*, *Quantification patterns* and *Mirror patterns*. The definition of a *hop* here is the router, its in-queue and the outgoing link. It is assumed all queues are FIFO-queues and there is no isolation of flows, for example fair queuing. The dispersion of adjacent packets in a packet train is equal, when leaving the probing generator. This dispersion is varied to achieve different probe rates and the packet size is fixed. The authors also define what they call **I** and **B** packets, an **I** packet is never queued behind another probe packet in the outgoing queue, while **B** packets are.

- **Chain Pattern** - is the effect of cross traffic slow a probe packet (i) in such way that at least $(i+1)$ and $(i+2)$ are transformed from **I** packets to **B** packets and are shown in Figure 4.7. The shaded packet is a cross-packet arriving just before probe packet $(i-1)$, they arrive on different links but are assumed to leave on the same. Packet $(i-1)$ must wait for the router to process the cross traffic before it can leave. This also make packet (i) and $(i+1)$ delayed, when leaving the last two packets have become **B** packets since they queue behind packet (i) on the outgoing link.
- **Quantification patterns** - When a cross traffic packet arrives at the incoming queue between two back-to-back probe packets, they will be separated by the service time of the cross packet as seen in

Figure 4.8. The second one entering between probe packet $(i-1)$ and (i) create a time gap and the quantification pattern is a fact.

- **Mirror patterns** - The mirror pattern arises when a packet train consists of at least three **I** packets $(i-1)$, (i) and $(i+1)$. The delay for an **I** packet is described by following equation:

$$\delta_i = (x_i - x_{i-1}) + (w_i - w_{i-1})$$

Where x is service time and w is waiting time in queue. The mirror pattern arises when probe packet (i) is affected by a cross traffic packet but not probe packet $(i-1)$ and $(i+1)$ as seen in Figure 4.9. While not going into details, the *mirror* effects appear in the delay for packet $(i+1)$ and (i) and are $\delta_{i+1} = -\delta_i$.

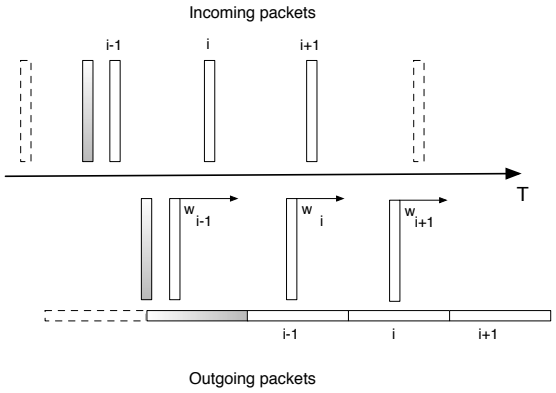


Figure 4.7: Chain pattern.

4.1.3 Combination of Passive/Active Probing

Both the passive approach and the active approach have some advantages and disadvantages. In an attempt to benefit from both techniques, a combination of passive/active is suggested in [11] and a model called *Change-of-Measure Based Passive/Active Monitoring* (CoMPACT Monitor) is explained. The system has a set of active probes and a NOC. The monitoring operations are controlled by the NOC, initiating measurements task done between active probing agents. The results in form of jitter, delay and loss

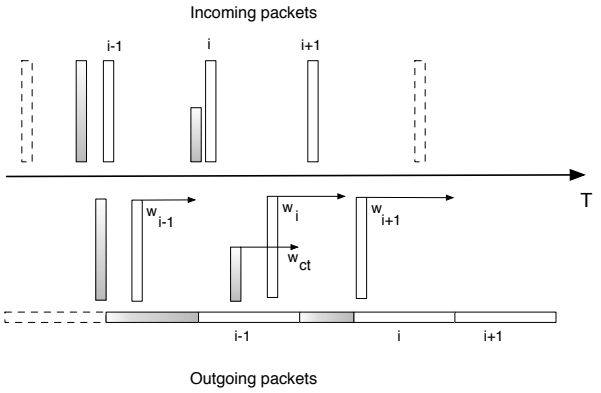


Figure 4.8: Quantification pattern.

are then sent to the NOC which keep track of the service quality in the network. Although this model is tested in a VoIP environment, the concept can be used for other real time media such as IPTV. The probes use the *Network Time Protocol* (NTP) to synchronize their clocks, a timestamp is then put in the RTP payload and sent. The average one-way delay includes a packetization delay, which is the delay of filling a payload with encoded/compressed data. This delay depends on the codec being used. The one-way delay also consists of a jitter buffer delay, which depends on the size of the jitter buffer.

When a packet is received, the timestamp of the sender is subtracted from the current time at the receiver to get the propagation delay. The packetization delay value for different codecs is predefined and is added in order to get the total delay. If the jitter buffer size is greater then zero, this will also be added to the total delay. Yet another delay is added at the end, the so called playback delay, D_{pb}, how this delay is handled depends on the media and is left out in this report. For the passive part of the system, a number of passive probes are set up on the links and dissect the protocol field in packets, if it turns out to belong to a protocol of interest, statistics of the session are collected and stored (e.g recorded). This information is then sent to the NOC which correlates the measurements to the quality at that point in the network. Each record contains three fields: *protocol*, *key* and *statistics*. The key corresponds to a unique identity for a record of a specific connection and the measurements from the probes are stored in the statistics field. For each packet received, a check is done to establish if a record already exists for that key, then an update is made. For *Session Initiation Protocol* (SIP), the type of header and local time of arrival is

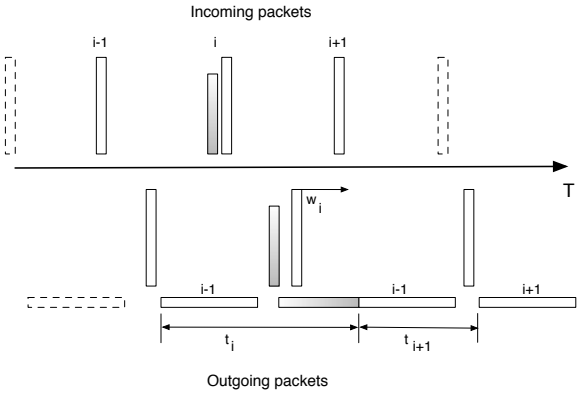


Figure 4.9: Mirror pattern.

stored. For RTP, the number of packets observed are incremented , computing average timestamp, updating the minimum and maximum observed sequence numbers, and update the reception time of the last packet. The passive/active approach may have some benefits. A combination of both can, if combined correct, lower cost, time and network space. An interesting solution can be to have passive probes at key points in the network, tentatively at the border between the core network and the access network. These probes can be set to detect anomalies within traffic of interest and when a problem occurs, it can trigger alarms to the operators, who can take actions accordingly.

4.2 Traffic Classification

The part of traffic classification in the passive branch of probing is of great importance for an *Internet Service Provider* (ISP) and is used for network planning, security and QoS. In an attempt to provide QoS for OTT-traffic, one step is to be able to identify the traffic of interest in the network. This is not as easy as it sounds due to encrypted traffic, dynamic port selecting applications, and on top of that both content providers and internet service providers may have to respect the privacy of their customers. A lot of research has been made in the area of network traffic classification and a majority of the methods are based on:

- Transport port numbers, this approach however lacks efficiency due to dynamic port selection and port-tunneling.

- Signature-based, which fails when it comes to encrypted payloads.
- Heuristics and/or behavioral based, which is not especially efficient for real-time or online classification.

There are several different techniques today and many of them are based on older versions, two major directions can be distinguished as *Content Based* and *Statistical Methods* as seen in Figure 4.10. The Content based part of the classification-tree is a *Deep-Packet-Inspection* (DPI) approach, and uses information such as IP/TCP headers, port numbers and payload information. The right part of the tree however, sees classification as a statistical problem and decisions are based on statistical features of packet flows, such as number of packets, inter arrival time and packet size. The trend in classification techniques are changing, along with applications and time, and can be seen in Figure 4.12

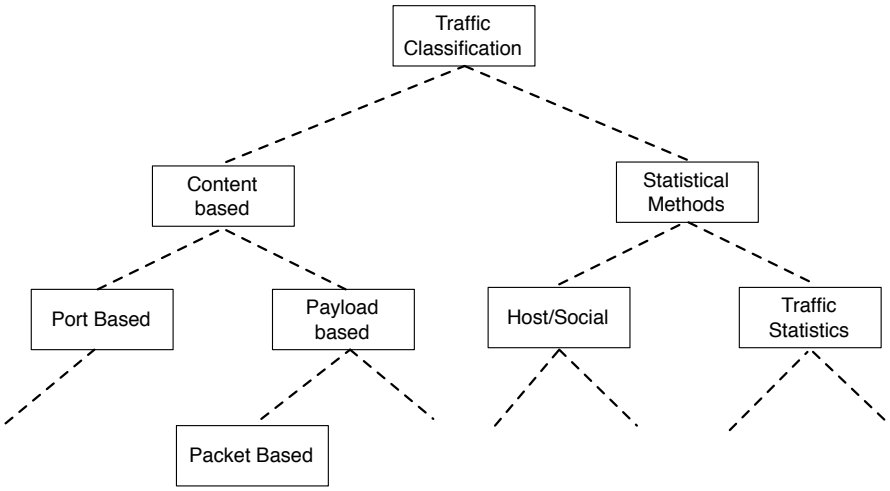


Figure 4.10: Conceptual picture of Classification Methods

4.2.1 Content Based Methods

Port-based Classification

Port-based classification is the most straight forward method of classifying network traffic. The early network applications such as FTP, SSH and HTTP are designated a pre-defined port number. These applications use

TCP or UDP, both of these protocols provide the port number of the connection in the header making classification rather straightforward. Port-based classification can still be used for classification of applications using static port numbers, but fails when it comes to applications using dynamic ports, such as passive FTP, Skype and P2P according to [13]. This is a very fast and very accurate technique, but an increasing number of applications today do not use static ports.

Payload Classification

One way of getting around the problem of dynamic ports is to inspect the payload, so called payload classification or *Deep Packet Inspection* (DPI). DPI is the process of any network equipment using non-header information, typically the actual payload. For each application, a signature has to be identified in that particular applications traffic. It is important to keep these signatures as simple as possible to allow operation in high bandwidth links. This approach is very accurate once signatures are determined, the down side is that signatures have to be updated when an application is updated or when a new application is developed and uses the same signature as an existing one. Other problems to deal with are storage capacity, computationally power and privacy.

Deep Packet Inspection (DPI)

Most DPI methods use signature analysis when trying to understand and verify applications. Signatures are unique patterns associated with every application. This can be a particular bit pattern or a unique field in the application protocol. Each application of interest must be pre studied, the collected signatures are then stored in a reference database which the classification engine can compare to later on. There are different signature analysis methods and the most common ones are:

- **Pattern Analysis** - Many applications embed patterns like bytes, characters and strings in the payload, which can be used by the classification engine to identify a protocol. Not all protocols have this type of pattern, then this method will not work at all.
- **Numerical Analysis** - This method looks at the numerical characteristics of packets, such as payload size, number of response packets and offsets. Older versions of Skype are good examples of where this approach works well. In this case the client request is 18 bytes long and the response is usually 11 bytes. The analysis may spread over multiple packets and therefore the decision can take some time.

- **Behavior and Heuristic Analysis** - Analyzing the behavior of the traffic can produce greater insight into the applications running. This behavior is then used to classify such applications. The statistical, or heuristic, analysis of the inspected packets, makes the underlying protocol possibly to classify.
- **Protocol/State Analysis** - For some applications, the protocol used follows a certain sequence of actions, such as FTP GET request followed by a valid server response. Such protocol conformance can be used to classify such applications.

Packet-based Classification

To solve the problem of encrypted payload, other methods have to be used for defining signatures. Packet-based classification uses packets of single uni- or bi-directional flows and are then used in the decision of what application the flow belongs to. It is possible to inspect each and every packet in a flow, but due to the enormous load on the classification engine in a high-speed link, some methods use only the first packets in a flow. Payload packet-based algorithms can use different features to define signatures and are as follows:

- Header fields
- Packet size
- Inter-packet gaps
- Numbers of transmitted packets
- Sequence of packets between hosts

The packet-based classification uses a subset of the previously mentioned features. Figure 4.11 shows which header fields are most likely to be used. Header fields from both IP and TCP are used to extract information, such as the IP-address of sender/receiver, port number of sender/receiver and finally which protocol used. More recent algorithms also use the flags of the TCP header, but besides that the remaining fields are rarely used in traffic classification. Packet size and inter-packet gaps can also be used, but the latter strongly depends on the delay between sender/receiver and effective algorithms are rare. The number of packets transmitted are mainly used for short flows with rather small numbers of packets, the last signature with a sequence of packets. Information from the first packet from receiver to sender is used and then information from the second and third one sent by sender to receiver and so on.

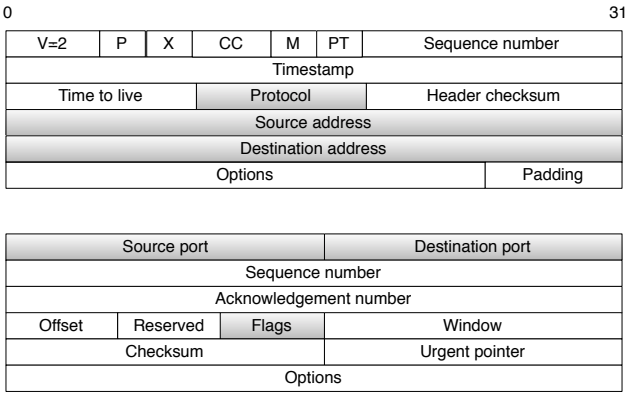


Figure 4.11: IP/TCP Header field used in traffic classification

4.2.2 Statistical Methods

This approach treats the problem as a statistical problem and is based on the fact that traffic at the network layer has statistical properties (such as the distribution of flow duration, flow idle time, packet inter-arrival time and packet lengths). These properties are unique for certain classes of applications, enabling different applications to be distinguished from each other. One area in focus regarding the statistical approach is machine learning, this is suitable when dealing with a large number of rules and/or when an adaptable program is preferred and will be discussed below. The basic concept of supervised learning is shown in Figure 4.13 while unsupervised is shown in Figure 4.14.

Machine Learning Classification

Machine Learning is a branch of artificial intelligence focusing on development of algorithms allowing computers to evolve their behavior based on empirical data. The main idea is to automatically learn to recognize more or less complex patterns and make decisions based on already known data. A problem lies in that the set of all possible behaviors given all possible inputs is of such magnitude, it is impossible to cover by the set of observed data (e.g. training data). Hence the learner must generalize from the examples given and make a useful output in new cases. Machine learning generally consists of two steps, model building and classification. A model is first built using training data and then the model is used with a classifier. Each traffic flow is then classified using features such as packet arrival interval, packet size, flow size, and flow duration. The technique can be divided into

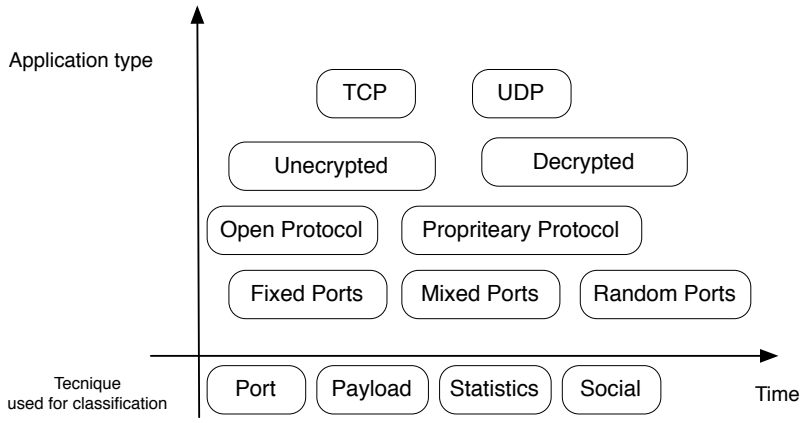


Figure 4.12: Trends in Classification

supervised and unsupervised learning. Supervised learning involves a machine learning from a set of pre-classified examples from which it builds a set of classification rules used to classify unseen examples. There exists a number of supervised learning classification algorithms, each differing mainly in the way the classification model is constructed and what optimization algorithm is used to search for a good model. The unsupervised learning does not use pre-classified data, instead it discovers natural clusters in the data using internalized heuristics. It clusters instances with similar properties, defined by a specific distance measuring approach such as Euclidian space, into groups. Regardless of what approach is used, machine-learning and statistical methods have the benefit of handling encrypted traffic and the lack of need for observing payload and heavy computational signatures.

Supervised learning requires a training phase, to link classes and applications, and the training phase requires a-priori classification, for this reason supervised learning may be attractive for the identification of a special application (or group of applications) of interest. Supervised learning works best when trained on examples of all classes expected to encounter in practice, otherwise performance can become degraded. The benefit of unsupervised learning is the lack of pre-labeled data. However, resulting clusters still need to be labeled by human expert.

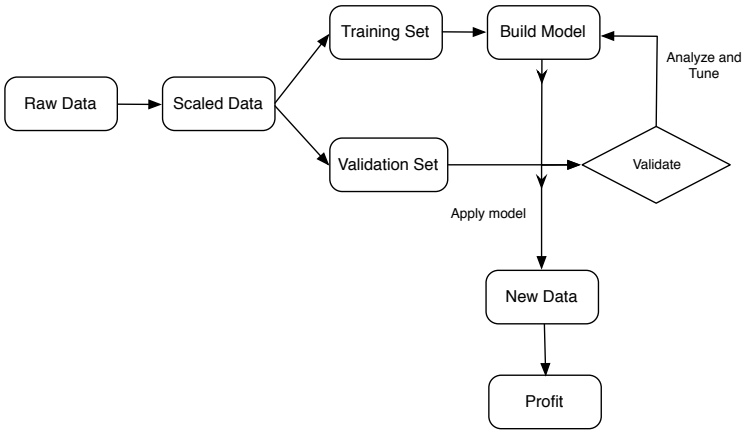


Figure 4.13: Workflow for Supervised Learning

Machine learning algorithms

- **Naive Bayes (NB)** - Bayesian classification is based on probability distribution of random events. NB determines the *a posteriori* probability for the event by the *a priori* probability in the dataset, assuming all features are equally valuable and independent. One advantage with the NB method over other machine learning methods is the simplicity to deal with complex situations, the assumption that all features are independent however, is not that realistic making NB to likely have less accuracy.

- **Bayesian Network** - Bayesian Network does not have the strict assumption of independence as NB, this method assumes conditional independency on the subset rather then the whole feature set. A Bayesian Network is a *Directed Acyclic Graph* (DAG) that encodes a joint probability distribution over a set of discrete random feature variables.

- **Naive Bayes Tree (NBTree)** - Is a hybrid of Naive Bayes and decision tree classifier and is best described as a decision tree of nodes and branches with Bayes classifier on the leaf-nodes.

- **C4.5 Decision Tree (C4.5)** - Is a greedy divide and conquer algorithm used for building of decision trees. C4.5 builds the tree from a set of training data using the concept of information entropy. The training data is a set of already classified samples. At each node of the tree, one attribute is chosen of the data in a way that it most

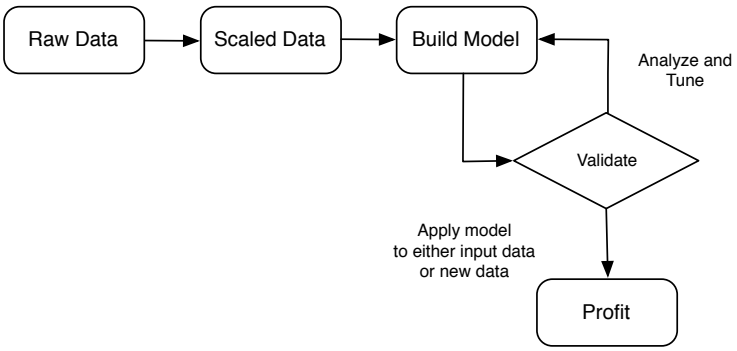


Figure 4.14: Workflow for Unsupervised Learning

effectively splits the sample set into subsets enriched in one class or the other. Its criterion is the normalized information gain that results from choosing an attribute for splitting the data. The attribute with the highest normalized information gain is then chosen to make the decision. The C4.5 algorithm then recurs on the smaller sublists.

- **Random Forests (RF)** - Random Forest is a classifier consisting of a collection of tree-structured classifiers and is a combination of tree predictors, in a way that each tree depends on the value a random vector. The algorithm has shown to have desirable properties such as convergence of generalization errors.

4.3 Classification Methods for Real Time Traffic

As seen above, a variety of methods exist in the field of classification and the classification tree can be expanded. The content-based approach is rather straight forward with good performance and accuracy. On the downside of this approach is the problem with dynamic ports, decrypted data and privacy concerns. The other approach - Statistics - is a method on the rise and can most often go around the problem mentioned above but often demands more initial work. Multimedia traffic like VoIP and on-demand-video often uses RTP and being able to classify this type of protocol is essential for classifying OTT traffic. However, some applications use their own protocol but still fall under the real-time media category. Skype is an example of delay sensitive application running over-the-top, not using RTP. Applications like that must also be detected in some way and different methods for detecting/classifying RTP, VoIP/Skype and will be explained

in the next section. Every application has their own special signature and it is almost impossible to have one solution for them all, however, if it is enough to classify the traffic as a whole, such as *video* or *audio*, some common features can be shown to exist.

4.3.1 RTP Traffic

RTP is, as mentioned before, an IP-based protocol used for transporting real time media such as video and audio. RTP is mainly designed for multicast real time data but is also used as unicast transport of video-on-demand and interactive services such as internet telephony.

Classification Methods of RTP Traffic

The authors of [18] suggests four methods of detecting RTP flows over an IP-network. In this thesis only three are explained since the the method developed by the authors did not show greater performance. The implementation of the classifiers is not explained, but some attributes that can be used to classify RTP traffic is described below. A detailed view of the RTP header can be seen in Figure 4.15.

1. **Method one** - The first classification method uses the following attributes for classifying RTP streams:

   - *Packet Length*: The length of the UDP packet must be more than 20 bytes, the size of the UDP header is 8 bytes and the size of the RTP header is at least 12 bytes. Consequently, it is impossible to be an RTP packet if the size of the UDP packet is not more then 20 bytes.
   - *Version number*: First two bits are the version number of RTP, current version is number two, hence, if the first two bits are not equal to two it is not a RTP packet.
   - *Payload Type*: Default payload types are defined by RFC 1890. So if the lower seven bits of the UDP payload second byte is not a valid RTP payload type, it cannot be a RTP packet.
   - *Sequence Number*: The third and fourth bytes of the RTP header are the sequence number. The initial sequence number is random so the validation of RTP sequence numbers of the current packet depends on the value of the previous one.

   The criteria for classifying a stream as a RTP stream, is at least n consecutive qualified packets from that stream.

2. **Method two** - The second classification method uses the same four attributes as method one and adds three more attributes:

   - *RTP P bit*: The **P** bit is the third bit of the RTP header, if this bit is set, the RTP packet contains one or more padding bytes at the end. The last byte of the padding contains the number of padding bytes and is not a part of the RTP payload and shall be ignored.
   - *RTP X bit*: The **X** bit is the fourth bit in the RTP header, if it is set, the fixed RTP header is extended by one header extension.
   - *RTP CC*: **CC** or CSRC Count is the lower four bits of the first byte of the RTP header and indicates the number of CSRC identifiers that follow the fixed header.

   In method one, we had a rather loose bound for packet length checking. In method two with the help of bit **P**, bit **X** and **CC**, a more precise bound is suggested.

3. **Method three** - The last method uses both RTP and RTCP packets at the same time. Every pair of streams shares a common flow ID and the sum of identified packets consists of both identified RTP packets and identified RTCP packets, compared to methods only using RTP packets. This method accelerates the process of RTP stream classification. To classify RTP packets, the same four attributes are used as in method one, to identify RTCP (as seen in Figure 4.16) packets are inspected by the following three attributes:

   - *UDP packet Length*: For the RTCP packet, the length of the UDP packet must be greater than 12 bytes, which is the sum of UDP header and RTCP header.
   - *RTCP Version Number*: The first two bits of a RTCP packet is the version number and the value of these must be two.
   - *RTCP Type*: The second byte of the RTCP header is the type of RTCP and must be one of the valid numbers defined for RTCP.

   The criteria for classifying a RTP/RTCP stream pair is when at least n qualified packets from that stream pair are captured, and m of the n packets are qualified RTP packets. The feature of qualified RTP packets is the same as in method one, for RTCP it is UDP packets with packet length greater then twelve, version number equal to two, and packet type is one of the valid pre-defined numbers.

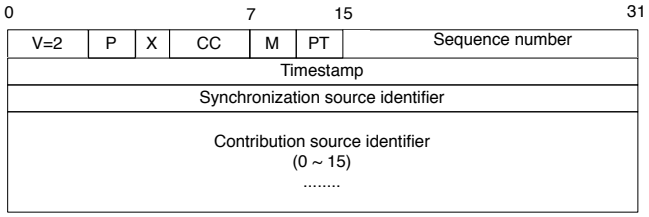


Figure 4.15: RTP Header

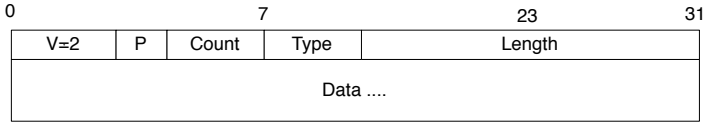


Figure 4.16: RTCP Header

4.3.2 Audio Traffic

VoIP is the family name for applications sending and receiving voice over an IP-network, two common applications are *Google Talk*, and *Skype*. However, these applications use different architectural and protocol layouts making analysis and comparison necessary to understand their behavior in an attempt to define the QoS for these applications. Other applications that uses audio but is not in the VoIP family are applications that stream music, *Spotify* is one example of such a service and is also of interest when in this research.

- **Skype** - One major success factor for Skype is its ability to operate behind firewalls and NATs. It also provides security in voice communication by encrypting the data and uses randomly selected ports in an attempt of disguising itself. The network of Skype nodes consist of so called *ordinary nodes* and *super nodes* and form a P2P network. The ordinary node is a client which is used to place a call, an ordi-

nary node becomes a super node if it has a public IP address and enough bandwidth. The ordinary node must connect to a super node and the Skype login-server to be part of the Skype network, the server performs user authentication and maintains password and certificates for the members. Online and offline user information is stored and propagated in a decentralized manner, this also includes user search queries. To search after users, Skype implements a third generation P2P technology called *Global Index Technique*. This is a multi tier network where each super node can obtain knowledge of all available users and resources with minimum latency. To solve the problem of working behind NATs and firewalls Skype also uses a variant STUN protocol to determine the NAT and firewall type. Usually in NAT scenario, Skype uses a relay node which communicates between two end systems. Each node of Skype contains a host cache, which is a list of reachable nodes and contains IP addresses and port numbers of super nodes, during a Skype operation the client uses this list to find a super node to connect to. Skype uses a non public protocol, where both signaling and media traffic are encrypted for any kind of Skype connection, making both port based and payload based classification of Skype traffic hard, if at all possible. Instead more statistical approaches have been suggested, by gathering information such as packet arrival rate and packet length.

- **GTalk** - GTalk is based on an open standard, XMPP-Jabber/Jingle, which enables a client of GTalk to communicate with other XMPP enabled VoIP applications. The XMPP standard GTalk is based on is an XML streaming technology and is a robust way of transporting data, in real time, between users and applications. The VoIP extension of XMPP is known as Jingle and its purpose is to provide peer-to-peer media sessions between XMPP entities. The negation is made over the channel of XMPP but the media is sent outside this channel using techniques such as RTP. GTalk solves the NAT problem by using Jingle ICE, a variation of STUN protocol. GTalk was released in 2006 and not much research has been made in the field of classification of GTalk traffic. The traffic between the GTalk client and the GTalk server is encrypted but end-to-end encryption is however not yet supported.Future releases will most likely support end-to-end encryption as standard. GTalk traffic is easier to detect since the application uses an open standard protocol and fixed ports.

- **Spotify** - Spotify is a streaming music service using peer-to-peer

technology. One distinguishing feature is the low playback latency (median 265 ms). Spotify uses a proprietary client and protocol and works on most platforms as well as on smart phones. The latter is not part of the peer-to-peer network and only streams music from servers. The audio is encoded with *Ogg* and the bit-rate is either, 96kbps, 160 kbps or 320 kbps, depending on if it's a free or premium user. In the Spotify network, there are no *super-nodes* as in the Skype network, a client will only connect to new peer when it wishes to download a track it believes that peer has. TCP is the used transport protocol and the explicit connection signaling helps stateful firewalls. Spotify lacks the ability of NAT traversal, this is however mitigated by two factors. First, when a client wishes to connect to another peer, a request is sent through the server asking the peer to attempt a TCP connection back to the connector, secondly, clients use the *Universal Plug and Play* (UPnP) protocol to ask home routers for a port to use for incoming connections. For a more detailed description of Spotify, see [22]. Spotify is, as mentioned, a P2P application able to use different port numbers making classification difficult, a suggestion is that more advanced DPI techniques or some host/behavioral techniques can be used.

Classification Methods of VoIP Traffic

Authors of [16] explain a solution for detecting encrypted VoIP traffic in IPSec tunnels and also try to increase the QoS parameters. VoIP traffic packet length usually is in the range of 60 to 150 bytes, and not a lot of other network traffic is in this range of size. A very simple algorithm is proposed, if data packets are within this range of size, they will be passed right through the proxy server used in the testbed, otherwise they will be delayed to cause non VoIP traffic to slow down. When a packet arrives to the first network interface of a proxy device it runs through a so called pre-routing phase, here it is decided whether the packet is going to be modified or any NAT decisions to be made. The packet is then sent to queue and there the VoIP identifier is invoked, the identifier looks at the packet and its own record history of packets to decide if it is most likely a VoIP packet or not. A VoIP packet is passed right through, and all none VoIP packets are queued 100 *ms*, giving priority to VoIP-traffic. Three scenarios are tested, the first one uses two host machines connected trough IPSec, the second one uses the same scenario but adds 2.5 megabits of random cross traffic and finally the third scenario the detection of VoIP/no-VoIP traffic is turned on. Three different codecs were used in the test, G711, G723 and G729. The authors use three sub-fields of Machine Learning algorithms,

Supervised Learning methods, Unsupervised Learning methods and Hybrid Learning approaches. Their conclusion show that it is possible to detect decrypted VoIP traffic with good results with help of Machine Learning approaches. Their results shows increasing performance for network layer tunneled VoIP traffic, but also state that more research has to be done in the area of network layer encrypted tunnels.

In [19], it can be read that in spite of the fact that Skype conceals its application-layer protocol, it is still possible to monitor the network and transport layer and analyze used IP addresses and ports. The method is split into two parts, the first one is detecting Skype activity and the second part is detection of Skype voice calls. The first possible way of identifying Skype hosts is to look for Skype-specific connections such as connections to the *login server*, *buddy-list server* or *supernodes*. Occurrence of any of these infers the presence of Skype. Two other characteristic features of Skype is the connection to the *update server* and a TCP connection on port *33033*. The latter is the default port for super node connections. If a host is already logged in it will not be possible to detect the above actions, instead the permanent connection to the super node can be supervised. The client communicates not only with super nodes and servers but also maintains direct relations with several other Skype clients. The UDP relation has well defined characteristics when it conducts voice calls or is in idle state, the two states can be separated by the size of packets and can be detected by a simple detection method in three steps:

- Select UDP flows of more then 10 packets and the source/destination port does not belong to a well known application.
- For the remaining flows with packets smaller than 60 bytes, calculate the main mode of inter arrival-time. The inter-arrival time is generally 20 seconds but to avoid errors from deviation in the arrival-time the main mode (center of) the histogram is calculated.
- The flow is likely an UDP relation if the main mode equals 20 seconds.

The first rule is applied in order to get rid of the flows that can be unambiguously identified as not being a signaling flow and reduces the time needed to verify rule two. All flows are, according to rule one, discarded if they do not contain enough packets to be a UDP relation or have a source/destination port of a well known application. The identified UDP relations then result in a list of IP-addresses and port number pairs that can be used for identification. This is a heuristic approach based on flow-dynamics and

characteristics on packet-level and flow-level to identify Skype traffic. One benefit is that there is no need for payload information. The method expects pre-captured data as input but can be built into an online identification tool according to the authors.

4.3.3 Video Traffic

The following is stated by *International Telecommunication Union focus group on IPTV* (ITU-T FG IPTV). *"IPTV is defined as multimedia services such as television/video/audio/text/graphics/data delivered over IP based networks managed to provide the required level of quality of service and experience, security, interactivity and reliability."*
IPTV services can be divided into three major groups:

- Live TV - Sending live and can be with or without interactivity.
- Time-Shifted TV - Replays a video sequence broadcasted in the past.
- Video-on-Demand (VOD) - Watched video on end user demand.

Depending on the network architecture of the service providers the server architecture can be of mainly two kinds, centralized or decentralized. The centralized solution does not require a comprehensive content distribution system, the decentralized or distributed system has however bandwidth advantages but requires an intelligent and sophisticated content distribution.

- **YouTube** - YouTube is a video sharing website making it possible for users to upload, share and view videos. In 2010 an experimental version of the site was launched and is based on *HTML5* and the need of *Adobe Flash Player* is then not required. You Tube originally offered only one quality level (320x240) using the *Sorenson Spark* codec which is a variant of *H.263* and audio in mono *MP3*. Today You Tube supports *3GP*, *720p HD* and *1080p HD*. One key feature of You Tube is the ability for users to view the content on webpages outside the site, each video is accompanied by a piece of HTML that can be used to embed the content on an outside webpage. You Tube Mobile was launched in 2007 and uses RTSP for the streaming video, not all videos are accessibly this way.
  Classifying HTTP video applications by port number can be difficult since many applications use port 80. DPI can be used, but one must have privacy of the users in mind. Some machine learning approaches have been suggested, using features such as flow duration or average packet size.

- **PPLive** - is a peer-to-peer streaming video mesh-network and is a part of a new generation P2P applications that combines P2P and Internet TV, so called *P2PTV*. The main component of PPLive is the TV engine which downloads video blocks from the PPLive network and then streams it to the media player. The data is streamed through two buffers, one in the engine and one in the media player. This is done to reduce the impact of network jitter and keeping a more efficient distribution of data between peers. A client downloads the media content from several peers and uploads the data in the cache at the same time. The video bit rate ranges mainly from 250 kbps to 400 kbps, but some few channels have a bit rate at 800 kbps. PPLive is a proprietary system, neither source code nor protocol available. The platform of PPLive consists of multiple overlays, a single overlay corresponds to a PPLive channel and each peer in an overlay is identified by its IP-address and port number pair. For a more detailed description of PPLive see [23], the authors also test an active/passive approach of investigating the PPLive traffic.

Classification Methods of Video Traffic

In [20], a method of classifying the P2P-TV application is suggested, only relying on the count of packets exchanged with other peers during a small time window. The aim is to classify P2P-TV end-points identified by their port number and IP address and focus on UDP traffic since it is often preferred by P2P-TV applications. Also only the downlink is used for gathering information. Furthermore, these types of applications exchange the video stream data in chunks, since each application independently selects the size of the chunks. Differences in this choice will be reflected by the raw packet count. Now consider the traffic received by an end-point $P_x = (IP_x, Port_x)$ during a time interval ΔT. Under this interval peer P_x will be contacted by $K(x)$ other peers and receives a different number of packets from each and one of them. The number of peers that sent a number of packets in a time interval $I = [a, b]$ to peer P_x is then derived. $B{+}1$ intervals are used with exponential width $\{I_0...I_i...I_B\}$ such that $I_0 = (0, 1]$, $I_i = (2^{i-1}, 2^i]$ and $I_B = (2^B, \infty]$. This means that $N_x^i = N_x^I$ counts the number of peers sending to P_x in the interval $(2^{i-1}, 2^i)$, and $N_x^B = N_x^{I_B}$ counts all peers sending at least 2^B packet to P_x. A behavioral signature $\underline{n}^x = (n_0^x, ..., n_B^x) \in R^{B+1}$ is then built for each time interval ΔT by normalizing N_i^x over the total number of peers contacted P_x during that interval. The classification framework uses *Support Vector Machines* (SVM) which belong to the Learning Machines methods. Although the authors see their method as a *first step* towards fine-grained behavioral classification it shows some good results.

In [12], a packet-level traffic classification is suggested based on *Hidden Markov Model* (HMM). Specifically the Packet-Level Hidden Markov Model is proposed. The classification is done by using real network traffic and then estimated with the characteristics of packet size (PS) and inter packet time (IPT), making it well suited for encrypted traffic as well. In Figure 4.17, the general architecture is shown and consists of a bank of parallel PL-HMMs. In order to capture the characteristics of N different typologies of network traffic it is assumed that N different PL-HMMs are obtained via *Baum-Welch* training. The Baum-Welch algorithm is an iterative procedure looking for model parameters and maximizing the probability that the model itself generates the sequence used as the training set. Each of the PL-HMMs in the bank is then used to compute the probability that the test sequence belongs to the traffic typology associated to a particular PL-HMM. Finally the maximum likelihood selects the best estimate for the traffic typology. Specially peer-to-peer video streaming, also called *PPTV* or *PPLive* is tested and evaluated. PPTV/PPLive is a peer-to-peer streaming video network and is a part of P2P applications that combine P2P and Internet TV. The conclusion of PL-HMM shows it is a promising technique and can be well suited for encrypted traffic and used in a multi-classifier system in the attempt of trying to classify OTT-traffic.

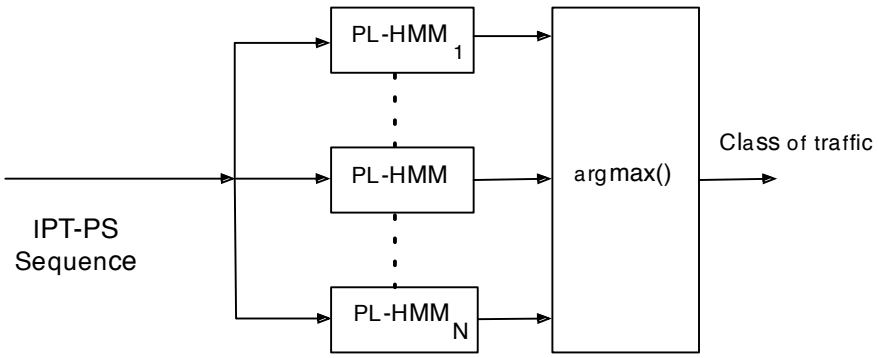


Figure 4.17: Architecture of PL-HMM classifier

In [17] a statistical approach is taken based on the intuition that video

and voice streams have strong regularities when it comes to packet inter-arrival time and packet size. The authors propose a system called *VOVClassifier* that not only identifies voice and video traffic, but also labels the flows with corresponding application. The VOVClassifier works in two steps, offline training phase and online detection phase. In the offline mode a sample set of flows from applications of interest is fed to the classifier and extracts correlations between packet size and inter arrival time. The result is a fingerprint that can be used in the classification. The classifier is an automated learning system and uses packet headers from raw packets from the link. Three major modules operates in cascade and are:

- **Flow Summary Generator (FSG)** - All packets are processed by the FSG module and organize packets according to their 5-tuple (e.g source/destination IP address, source/destination port number and protocol type). The processed flow is then characterized in terms of packet sizes and and inter-arrival times.
- **Feature Extractor (FE)** and **Voice/Video Subspace Generator (VSG)** - The output of the FSG is forwarded to the FE which computes a feature vector for every flow by analyzing the power spectral density, this is done in order to exploit regularities residing in voice and video traffic. The VSG then processes the feature vectors by partitioning the vector space into non overlapping clusters and extracts the characteristic of each cluster.
- **Voice/Video Classifier (CL)** - In the detection phase data are processed by the CL which calculate the distance from the feature vector from the current flow to the subspace generated during the training phase in order to classify it as either voice or video.

As a summary of this approach the authors conclude that by using a stochastic process, that combines packet size and inter arrival time, that extracts regularities and highlights their major differences, the VOVClassifier could achieve very good results.

4.4 QoS/QoE Correlation

The last major step in the process of PAXSa is mapping. When all probing and classification is done the gathered data from QoS parameters must be mapped, if the desire is to estimate QoE for end customer. Mapping is mostly a pre-step, once a suitable map is configured it is more a question of simply looking up what QoE value corresponding to QoS value on hand. This part may appear to be easy, but QoE is a very subjective assessment

and can be hard to determine, some research has been done to categorize different levels of expected QoE for end-customers such as in the work of [2]. It is shown that different types of content are affected differently by network performance. Three maps are created in an attempt to map network performance (e.g. packet loss, jitter, delay) to a prediction of how the media-traffic is effected, and in the end effecting QoE for end-users. This means that information regarding the type of media that shall be observed must be well known and predetermined, there are also levels within each media type. Rapidly changing video content, such as an action movie, is more likely to suffer from bad QoS then slow video such as news and weather.

According to [3], end-users are more likely to notice performance degradation when the connection is in one of following explained states. The *Route Change State* is usually caused by failures in routers/links or when a failed component recovers from an outage, the author distinguishes between layer 2 and layer 3 route changes and without going in on details, layer 3 changes can be detected at the end node from IP time to live (TTL) and trace-route changes. Layer 2 changes are more difficult to detect but a method of observing patterns in the *Round-Trip-Time* is suggested. Route changes typically cause long burst of lost packets and most certainly will have impact on both QoS and QoE. The second state is *Burst Loss State* and occurs when a large number of consecutive packet are lost over several seconds, coding and interleaving techniques usually solve these issues but not in this state. The network is said to be in *High Random Loss State* when loss probability is greater than some threshold value, above this threshold packet losses introduce audio/video-impairments that cause QoS parameters to decrease to an unacceptable level. In many cases high random losses can be the worst, most often a large receiver buffer can handle jitter but this also make the total sum of delay larger. The last state is maybe not so much a state but more lack of a state, it can be hard to easily fix and will cause the worst QoE for end-customers, the *Disconnected State* is when all transmitted consecutive packets are lost for a long period of time and are due to fiber cut in core network, loss of power or other hardware/software failures.

The work of gathering data about how QoS parameters effects the QoE for end-users can be both time-consuming and expensive and some techniques used are human interaction (e.g. people watching video). *Mean Opinion Score* (MOS) is a widely used metric and gives a numerical indication of the perceived quality of the media received after being transmitted and eventually compressed using codecs. The MOS value is the arithmetic mean of all individual scores set by the test panel and ranges from 0 to 5,

with 5 being the best possible and 0 the worst. It is also shown that if the mouth-to-ear delay is greater then 400 *ms* most end-users will be dissatisfied. For multiplayer games online an end-to-end delay of 200 *ms* will be noticeable and annoying for the end-users, each and every multimedia type has its own limit regarding delays. This makes it a huge task to set up a well suitable map. One parameter often missed, according to the author of this thesis is the parameter of *expectation*, it is of my opinion that QoE is connected to expectation. If allowed to generalize, people expect more from SVT Play than YouTube, these expectations of course change over time and most certainly differs between gender and age. Many other factors such as size of buffer on receiver side and codec used affects the correlation between QoS and QoE. In this field, much more research has to be done in an attempt of making a correct mapping.

Although many problems occur on the upper levels of the OSI model such as, transport layer and application layer, it is of interest to find out how layer one (Physical layer) affect the QoE for end customers. The physical layer is responsibly for the ultimate transmission of data over a network in form of electric voltage, radio frequencies and optics. It is also here disturbances from other electro magnetic equipment have their impact. At the application layer one can observe the Peak Signal-to-noise Ratio (PSNR), a well known metric to measure video quality in an objective way. At the transport layer observations regarding packet loss can reveal information about the quality of for example video traffic. At the physical layer metrics such as number of damaged blocks received by end-users modem, line bit rate and actual SNR are of interest.

Chapter 5
Controlled Disturbance

In order to evaluate probing methods and correlation between QoS and QoE as described in previous chapters there is a need of controlling a DSL link in a noisy environment. The solution is a controlled disturbance signal, seen in Figure 5.1, where both center frequency and bandwidth can be varied. Such a device and its deployment is describe in this chapter and is executed over a laboratory DSL network in the department of electrical and information technology in Lund. As mentioned previously, DSL technique use Discrete Multi Tone and this is performed by the DSLAM on the network operator side and by the *Customer Premises Equipment* (CPE) on customer side. DMT is a method of converting data into so called tones where each tone forms a sub-channel of 4.3125 kHz. The amount of bits that can be carried by these tones depends upon the SNR at that particular frequency. If the SNR is good a maximum of 15 bits can be allocated to that tone. A disturbance signal can be generated using *Pulse Amplitude Modulation*, or PAM for short. The reason for this choice of modulation is that PAM has similar frequency spectrum as signals from radio stations that is known to interfere with DSL links. A short introduction to PAM and signal theory is given in next section.

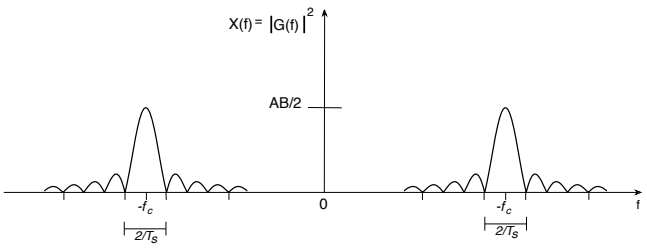


Figure 5.1: Frequency content in $x(t)$

5.1 Signal Theory

A very common PAM signal is the square wave, here denoted $g(t)$, and there are two types of techniques for this. The first one is called *unipolar square wave*, sometimes called a rectangle wave and it transmits a logical 1 using a square pulse of amplitude $+A$ voltage and a logical 0 is transmitted with 0 voltage. The second one is the *bipolar square wave* and transmit a logical 1 as a square wave with amplitude $+A$ voltage and a logical 0 as a square wave with amplitude $-A$ voltage. In a time period $0 \leq t \leq T_s$, the l:th signal of a general PAM signal can be expressed as:

$$s_l(t) = A_l g(t) \tag{5.1}$$

In order to control a signal there are two important parameters, the first is the frequency f and the second is the bandwidth W. Via two equations these parameters makes it is possible to control the width of the main-lobe in the signal.

$$f = \frac{1}{T_s} \tag{5.2}$$

$$W = \frac{2}{T_s} \tag{5.3}$$

In signaling systems it is important to investigate the characteristics of the signal in both the time domain and the frequency domain. By Fourier transform it is possible to alternate between these domains for any given signal, the Fourier transform decomposes a function into the sum of a (potentially infinite) number of sine wave frequency components. A general unipolar square wave signal in the frequency domain can be seen in Figure 5.2. The Fourier transform is given by:

$$G(f) = \int_{-\infty}^{\infty} f(t) e^{-j2\pi ft} dt \tag{5.4}$$

A better way to describe the PAM signal is the sum of the signal over a given period of time. Often a signal is sent not only during one signal time but rather over an extended period of time. A PAM signal can then be seen as:

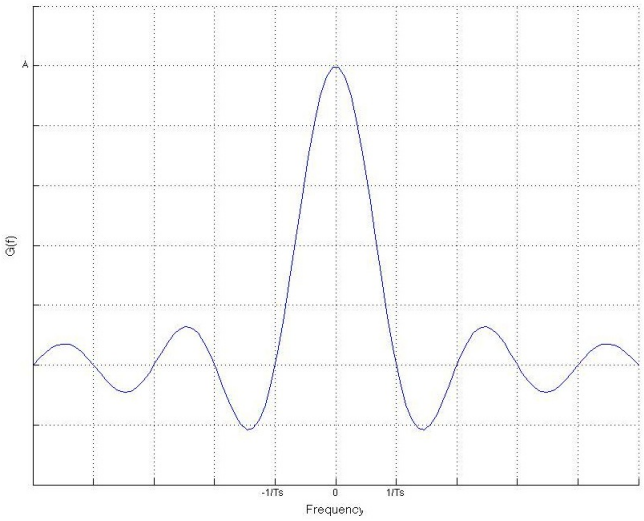


Figure 5.2: Rectangular pulse G(f) in frequency domain

$$s(t) = \sum_{k=0}^{\infty} A_k g(t - kT_s) \tag{5.5}$$

where A_k is the amplitude of the generated signal. If a bipolar square wave is used it will alternate between $\pm A$ voltage with some given probability. The power spectral density function $R(f)$ of the signal $s(t)$, here denoted $R_s(f)$, is of importance since it shows how the average signal power in $s(t)$ is distributed along the frequency axis. $R_s(f)$ can be calculated in two steps, first the autocorrelation function $r_s(\tau)$ is calculated as:

$$r_v(\tau) = E\{v(t+\tau)v(t)\} \tag{5.6}$$

Where $E\{...\}$ denotes the expected value. The second step $R_s(f)$ is obtained by the Fourier transform of $r_s(\tau)$:

$$R_s(f) = \int_{-\infty}^{\infty} r_s(\tau) e^{-j2\pi f\tau} d\tau \tag{5.7}$$

The power spectral density of a general signal $s(t)$ can be seen in Figure 5.3. To set the baseband signal $s(t)$ at the center of a carrier frequency

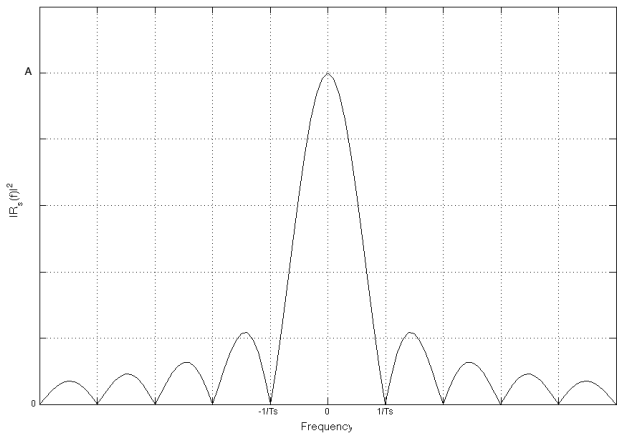


Figure 5.3: Power spectrum density of $s(t)$

f_c, convolution of $s(t)$ and some cosine function here denoted $B\cos(2\pi f_c t)$ is done. The use of a carrier frequency allow placement at a desirable frequency in the ADSL band making it possible to not only put the signal at arbitrary frequency but also influence the frequency range it will affect. A signal around a carrier frequency is called a pass-band signal and here denoted $x(t)$ and expressed as:

$$x(t) = \sum_{k=0}^{\infty} A_k g(t - kT_s) \cdot Bcos(2\pi f_c t)) \tag{5.8}$$

This operation is based on the Fourier transform relationship (5.10) and (5.11) and gives a characteristic lobe at $\pm f_c$ as illustrated in Figure 5.1.

5.2 Methodology

The signal theory in the previous section is applied and an external signal is generated by a software written in C# and the complete code can be seen in Appendix A. Besides the above mentioned software the set-up consist of a *UM232R - USB Serial UART Development Module*, a *Maxim 233EEP* device and a *Function Generator* as seen in Figure 5.4. A circuit diagram over the connections between UM232R and Maxim 233EEP can be seen in Appendix B. The base-band signal generated via the UM232R unit has a frequency range from 200 Hz to 3 MHz. Since the frequency f is connected to the bandwidth W as mentioned before this is a very general signal and can be used also in applications outside the scope of this thesis. However, the Maxim 232EEP unit only handles frequencies up to about 150 kHz and sets the upper limit. In this test the frequency used will be 100 kHz and is well within the limits of both devices. Note that the generated pass-band signal is *not* cyclic. The importance of having a random signal is related to the fact that any repeating non-sinusoidal waveform can be equated to a combination of discrete components. The resulting signal would not have the desirable characteristics in this case.

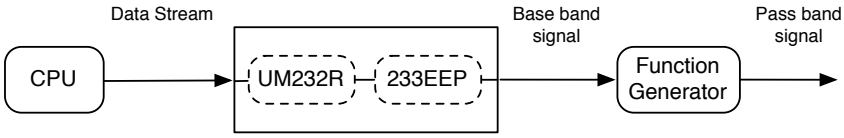


Figure 5.4: Generalized view over lab set-up.

The UM232 USB device is a development module connected via the USB interface to the computer running the written software and generates a baseband unipolar square signal. In order to generate the wanted output signal from the USB device the software controls the device to write random bytes at selected speed. This is handled by a GUI where execution time as well as frequency can be tuned. The frequency, or actually the baud-rate, can vary between 200 $Baud$ to 3 $MegaBaud$. Since we have binary signaling the baud-rate equals the bit-rate (disregarding over-head) so baud-rate and frequency more or less also becomes equal. The output of the UM232 generates +5 voltage when transmitting a 1 and +0 voltage when transmitting a 0. The signal is then transmitted to the Maxim 233EEP, a unit intended for all EIA/TIA-232E communications interfaces. This is done since the output of the UM232 does not produce a signal with

sufficiently large amplitude difference. The 233EEP unit however, produces an output with greater difference in amplitude. The amplitude of the now generated signal alternates between $\pm 5V$ with the same probability $p(A_k = +5) = p(A_k = -5) = 1/2$. Finally the base-band signal is placed at the center of a carrier frequency. By placing the base-band signal around a carrier frequency it can be set at a desirable location in the xDSL band and making it possible to not only put the signal at arbitrary frequency but also influence the frequency range it will affect and is done by the function generator. The generator used in the test adds a bias term at frequency 0 Hz, which means that there is a spike in center frequency used. This is however not a problem of great magnitude.

The now generated pass-band signal (disturbance signal) is used in the DSL network with 2.1 km of cable. Due to lack of space the cable is still on the original cable drum and not unwound. Before any test is done a reference measure is executed and bit-loading, quiet line noise (QLN) and SNR can be seen, on an undisturbed line, in Figure 5.5, 5.6 and 5.7 respectively. During six ten-minutes intervals, a simulated IPTV transmission is sent over the DSL network by the technique used in ADSL2+. The controlled disturbance signal is injected on a parallel pair in the same cable, in time intervals, during the first $500m$. All lines are ended with an end resistor with 100 ohm. The frequency of the generated signal is static at 100 kHz and via equation (5.1) and (5.2) it is shown that the bandwidth of the generated disturbance is approximately 200 kHz. The center frequency is however, alternated during the test and is displayed in Table 5.1.

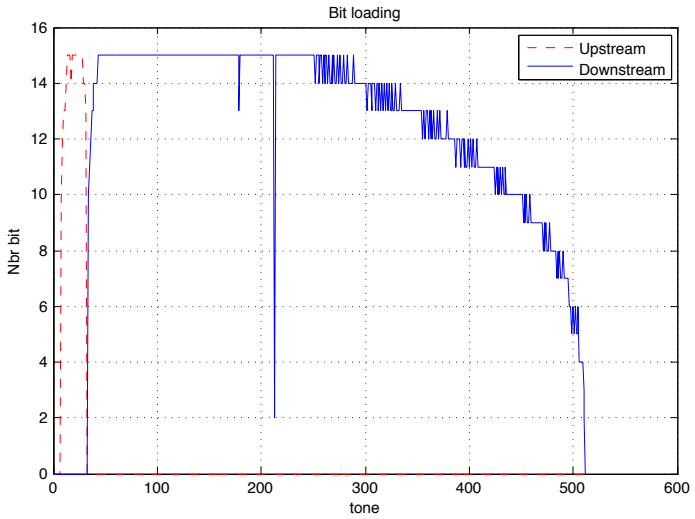


Figure 5.5: Bitloading without any disturbance.

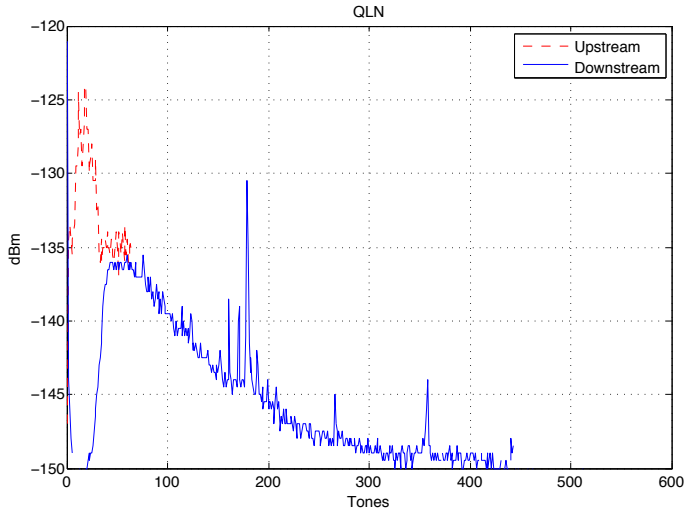


Figure 5.6: Quiet line noise without any disturbance.

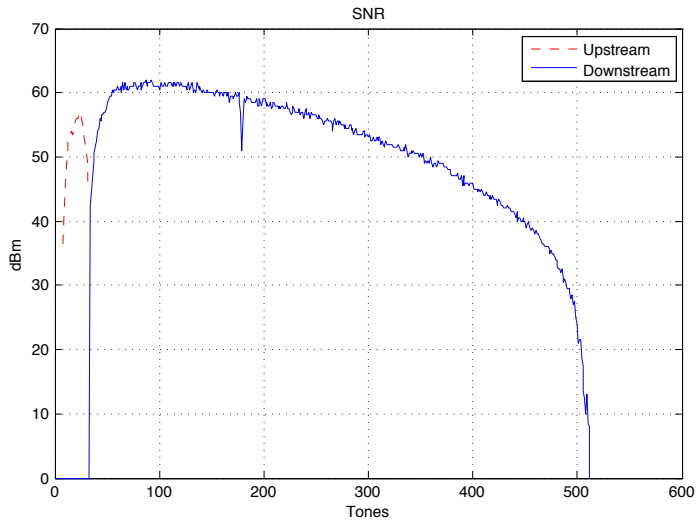


Figure 5.7: SNR without any disturbance.

Test interval	Time	Disturbance
1 :	$0 - 10min$	No Disturbance
2 :	$10 - 20min$	Disturbance at Center frequency 0.5 MHz
3 :	$20 - 30min$	No Disturbance
4 :	$30 - 40min$	Disturbance at center frequency 1 MHz
5 :	$40 - 50min$	Disturbance at center frequency 1.5 MHz
6 :	$50 - 60min$	Disturbance at varying center frequency

Table 5.1: Time intervals of controlled disturbance

5.3 Results

The results on the impact of the generated pass-band (disturbance) signal previously explained will be discussed here. In Figure 5.8, 5.9 and 5.10, it is clearly seen how the disturbance causes severe problems at the respective center frequency and that the higher frequencies in the DSL band suffer most. In Figure 5.11 the number of lost packets in the emulated IPTV stream is illustrated. The peaks have a clear correlation between the starting time of the disturbance signal. It is once again clearly seen how higher frequencies suffer most from external disturbances and have higher packet loss than lower frequencies, something that will affect the QoS. In time interval two and three, 10-20 min and 30-40min, the effect from bit-swapping is seen as narrow peaks. Bit-swapping takes some time to perform and in time interval 6, 50-60 min, when constantly changing the center frequency, the effect of this process is very clear. This effect is also seen in Figure 5.12, Inter packet gap (IGP) is more than 10 seconds in the worst case and will severely degrade the QoE for end-users.

Number of lost packets in the test can be seen in Figure 5.11. In Figure 5.5, bit-loading can be seen for the DLS link with no disturbance affecting it. The dotted line represent upstream band while the solid line represent the downstream band. The dip around tone 210 is a synchronization signal, also called pilot-tone, of two bits. Compared to Figure 5.13 and 5.14, when the disturbance signal is used at frequency 10 kHz and 100 kHz respectively, it is seen how the disturbance affect the DSL link. As mentioned in the previous chapter the width w of the disturbance is connected to its frequency, and the last two figures show how we can increase or decrease the frequency interval to affect in the DSL traffic by changing the frequency of the base-band signal. In the figures of quiet line noise (QLN) the noise level on the link is measured without any traffic active. In Figure 5.6 QLN is shown for the link without the disturbance, the disturbance seen in the interval of tone 170 to 190 (approximately), is an unknown external disturbance not yet revealed. Regardless of this disturbance the effect of our disturbance signal is clearly apparent in Figure 5.15 and Figure 5.16. The signal to noise ratio, SNR, for the same scenario is visible in Figure 5.7, 5.17 and finally 5.18.

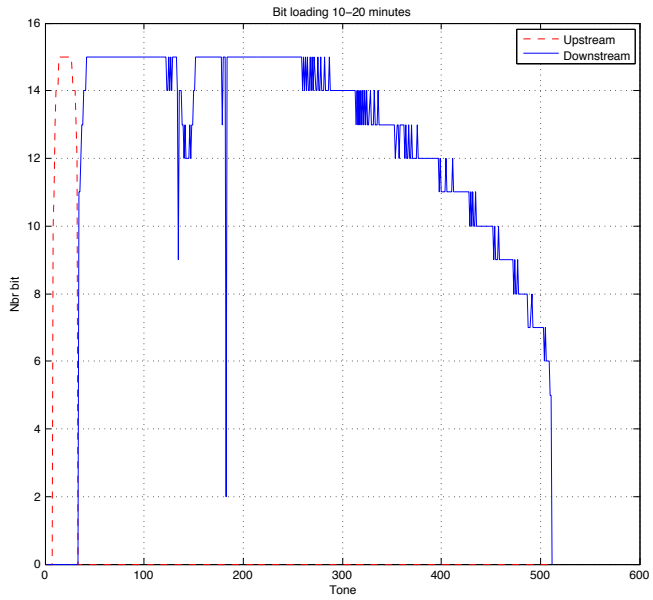


Figure 5.8: Bit-loading in time interval 10 to 20 minutes

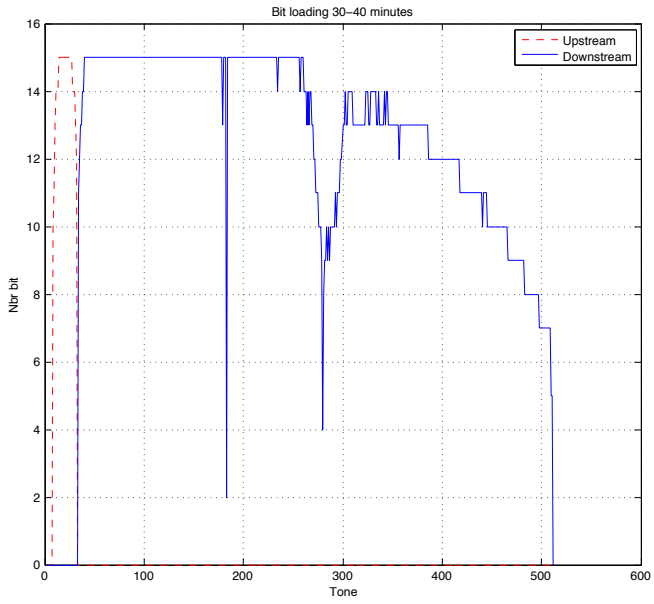


Figure 5.9: Bit-loading in time interval 30 to 40 minutes

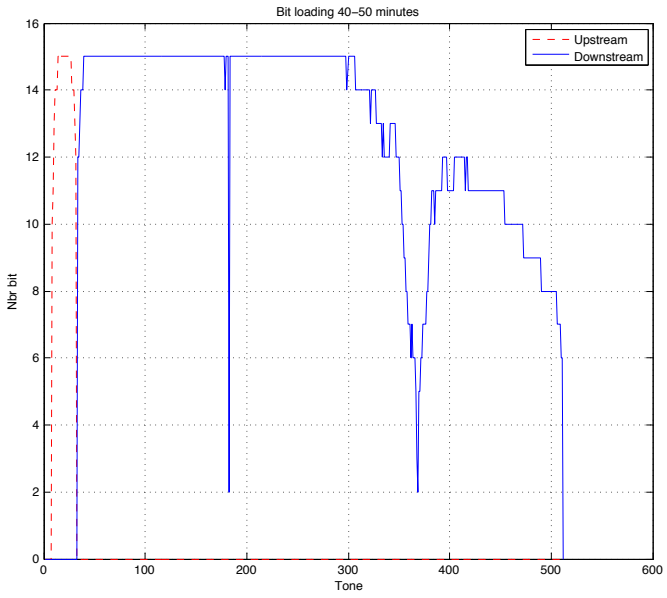


Figure 5.10: Bit-loading in time interval 40 to 50 minutes

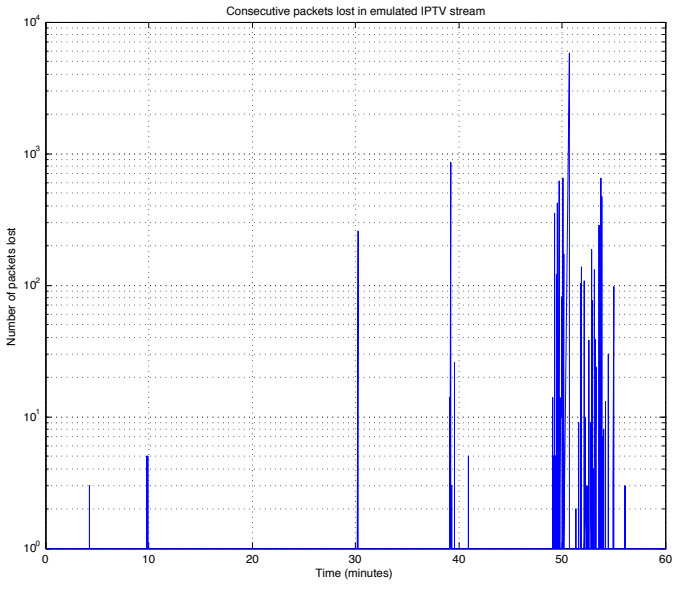


Figure 5.11: Number of packets lost

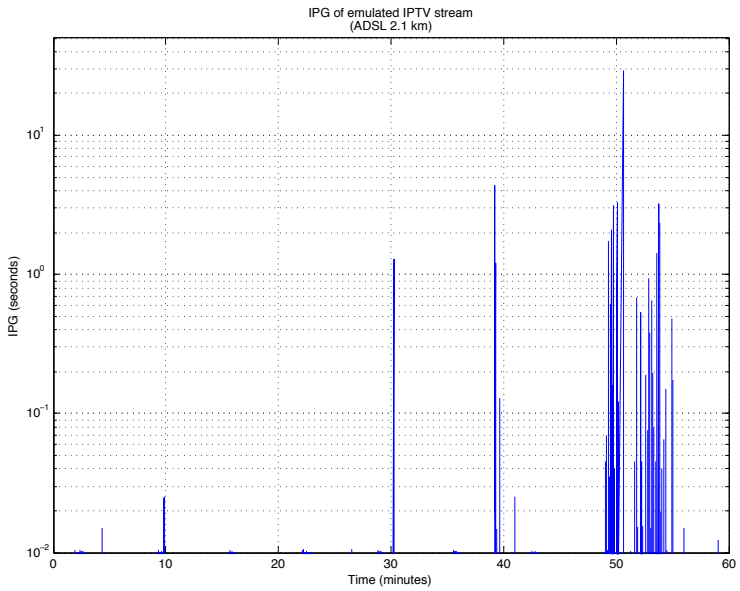


Figure 5.12: Inter Packet Gap in emulated IPTV

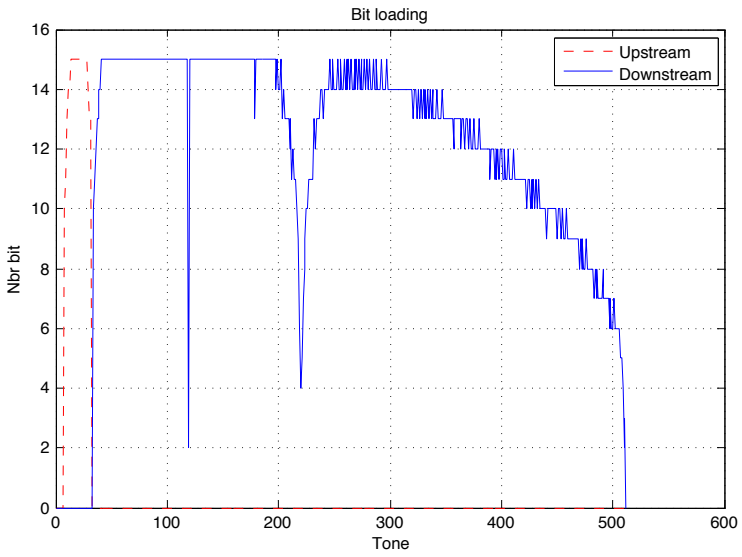


Figure 5.13: Bitloading with disturbance at $10kHz$ at cfq $1MHz$

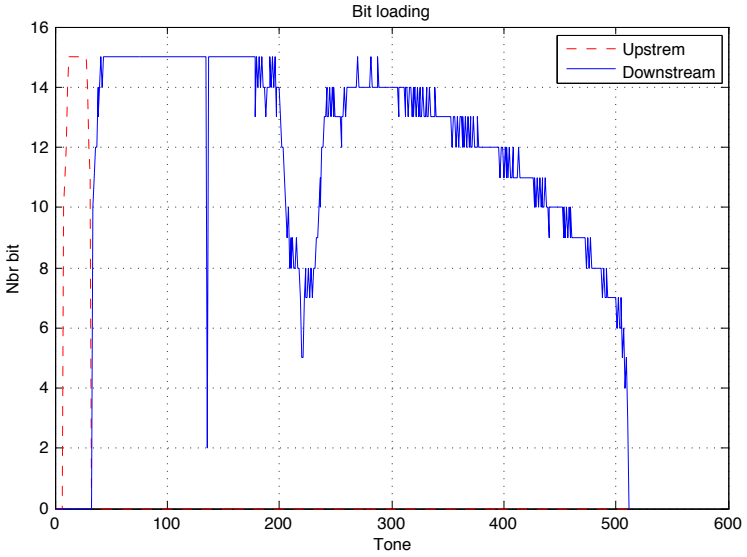


Figure 5.14: Bitloading with disturbance at 100kHz at cfq 1MHz

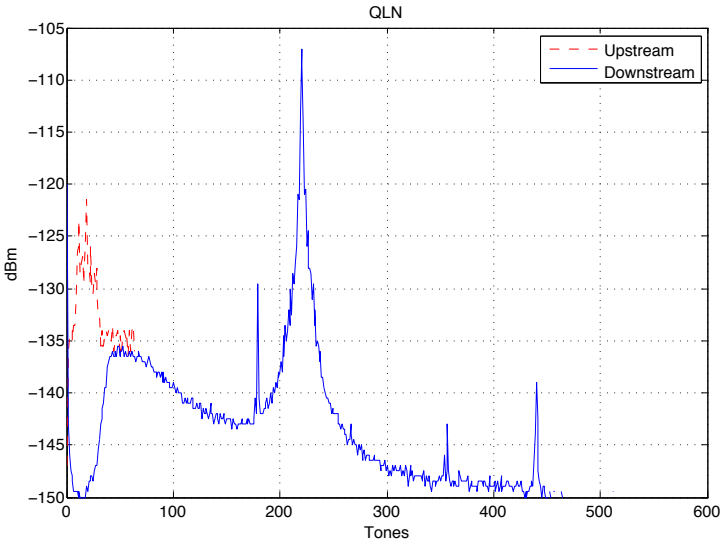


Figure 5.15: Quiet line noise with disturbance at 10kHz at cfq 1MHz

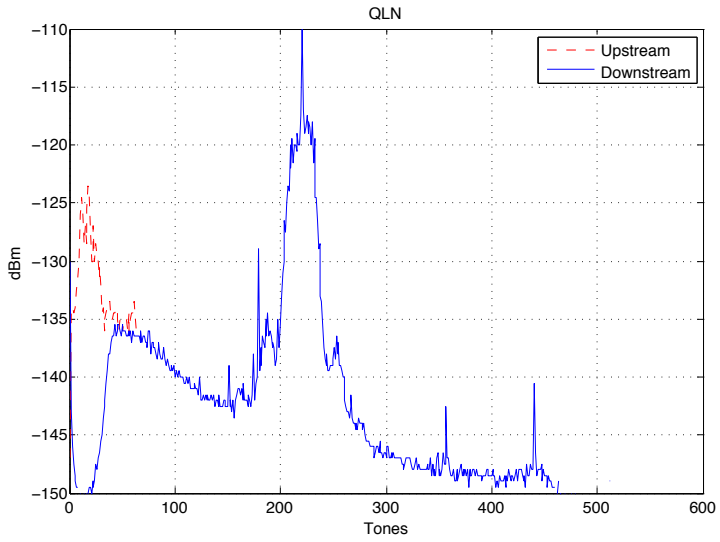


Figure 5.16: Quiet line noise with disturbance at 100kHz at cfq 1MHz

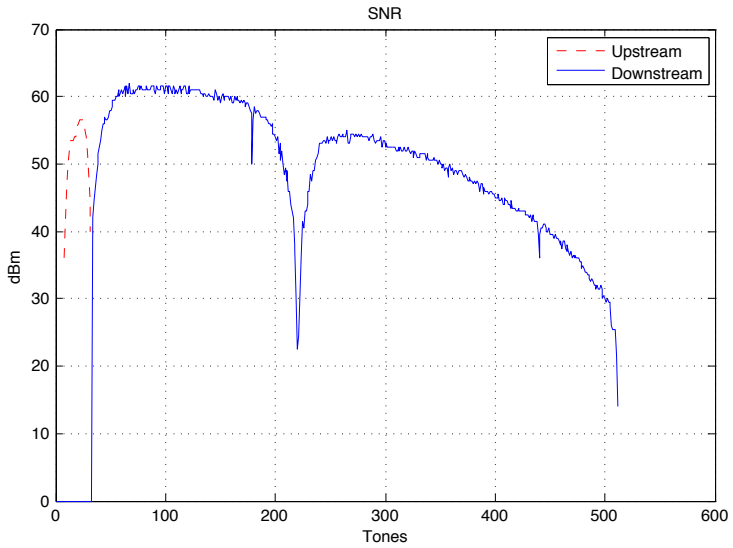


Figure 5.17: SNR with disturbance at 10kHz at cfq 1MHz

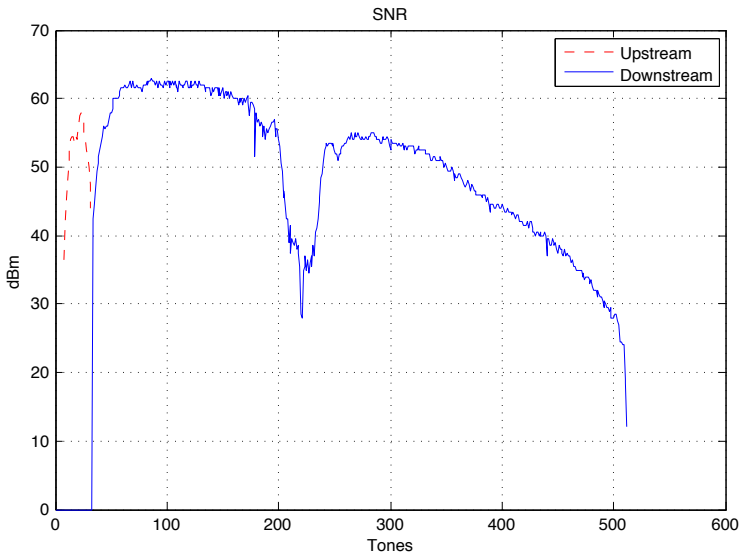


Figure 5.18: SNR with disturbance at 100kHz at cfq 1MHz

Chapter 6

Conclusion and Future Work

The work of classification, probing and correlation between QoS and QoE is an enormous task, especially when dealing with traffic over the best-effort channel on xDSL links. The core network of today is indeed a high speed network often based on fiber technology, but the access network, or the so called *last mile*, is still mostly old copper cables and is receptive for external disturbances. These disturbances can come from both nearby electronic equipment, such as voltage cables and transformers, and far away sources that radiates electromagnetic disturbances, such as radio stations. It is important to probe and classify the network traffic in an attempt of making the QoE better for end-customers. Probing can be both passive and active, in some cases a combination seems to be a strong candidate, taking advantage of both technologies. The active probing is well suited for end-to-end measurements but one must be aware of the load put on the network while sending packet trains across an already, sometimes heavily, loaded link. There is a lot of research done in the field of how to construct these packet trains and how to departure them in such a way that the final results are as accurate as possible. Also how cross-traffic affects these trains is of outmost importance to gain knowledge about and depends very much on the hardware such as routers and switches and how they handle traffic queues. Moreover, it is not just the overall loss rate or jitter, but also the characteristics of these QoS parameters that are important, with different loss patterns having different implications for different applications and in the end QoE for end-customers. For example, does packet loss occur independently as single events, or are they grouped together. The latter often indicates that packet losses are being caused by transient congestion, while the former may indicate physical layer problems (e.g. high noise levels on the link).

The passive approach clearly has the advantage of monitoring the network traffic in a completely non intrusive way and by that not effecting the network itself. The burden is instead put on the already, existing hardware

(e.g. routers and switches) and is not always suitable for time sensitive measurements. The more active technique of the passive ones are as mentioned network taps, these taps copies the whole traffic stream with physical errors and provides full visibility into full-duplex networks. One major question must be answered regardless of which technique used, if there really *is* a problem. OTT-traffic on the best-effort channel is just what it sounds like, traffic streaming over a best-effort channel. So to what degree will customers have to accept poor quality? If we dedicate more resources to a particular channel, is it still a best-effort channel?

Another important task is traffic classification, this can be done in several ways such as fast and simple port classification or more advanced statistical methods. Port classification still works very well but with an increasing amount of applications using dynamic ports it is not always a suitable solution. DPI investigates the actual payload instead. This however, demands well-identified signatures for those applications to be identified and lacks the ability to handle encrypted traffic. Signatures also needs to be up to date since applications change over time. It is also of the author's opinion that end-customers privacy must be kept in mind. The statistical approach on the other hand, shows interesting properties, privacy can be kept and still handles encrypted traffic. Machine learning, which is based on statistics, also handles a broad variety of traffic and can adapt and evolve more easily then other classification methods. What method to use to classify the traffic must be evaluated at a point after a decision is made regarding what traffic and/or application that is of interest.

In the work of mapping QoS parameters to expected QoE for end-users some studies have been made, the problem occurs when one aims at mapping diverse types of traffic, network traffic with different characteristics will be affected in different ways, making it hard to construct a general QoS/QoE map. As mentioned it is suggested to introduce an *expectation* term in the process of mapping since the expectation of a customer is connected to QoE. With more time a better correlation-study, regarding QoS and QoE, could be done. The controlled disturbance signal can be used in more extensive studies in future work and from the performed test it is clearly seen how great the impact is on an IPTV transmission. Most likely this effect will be noticeable for end-users using IPTV or other time sensitive media. The disturbance signal has the same characteristics in the frequency domain as distant radio stations.

This demonstrates how sensitive the DSL network is for not only nearby sources of disturbances, but also long distance ones. There is also a wish to improve the set-up for the generated signal. Further investigations regarding the unexplained disturbance around tone 200 kHz in the lab is also to be

investigated.

From both operators and customers point of view the subject of traffic classification and probing is of importance. The amount of traffic today is enormous in the network, and nothing point in the direction of that to change in the coming years. More time sensitive media is growing ground and QoS parameters are a key factor to maintain and even enhance the quality for end-customers. This demands for operators to be ably to discriminate traffic, especially the traffic paid for by end-customers. Probing is a way of gathering information about the QoS parameters in the network, once that step is taken traffic classification is the tool to change the situation and perhaps label some traffic with higher priority. Traffic classification and higher priority to time sensitive media will in the future most certainly make the QoE better for customers.

References

[1] *Martin, J.; Nilsson, A.;, On service level agreements for IP networks.INFOCOM 2002. Twenty-First Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, vol.2, no., pp.855-863, 2002*

[2] *Murphy, S.; Searles, M.; Rambeau, C.; Murphy, L.;, Evaluating the Impact of Network Performance on Video Streaming Quality for Categorized Video Content, 14th International Packet Video Workshop, 2004, vol., no., pp.1-8, 2002*

[3] *Balin, S.; Jinn, Y; Frost, V. S.; Duncan, T.;, Characterizing user-perceived impairment events using end-to-end measurements, John Wiley Sons, Ltd., International Journal of Communication Systems, vol.18, no.10, pp. 935-960, December 2005*

[4] *Jain, R.; Routhier, S.;, Packet trains – measurement and a new model for computer network traffic, IEEE Journal on Selected Areas in Communications, vol.4, no.6, pp.986-995, September 1986*

[5] *Brodie, M.; Rish, I.;Ma, S.; Grabarnik, G.; Odintsova, N.;, Active Probing, I.B.M. T.J. Watson Research, vol., no., pp.1-15, 2002*

[6] *Nilsson, M.; Measuring available path capacity using short probe trains, Swedish Institute of Computer Science (SICS), Network Operations and Management Symposium (NOMS), 2010 IEEE, vol., no., pp.910-913, 19-23 April 2010*

[7] *Ahlgren, B.; Björkman, M.; Melander B.;, Network Probing Using Packet Trains, Swedish Institute of Computer Science (SICS), vol., no., pp.1-6, 15 March, 1999*

[8] *Johnsson, A.; Melander, B.; Björkman, M.;, Analyzing Cross Traffic Effects on Packet TrainsUsing a Generic Multihop Model, The Department of Computer Science and Engineering, Mälardalen University of Sweden, vol., no., pp.1-9*

[9] *Agrawal, S. ; Naidu, K.V.M. ; Rastogi, R.;, Diagnosing Link-level Anomalies Using Passive Probes, INFOCOM 2007. 26th IEEE International Conference on Computer Communications, IEEE, vol., no., pp. 1757-1765, 6-12 May 2007*

[10] *Brodie, M.; Rish, I.; Ma, S.;, Optimizing Probe Selection for Fault Localization, IBM T.J. Watson Research Center, vol., no., pp.1-12*

[11] *Ishibashi, K.; Kanazawa, T.; Aida, M.;, Active/passive combination-type performance measurement method using change-of-measure framework, Global Telecommunications Conference, 2002. GLOBECOM '02. IEEE, NTT Inf. Sharing Platform Labs., NTT Corp., Tokyo, Japan, vol.3, no., pp. 2538-2542, 17-21 November 2002*

[12] *Dainotti, A.; de Donato, W.; Pescape, A.; Salvo Rossi, P.;, Classification of Network Traffic via Packet-Level Hidden Markov Models, Global Telecommunications Conference, 2008. IEEE GLOBECOM 2008. IEEE, vol., no., pp.1-5, 30 November 2008 - 4 December 20048*

[13] *Mohd, A. B.; Dr. Mohd, Nor S.;, Towards a Flow-based Internet Traffic Classification for Bandwidth Optimization, International Journal of Engineering, vol.3, no.4, pp.370-379, 2009*

[14] *Dornger, P.;, Real-Time Detection of Encrypted Traffic based on Entropy Estimation, Salzburg University of Applied Sciences Degree Program, vol., no., pp.1-89, August 2010*

[15] *Zhang, M.; John, W.; Claffy, K. C.; BrownleeN.;, State of the Art in Traffic Classification: A Research Review, PAM '09: 10th International Conference on Passive and Active Measurement, Student Workshop, vol., no., pp.1-2, 1-3 April 2009*

[16] *Yildirim, T.; Dr. Radcliffe, P. J.;, VoIP Traffic Classification in IPSec Tunnels, Electronics and Information Engineering (ICEIE), 2010 International Conference On, IEEE, vol.1 no., pp.V1-151 - V1-157, 2010*

[17] *Fan, J.; Nucci, A.; Keralapura, R.; Gao L.;, Protocol Oblivious Classification of Multimedia Traffic, University of Florida, Dept. of Electrical Computer Engineering, vol., no., pp.1-25, 2009*

[18] *Na, W.;, Design and implementation of RTP stream Classification Methods, National University of Singapore, vol., no., pp.1-110, 2004, [Online], Available: http://scholarbank.nus.edu.sg/handle/10635/13968, Viewed: June 2011*

[19] *Molnár, S.; Peréyi, M.;, On the identification and analysis of Skype traffic, Department of Telecommunications and Media informatic, Budapest Univeristy of Technology and Economics, International Journal of Communication Systems, 2011, vol.24, no.1, pp.94-117, 2010*

[20] *Valenti, S.; Rossi, D.; Meo, M.; Mellia, M.; Bermolen, P.;, Accurate, Fine-Grained Classification of P2P-TV Applications by Simply Counting Packets, Royal Institute of Technology, International In Traffic Measurement and Analysis (TMA) Workshop at IFIP Networking 2009, vol., no., pp.84-92, [Online], Available: http://www.csc.kth.se/ gkreitz/ spotify-p2p10/spotify-p2p10.pdf, Viewed: June 2011*

[21] *Schapire, R.;, Theoretical Machine Learning, Princeton University Computer Science Department, vol., no., pp.1-6, 2008, [Online], Available: http://www.cs.princeton.edu/courses/archive/spr08/cos511/* $scribe_notes/0204.pdf, Viewed : June2011$

[22] *Kreitz, G.; Niemelä, F.;, Spotify - Large Scale, Low Latency, P2P Music-on-Demand Streaming, KTH ? Royal Institute of Technology, Proceedings of IEEE P2P 2010, vol., no., pp.1-10,*

[23] *Spoto, S.; Gaeta, R.; Grangetto, M.; Sereno, M.;, Analysis of PPLive through active and passive measurements, Dipartimento di Informatica, Università di Torino, International Workshop on Hot Topics in Peer-to-Peer Systems, 2009. vol., no., pp.1-7, [Online], Available: http://www.di.unito.it/ mgrange/conf/pplive09.pdf, Viewed: June 2011*

[24] *[Online], Available:* http://www.ntop.org/blog/ntop/port-mirror-vs-network-tap/*, Viewed: June 2011*

[25] *Forouzan, B. A.;, Data Communication and Networking, 2nd edition, McGraw-Hill Book Co. ISBN: 0-07-118160-1, vol., no., pp.231-271, 2001,*

[26] *[Online], Available:* http://www.kitz.co.uk/adsl/adsl$_technology.htm, Viewed$: $June$2011

[27] FTDI webpage, [Online], Available: http://www.ftdichip.com/, *Viewed: June 2011*

Appendix A
Program Code

Listing A.1: Program

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Windows.Forms;

namespace WindowsFormsApplication1
{
    static class Program
    {
        /// <summary>
        /// The main entry point for the application.
        /// </summary>
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new Form1());
        }
    }
}
```

Listing A.2: Form1

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.Threading;

namespace WindowsFormsApplication1
{
    public partial class Form1 : Form
    {
        private double runTime = 0;
        private Controller myController;

        public Form1()
        {
           InitializeComponent();
           InitializeGUI();
        }


        /*
         * Sets up GUI
         */
        private void InitializeGUI()
```

```
{
    txtBaudrate.Text = string.Empty;
    errorOutput.Text = string.Empty;
    txtRunning.Text = string.Empty;
    txtTime.Text = string.Empty;
    btnStart.Enabled = true;
    btnStop.Enabled = false;
}


/*
 * Writes errors to form.
 */
public void ErrorOutput(string error)
{
    errorOutput.Text += error + "\n";
}


/*
 * Writes information to form.
 */
public void InformatinOutput(string information)
{
    infoOutPut.Text += information +"\n";
}


/*
 * Action taken if user push Stop-button.
 */
private void btnStop_Click(object sender, EventArgs e)
{
    DialogResult dlgResult = MessageBox.Show("Do you really want to
        close the program?","", MessageBoxButtons.YesNo, MessageBoxIcon.
        Question);
    if (dlgResult == DialogResult.Yes)
    {
        myController.Stop();
        ClearForm();
        btnStart.Enabled = true;
        btnStop.Enabled = false;
    }
}

/*
 * Writes runTime in form.
 */
private void timer1_Tick(object sender, EventArgs e)
{
    //Trigger update of txtProgressTime
    txtProgressTime.Text = myController.GetElapsedTimeString();
}


/*
 * Starts timer and checks if user has given a runTime, if so the
     progressbar also starts.
 */
private void InitializeMyTimer()
{
    runTime = myController.getRunTime();
    timer1.Enabled = true;
    // Start the timer.
    timer1.Start();
    // Set the interval for the timer.
    if (runTime != 0)
    {
        timer1.Interval = Convert.ToInt32(((runTime * 60) / 100) * 1000)
            ;
        progressBar1.Maximum = Convert.ToInt32(100);
        progressBar1.Minimum = 0;
        timer1.Tick += new EventHandler(IncreaseProgressBar);

    }
}


/*
```

```
     * Increase progressbar and check if it is at its end.
     */
    private void IncreaseProgressBar(object sender, EventArgs e)
    {
        // Increment the value of the ProgressBar a value of one each time.
        progressBar1.Increment(1);
        // Display the textual value of the ProgressBar in the StatusBar
            control's first panel.
        label1.Text = "Progress " + progressBar1.Value.ToString() + "%";
        // Determine if we have completed by comparing the value of the Value
            property to the Maximum value.
        if (progressBar1.Value == progressBar1.Maximum)
        {
            timer1.Stop();
            MessageBox.Show("Signalgenerator has ended");
            ClearForm();
        }
    }


    /*
     * Action taken when user push Start-button.
     */
    private void btnStart_Click(object sender, EventArgs e)
    {

        myController = new Controller(this);

        //Call for validation of input before run signalgenerator
        if (myController.ReadAndValidateInput(txtBaudrate.Text, txtTime.Text
            ))
        {
            InitializeMyTimer();
            txtRunning.Text = txtBaudrate.Text;
            myController.Run();
            btnStart.Enabled = false;
            btnStop.Enabled = true;
        }
    }


    /*
     * Clears the GUI.
     */
    public void ClearForm()
    {

        txtBaudrate.Text = string.Empty;
        errorOutput.Text = string.Empty;
        infoOutPut.Text = String.Empty;
        txtProgressTime.Text = String.Empty;
        txtRunning.Text = string.Empty;
        txtTime.Text = string.Empty;
        timer1.Stop();//Av-kallar timer1_Tick
        timer1.Enabled = false;
        progressBar1.Value = 0;
        label1.Text = progressBar1.Value.ToString();
        btnStart.Enabled = true;
        btnStop.Enabled = false;
    }


    /*
     * Following methods must excist even if the are empty.
     */
    private void txtRunning_TextChanged(object sender, EventArgs e)
    {

    }

    private void txtProgressTime_TextChanged(object sender, EventArgs e)
    {

    }

    private void errorOutput_TextChanged(object sender, EventArgs e)
    {

    }
```

```
        private void infoOutPut_TextChanged(object sender, EventArgs e)
        {

        }

    }
}
```

Listing A.3: Form1.Designer

```
namespace WindowsFormsApplication1
{
    partial class Form1
    {
        /// <summary>
        /// Required designer variable.
        /// </summary>
        private System.ComponentModel.IContainer components = null;

        /// <summary>
        /// Clean up any resources being used.
        /// </summary>
        /// <param name="disposing">true if managed resources should be disposed
            ; otherwise, false.</param>
        protected override void Dispose(bool disposing)
        {
            if (disposing && (components != null))
            {
                components.Dispose();
            }
            base.Dispose(disposing);
        }

        #region Windows Form Designer generated code

        /// <summary>
        /// Required method for Designer support - do not modify
        /// the contents of this method with the code editor.
        /// </summary>
        private void InitializeComponent()
        {
            this.components = new System.ComponentModel.Container();
            this.lblBaudrate = new System.Windows.Forms.Label();
            this.grpInput = new System.Windows.Forms.GroupBox();
            this.lblTime = new System.Windows.Forms.Label();
            this.txtTime = new System.Windows.Forms.TextBox();
            this.btnStop = new System.Windows.Forms.Button();
            this.btnStart = new System.Windows.Forms.Button();
            this.txtBaudrate = new System.Windows.Forms.TextBox();
            this.grpInfo = new System.Windows.Forms.GroupBox();
            this.lblInfoOutPut = new System.Windows.Forms.Label();
            this.lblError = new System.Windows.Forms.Label();
            this.infoOutPut = new System.Windows.Forms.RichTextBox();
            this.lblRunningTime = new System.Windows.Forms.Label();
            this.txtProgressTime = new System.Windows.Forms.TextBox();
            this.label1 = new System.Windows.Forms.Label();
            this.lblRunning = new System.Windows.Forms.Label();
            this.txtRunning = new System.Windows.Forms.TextBox();
            this.errorOutput = new System.Windows.Forms.RichTextBox();
            this.progressBar1 = new System.Windows.Forms.ProgressBar();
            this.timer1 = new System.Windows.Forms.Timer(this.components);
            this.grpInput.SuspendLayout();
            this.grpInfo.SuspendLayout();
            this.SuspendLayout();
            //
            // lblBaudrate
            //
            this.lblBaudrate.AutoSize = true;
            this.lblBaudrate.Location = new System.Drawing.Point(6, 26);
            this.lblBaudrate.Name = "lblBaudrate";
            this.lblBaudrate.Size = new System.Drawing.Size(69, 13);
            this.lblBaudrate.TabIndex = 0;
            this.lblBaudrate.Text = "Set Baudrate";
            //
            // grpInput
            //
            this.grpInput.Controls.Add(this.lblTime);
            this.grpInput.Controls.Add(this.txtTime);
            this.grpInput.Controls.Add(this.btnStop);
```

```
this.grpInput.Controls.Add(this.btnStart);
this.grpInput.Controls.Add(this.txtBaudrate);
this.grpInput.Controls.Add(this.lblBaudrate);
this.grpInput.Location = new System.Drawing.Point(12, 12);
this.grpInput.Name = "grpInput";
this.grpInput.Size = new System.Drawing.Size(355, 125);
this.grpInput.TabIndex = 1;
this.grpInput.TabStop = false;
this.grpInput.Text = "Input";
//
// lblTime
//
this.lblTime.AutoSize = true;
this.lblTime.Location = new System.Drawing.Point(113, 26);
this.lblTime.Name = "lblTime";
this.lblTime.Size = new System.Drawing.Size(230, 13);
this.lblTime.TabIndex = 5;
this.lblTime.Text = "Set run Time in minutes (leave empty for
    infinity)";
//
// txtTime
//
this.txtTime.Location = new System.Drawing.Point(113, 43);
this.txtTime.Name = "txtTime";
this.txtTime.Size = new System.Drawing.Size(100, 20);
this.txtTime.TabIndex = 4;
//
// btnStop
//
this.btnStop.Location = new System.Drawing.Point(104, 84);
this.btnStop.Name = "btnStop";
this.btnStop.Size = new System.Drawing.Size(75, 23);
this.btnStop.TabIndex = 3;
this.btnStop.Text = "Stopp";
this.btnStop.UseVisualStyleBackColor = true;
this.btnStop.Click += new System.EventHandler(this.btnStop_Click);
//
// btnStart
//
this.btnStart.Location = new System.Drawing.Point(7, 84);
this.btnStart.Name = "btnStart";
this.btnStart.Size = new System.Drawing.Size(75, 23);
this.btnStart.TabIndex = 2;
this.btnStart.Text = "Start";
this.btnStart.UseVisualStyleBackColor = true;
this.btnStart.Click += new System.EventHandler(this.btnStart_Click);
//
// txtBaudrate
//
this.txtBaudrate.Location = new System.Drawing.Point(7, 43);
this.txtBaudrate.Name = "txtBaudrate";
this.txtBaudrate.Size = new System.Drawing.Size(100, 20);
this.txtBaudrate.TabIndex = 1;
//
// grpInfo
//
this.grpInfo.Controls.Add(this.lblInfoOutPut);
this.grpInfo.Controls.Add(this.lblError);
this.grpInfo.Controls.Add(this.infoOutPut);
this.grpInfo.Controls.Add(this.lblRunningTime);
this.grpInfo.Controls.Add(this.txtProgressTime);
this.grpInfo.Controls.Add(this.label1);
this.grpInfo.Controls.Add(this.lblRunning);
this.grpInfo.Controls.Add(this.txtRunning);
this.grpInfo.Controls.Add(this.errorOutput);
this.grpInfo.Controls.Add(this.progressBar1);
this.grpInfo.Location = new System.Drawing.Point(12, 143);
this.grpInfo.Name = "grpInfo";
this.grpInfo.Size = new System.Drawing.Size(729, 323);
this.grpInfo.TabIndex = 2;
this.grpInfo.TabStop = false;
this.grpInfo.Text = "Information";
//
// lblInfoOutPut
//
this.lblInfoOutPut.AutoSize = true;
this.lblInfoOutPut.Location = new System.Drawing.Point(9, 127);
this.lblInfoOutPut.Name = "lblInfoOutPut";
this.lblInfoOutPut.Size = new System.Drawing.Size(92, 13);
this.lblInfoOutPut.TabIndex = 9;
```

```
this.lblInfoOutPut.Text = "Information output";
//
// lblError
//
this.lblError.AutoSize = true;
this.lblError.Location = new System.Drawing.Point(346, 120);
this.lblError.Name = "lblError";
this.lblError.Size = new System.Drawing.Size(62, 13);
this.lblError.TabIndex = 8;
this.lblError.Text = "Error output";
//
// infoOutPut
//
this.infoOutPut.AcceptsTab = true;
this.infoOutPut.Location = new System.Drawing.Point(6, 146);
this.infoOutPut.Name = "infoOutPut";
this.infoOutPut.Size = new System.Drawing.Size(337, 177);
this.infoOutPut.TabIndex = 7;
this.infoOutPut.Text = "";
this.infoOutPut.TextChanged += new System.EventHandler(this.
    infoOutPut_TextChanged);
//
// lblRunningTime
//
this.lblRunningTime.AutoSize = true;
this.lblRunningTime.Location = new System.Drawing.Point(12, 72);
this.lblRunningTime.Name = "lblRunningTime";
this.lblRunningTime.Size = new System.Drawing.Size(49, 13);
this.lblRunningTime.TabIndex = 6;
this.lblRunningTime.Text = "Runtime:";
//
// txtProgressTime
//
this.txtProgressTime.Location = new System.Drawing.Point(12, 88);
this.txtProgressTime.Name = "txtProgressTime";
this.txtProgressTime.Size = new System.Drawing.Size(100, 20);
this.txtProgressTime.TabIndex = 5;
this.txtProgressTime.TextChanged += new System.EventHandler(this.
    txtProgressTime_TextChanged);
//
// label1
//
this.label1.AutoSize = true;
this.label1.Location = new System.Drawing.Point(131, 69);
this.label1.Name = "label1";
this.label1.Size = new System.Drawing.Size(165, 13);
this.label1.TabIndex = 4;
this.label1.Text = "Progres____(When running on time)";
//
// lblRunning
//
this.lblRunning.AutoSize = true;
this.lblRunning.Location = new System.Drawing.Point(9, 29);
this.lblRunning.Name = "lblRunning";
this.lblRunning.Size = new System.Drawing.Size(53, 13);
this.lblRunning.TabIndex = 3;
this.lblRunning.Text = "Baudrate:";
//
// txtRunning
//
this.txtRunning.Location = new System.Drawing.Point(12, 45);
this.txtRunning.Name = "txtRunning";
this.txtRunning.Size = new System.Drawing.Size(80, 20);
this.txtRunning.TabIndex = 2;
this.txtRunning.TextChanged += new System.EventHandler(this.
    txtRunning_TextChanged);
//
// errorOutput
//
this.errorOutput.Location = new System.Drawing.Point(349, 146);
this.errorOutput.Name = "errorOutput";
this.errorOutput.Size = new System.Drawing.Size(337, 177);
this.errorOutput.TabIndex = 1;
this.errorOutput.Text = "";
this.errorOutput.TextChanged += new System.EventHandler(this.
    errorOutput_TextChanged);
//
// progressBar1
//
this.progressBar1.Location = new System.Drawing.Point(134, 85);
```

```
            this.progressBar1.Name = "progressBar1";
            this.progressBar1.Size = new System.Drawing.Size(100, 23);
            this.progressBar1.TabIndex = 0;
            //
            // timer1
            //
            this.timer1.Tick += new System.EventHandler(this.timer1_Tick);
            //
            // Form1
            //
            this.AutoScaleDimensions = new System.Drawing.SizeF(6F, 13F);
            this.AutoScaleMode = System.Windows.Forms.AutoScaleMode.Font;
            this.ClientSize = new System.Drawing.Size(775, 478);
            this.Controls.Add(this.grpInfo);
            this.Controls.Add(this.grpInput);
            this.Name = "Form1";
            this.Text = "Form1";
            this.grpInput.ResumeLayout(false);
            this.grpInput.PerformLayout();
            this.grpInfo.ResumeLayout(false);
            this.grpInfo.PerformLayout();
            this.ResumeLayout(false);

        }

        #endregion

        private System.Windows.Forms.Label lblBaudrate;
        private System.Windows.Forms.GroupBox grpInput;
        private System.Windows.Forms.Button btnStop;
        private System.Windows.Forms.Button btnStart;
        private System.Windows.Forms.TextBox txtBaudrate;
        private System.Windows.Forms.GroupBox grpInfo;
        private System.Windows.Forms.RichTextBox errorOutput;
        private System.Windows.Forms.ProgressBar progressBar1;
        private System.Windows.Forms.Timer timer1;
        private System.Windows.Forms.Label lblRunning;
        private System.Windows.Forms.TextBox txtRunning;
        private System.Windows.Forms.Label lblTime;
        private System.Windows.Forms.TextBox txtTime;
        private System.Windows.Forms.Label label1;
        private System.Windows.Forms.TextBox txtProgressTime;
        private System.Windows.Forms.Label lblRunningTime;
        private System.Windows.Forms.Label lblInfoOutPut;
        private System.Windows.Forms.Label lblError;
        private System.Windows.Forms.RichTextBox infoOutPut;
    }
}
```

Listing A.4: Controller

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;
using System.Threading;


namespace WindowsFormsApplication1
{
    class Controller
    {
        private double runTime;
        private int baudRate;
        private Timer myTimer;
        private Form1 myForm;
        Thread myThread;
        private SignalGenerator signalGenerator;

        /*
         * Constructor
         */
        public Controller(Form1 iMyForm)
        {
```

```
        myForm = iMyForm;
    }

    /*
     * Set up Controller and runs the Signalgenerator
     */
    public void Run()
    {
        SetUp();
        if (!signalGenerator.SetUp())
        {
            myForm.ErrorOutput("Failed to set up USB device");
        }
        else
        {
            myTimer.Start();
            myThread.Start();
        }
    }


    /*
     * Starts a new thread that handles the signalgenerator.
     * Starts a new timmer with time chossen by user.
     */
    public void SetUp()
    {
         myTimer = new Timer(runTime);
         signalGenerator = new SignalGenerator(baudRate, this);
         myThread = new Thread(signalGenerator.Run);
    }


    /*
     * Stops the generator and timer, also ends the thread handeling the
         generator
     */
    public void Stop()
    {
        signalGenerator.Stop();
        myTimer.Stop();
        myThread.Abort();
    }


    /*
     * Displays the information in the form.
     */
    public void DisplayInformationOutoput(string information)
    {
        myForm.InformatinOutput(information);
    }

    /*
     * Displays the error-information in the form.
     */
    public void DisplayErrorOutput(string error)
    {
        myForm.ErrorOutput(error +"\n");
    }


    /*
     * Validates inputdata from user field.
     */
    public bool ReadAndValidateInput(string baudrate, string time)
    {
        bool timeValidated = ValidateTime(time);
        bool baudrateValidated = ValidateBaudrate(baudrate);
        //Check if both fields are empty
        if(String.IsNullOrEmpty(time) && String.IsNullOrEmpty(baudrate))
        {
            DisplayErrorOutput("You cant leave both fields empty");
            return false;
        }
        //Check if only Baudrate is given
        else if ((!String.IsNullOrEmpty(baudrate)) && String.IsNullOrEmpty(
            time))
        {
            if (baudrateValidated)
```

```
            {
                return true;
            }
            else
                DisplayErrorOutput("Enter a correct value for baudrate");
                return false;
        }
        //Check if only time is given
        else if ((String.IsNullOrEmpty(baudrate)) && (!String.IsNullOrEmpty(
            time)))
        {
            DisplayErrorOutput("Enter a correct value for baudrate");
            return false;
        }
        else if(timeValidated && baudrateValidated)
        {
                return true;
        }
        else
            DisplayErrorOutput("Enter correct values for baudrate and time")
                ;
            return false;
    }

    /*
     * Validates user input of baudrate.
     */
    private bool ValidateBaudrate(string baudrate)
    {
        return (int.TryParse(baudrate, out baudRate));
    }

    /*
    * Validates user input of time.
    */
    private bool ValidateTime(string time)
    {
        return (double.TryParse(time, out runTime));
    }

    /*
     * Get the baudrate.
     */
    public int getBaudRate()
    {
        return baudRate;
    }

    /*
     * Get the runTime.
     */
    public double getRunTime()
    {
        return runTime;
    }

    /*
     * Check if runTime is ended.
     */
    public bool IsTimerDone()
    {
        return myTimer.Done();
    }

    /*
     * Get elapsed time.
     */
    public double GetElapsedTimeSecs()
    {
        return myTimer.GetElapsedTimeSecs();
    }

    /*
```

```
         * Get elapsedtime in Strin form.
         */
        public string GetElapsedTimeString()
        {
            return myTimer.GetElapsedTimeString();
        }

    }
}
```

Listing A.5: SignalGenerator

```
using System;
using System.Collections.Generic;
using System.Linq;
using System.Text;
using System.Threading;
using FTD2XX_NET;
using System.Windows.Forms;
using System.Diagnostics;


namespace WindowsFormsApplication1
{

    class SignalGenerator
    {

        private bool runCheck;
        UInt32 BaudRate;
        UInt32 ftdiDeviceCount = 0;
        FTDI.FT_STATUS ftStatus = FTDI.FT_STATUS.FT_OK;
        // Create new instance of the FTDI device class
        FTDI myFtdiDevice = new FTDI();
        Controller myController;


        /*
         * Constructor
         *
         * */
        public SignalGenerator(int baudRate, Controller iController)
        {
            BaudRate = Convert.ToUInt32(baudRate);
            myController = iController;
         }


        /*
         * Initialize USB device.
         *
         */
        public bool SetUp(){

            runCheck = true;
            // Determine the number of FTDI devices connected to the machine
            ftStatus = myFtdiDevice.GetNumberOfDevices(ref ftdiDeviceCount);
            // Check status
            if (ftStatus == FTDI.FT_STATUS.FT_OK)
            {
                myController.DisplayInformationOutoput("Number of FTDI devices:
                    " + ftdiDeviceCount.ToString());
            }
            else
            {
                myController.DisplayErrorOutput("Failed to get number of devices
                    (error " + ftStatus.ToString() + ")");
                runCheck = false;
            }


            // If no devices available, return
            if (ftdiDeviceCount == 0)
            {
                myController.DisplayErrorOutput("Failed to get number of devices
                    (error " + ftStatus.ToString() + ")");
                runCheck = false;
            }
```

```
// Allocate storage for device info list
FTDI.FT_DEVICE_INFO_NODE[] ftdiDeviceList = new FTDI.
    FT_DEVICE_INFO_NODE[ftdiDeviceCount];

// Populate our device list
ftStatus = myFtdiDevice.GetDeviceList(ftdiDeviceList);

if (ftStatus == FTDI.FT_STATUS.FT_OK)
{
    for (UInt32 i = 0; i < ftdiDeviceCount; i++)
    {
        myController.DisplayInformationOutoput("Device Index: " + i.
            ToString());
        myController.DisplayInformationOutoput("Flags: " + String.
            Format("{0:x}", ftdiDeviceList[i].Flags));
        myController.DisplayInformationOutoput("Type: " +
            ftdiDeviceList[i].Type.ToString());
        myController.DisplayInformationOutoput("ID: " + String.
            Format("{0:x}", ftdiDeviceList[i].ID));
        myController.DisplayInformationOutoput("Location ID: " +
            String.Format("{0:x}", ftdiDeviceList[i].LocId));
        myController.DisplayInformationOutoput("Serial Number: " +
            ftdiDeviceList[i].SerialNumber.ToString());
        myController.DisplayInformationOutoput("Description: " +
            ftdiDeviceList[i].Description.ToString());
        myController.DisplayInformationOutoput("");
    }
}
else
{
    myController.DisplayErrorOutput("Failed to populate devicelist (
        error " + ftStatus.ToString() + ")");
    runCheck = false;
}


// Open first device in our list by serial number
ftStatus = myFtdiDevice.OpenBySerialNumber(ftdiDeviceList[0].
    SerialNumber);
if (ftStatus != FTDI.FT_STATUS.FT_OK)
{
    myController.DisplayErrorOutput("Failed to open device (error "
        + ftStatus.ToString() + ")");
    runCheck = false;
}


// Set Baud rate
ftStatus = myFtdiDevice.SetBaudRate(BaudRate);
if (ftStatus != FTDI.FT_STATUS.FT_OK)
{
    myController.DisplayErrorOutput("Failed to set BaudRate (error "
        + ftStatus.ToString() + ")");
    runCheck = false;
}


// Set data characteristics - Data bits, Stop bits, Parity
ftStatus = myFtdiDevice.SetDataCharacteristics(FTDI.FT_DATA_BITS.
    FT_BITS_8, FTDI.FT_STOP_BITS.FT_STOP_BITS_1, FTDI.FT_PARITY.
    FT_PARITY_NONE);
if (ftStatus != FTDI.FT_STATUS.FT_OK)
{
    myController.DisplayErrorOutput("Failed to set data
        characteristics (error " + ftStatus.ToString() + ")");
    runCheck = false;
}

// Set flow control - set RTS/CTS flow control
ftStatus = myFtdiDevice.SetFlowControl(FTDI.FT_FLOW_CONTROL.
    FT_FLOW_NONE, 0x11, 0x13);
if (ftStatus != FTDI.FT_STATUS.FT_OK)
{
    myController.DisplayErrorOutput("Failed to set flow control (
        error " + ftStatus.ToString() + ")");
    runCheck = false;
}
```

```
            // Set read timeout to 5 seconds, write timeout to infinite
            ftStatus = myFtdiDevice.SetTimeouts(5, 0);
            if (ftStatus != FTDI.FT_STATUS.FT_OK)
            {
                myController.DisplayErrorOutput("Failed to set timeouts (error "
                    + ftStatus.ToString() + ")");
                runCheck = false;
            }

            return runCheck;

        }


        /*
         * Stops the output signal
         */
        public void Stop()
        {
            runCheck = false;
            CloseUSB();
        }


        /*
         * Starts the output signal
         */
        public void Run()
        {
            //Random bytes written to USB unit
            byte[] array = new byte[1000];
            Random random = new Random();

            while (runCheck && myController.IsTimerDone())
            {
                //Get next byte
                random.NextBytes(array);
                UInt32 numBytesWritten = 0;
                // Note that the Write method is overloaded, so can write string
                    or byte array data
                ftStatus = myFtdiDevice.Write(array, array.Length, ref
                    numBytesWritten);

                if (ftStatus != FTDI.FT_STATUS.FT_OK)
                {
                    myController.DisplayErrorOutput("Failed to write to device (
                        error " + ftStatus.ToString() + ")");
                }
            }
        }


        /*
         * Closes the USB device
         */
        public bool CloseUSB()
        {
            ftStatus = myFtdiDevice.Close();
            if (ftStatus == FTDI.FT_STATUS.FT_OK)
                return true;
            else
                return false;
        }
    }
}
```

Listing A.6: Timer

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.Diagnostics;

namespace WindowsFormsApplication1
```

```
{
    class Timer
    {

        public DateTime startTime;
        public DateTime stopTime;
        private  DateTime currentTime;
        private double runTime;
        private bool running;
        public double elapsedTime;
        double remainingTime;


        /*
         * Constructor.
         */
        public Timer(double runTime)
        {
            this.runTime = runTime;
        }


        /*
         * Starts the timer.
         */
        public void Start()
        {
            startTime = DateTime.Now;
            running = true;
        }


        /*
         * Stops the timer
         */
        public void Stop()
        {
            this.stopTime = DateTime.Now;
            running = false;
        }


        /*
         * Checks if timer is done.
         */
        public bool Done()
        {

            elapsedTime = GetElapsedTimeSecs();

            remainingTime = (runTime*60) - elapsedTime;

            if (remainingTime > 0 || runTime == 0)
            {
                return true;

            }
            else
            {
                return false;
            }
        }



        /*
         * Get elaspsed time in String form.
         */
        public string GetElapsedTimeString()
        {
            TimeSpan interval;

            if (!running)
                return ""; //interval = DateTime.Now - startTime;
            else
            {
                this.currentTime = DateTime.Now;
                interval = currentTime - startTime;
            }
```

```
            int days = interval.Days;
            double hours = interval.Hours;
            double mins = interval.Minutes;
            double secs = interval.Seconds;
            string x = "";
            if (days != 0)
            {
                x += days.ToString() + ":";
            }
            if (hours != 0)
            {
                x += hours.ToString("00") + ":";
            }
            x += mins.ToString("00") + ":";
            x += secs.ToString("00");

            return x;
        }


        /*
        * Get elaspsed time in double form.
        */
        public double GetElapsedTimeSecs()
        {
            TimeSpan interval;

            if (running)
                interval = DateTime.Now - startTime;
            else
                interval = stopTime - startTime;

            return interval.TotalSeconds;
        }


    }
}
```

Appendix B
Wiring Diagram

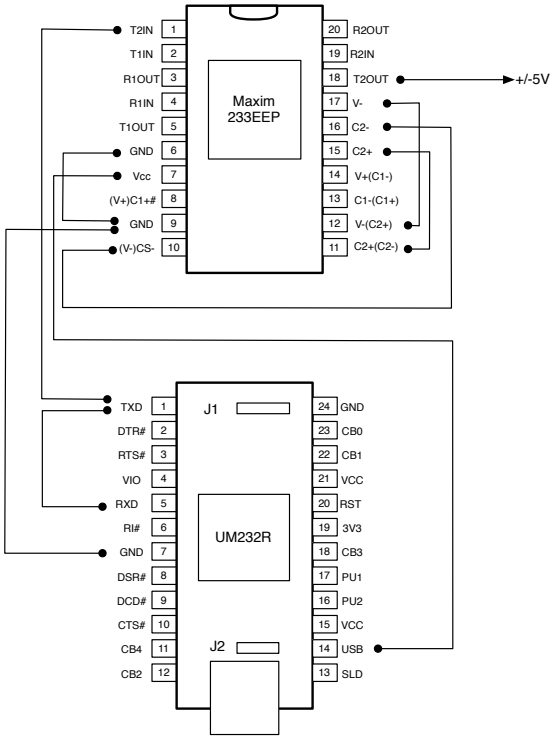


Figure B.1: Circuit diagram over UM232R and Maxim 233EEP