Evaluating Retransmission as Technique to Improve Streaming of TV Audio on a Wireless Network

Kim Højgaard-Hansen

Kristian Engh Lundgreen

AALBORG UNIVERSITY

10th Semester Project, June 2009

The Faculties of Engineering, Science and Medicine

Department of Electronic Systems

AALBORG UNIVERSITET

Institute of Electronic Systems

Fredrik Bajers Vej 7

9220 Aalborg Øst

Telephone: 99 40 86 00

http://www.ies.aau.dk/

Title:

Evaluating Retransmission as Technique to Improve Streaming of TV Audio on a Wireless Network

Theme:

Performance Analysis and Network Planning

Project period:

10th Semester, Spring 2009

Project group:

Group 1005

Group members:

Kim Højgaard-Hansen

Kristian Engh Lundgreen

Supervisor:

Anders Nickelsen

Tatiana K. Madsen

Number printed:

5

Number of pages:

Report: 78

Total: 80 Appendix: One CD-ROM

Abstract:

This work investigates the use-case of streaming audio from a TV to a number of headset using WLAN. During this investigation two requirements are identified: Lip synchronization between audio and video requires a maximum skew of 80ms. An acceptable audio experience requires a maximum packet loss of 1.8%

Initial experiments showed that packet loss is the main problem. An analysis is made to find techniques for packet loss recovery, and retransmission is chosen as the packet loss recovery technique to be investigated.

A new method of measuring whether the skew requirement is fulfilled is proposed. 802.11 MAC retransmission, UDP, TCP and a developed simple application level retransmission protocol were all evaluated in terms of their performance regarding the skew requirement, the ability to recover from packet loss and the bandwidth usage.

It is concluded that there is no significant gain in using more than 802.11 MAC retransmission to recover from packet loss when transmitting unicast streams with high delay requirements.

Completed:

June 3rd, 2009

Frontpage photo:

Creative Commons, Copyright: http://www.instructables.com

Preface

This report is written during the 10th semester of Networks and Distributed Systems (NDS) studies at the department of Electronic Systems at Aalborg University. The primary purpose of this semester is to document that the student independently or in a small group is capable of planning and completing a project at a technically high level. The final thesis must document the students ability to apply scientific theories and methods. The project started February 2th 2009 and ended June 3rd 2009.

This project is based on a proposal constructed in cooperation with DoréDevelopment Aps, Hadsund, Denmark. The proposal asked for an evaluation on whether it is possible to use a Wireless Local Area Network (WLAN) as carrying media for a live audio stream from a television, with strict requirements in time. The wireless medium is influenced by noise and other communicating nodes, and is thereby considered unreliable. This led to the main objective of this report; an evaluation of retransmission techniques to improve live streaming of TV audio on such networks.

References to source material are indicated as [1] which refers to the numbered list of references found at the end of the report. Figures, tables, equations and text sections are referred to by the number corresponding to the object. Acronyms are written in full length when they are introduced e.g. Line Of Sight (LOS) and a list of acronyms can be found next to the bibliography.

A CD is enclosed with the report which contains the source code for the data processing of the measurement results as well as an electronic copy of this report in PDF format.

Definitions

Throughout the report a phrase 'inter packet delay' is used, as another therm for inter packet arrival. This is defined as the time difference between two successive received packets. The terms 'inter packet delay' and 'inter packet arrival' is used interchangeable.

Contents of the CD-ROM

- This report in PDF format (`/report.pdf`)
- Python scripts used for calculations
- Experimental data

Acknowledgments

The group would like to thank Mads Doré, Mads Lange, Esben Haabendal and all other employees at DoréDevelopment for their help and cooperation during the project. A special acknowledgement is given to our two supervisors: Anders Nickelsen and Tatiana K. Madsen.

Author Signatures

Aalborg East, June 3th 2009

Kim Højgaard-Hansen

Aalborg East, June 3th 2009

Kristian Engh Lundgreen

Contents

1 Introduction

Open office environments are more and more in use by companies who practise working in project groups. Placing people who works on the same project near each other, without delimiting walls, increases the knowledge sharing and possible the efficiency of the team.

However, this type of open office environment introduces some problems with light, noise and other disturbances. People working in such an environment has different needs, such as telephony conversations and informal meetings in order to do their jobs. Many people starts listening to music or radio stations using headphones to reduce the noise from people around them.

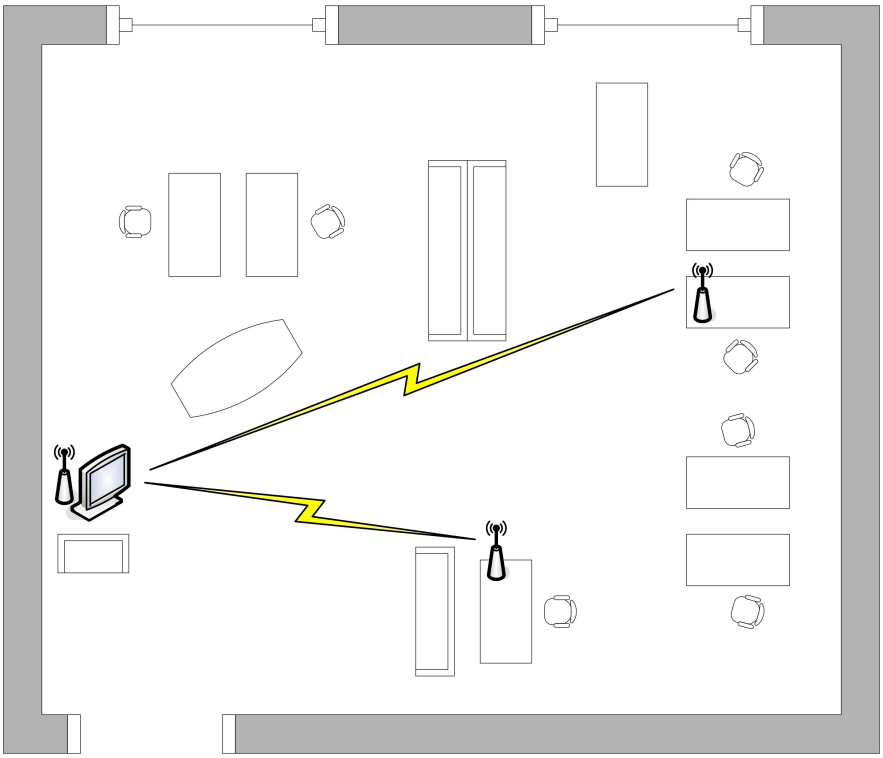

Figure 1.1: The figure shows an overview of the open office environment with the flat screen TV at one wall. Desks are spread around the office, and each of these desks could be a potential receiver of the streamed audio.

In an open office environment a large flat screen TV can potentially be shared among the employees, watching the news or other relevant shows, sometimes even with split-screen of two different shows. This sharing does however create a problem when it comes to the sound from the TV. An open office environment is very sensitive to disturbance hence it is not wishful to have the volume from the TV set

high enough that everybody can listen to it. This project will focus on how it is possible for a company to give their employees the opportunity without disturbing their colleagues.

An example of such a company is DoreDevelopment which will be the case study throughout this project.

1.1 Use-case

DoreDevelopment is a small firm developing and managing embedded software solutions. Their open office environment has room for 7 people, where a 40 inches flat panel screen let the employees watch the news trough the working day. DoreDevelopment need a solution for streaming the sound of the TV channel to the employees that are watching, but without introducing noise to the other colleagues. This project proposes an audio streaming solution, which captures the audio from the TV and streams it to a number of employees using wireless technology. By streaming the audio to a number of headsets, the disturbance from the TV is removed. The concept is illustrated in Figure 1.1.

DoreDevelopment would like to reuse their existing Wireless LAN infrastructure, to avoid establishing a whole new network to transport the audio stream. Streaming on a unreliable wireless link, affected by noise, introduces some interesting aspects due to media access time and packet loss caused by the use of a shared medium. This leads to the initial problem statement:

- How can the audio from a TV be distributed to a number of receivers using WLAN without affecting the TV watching experience?

The following chapter will give an overview of the live streaming scenario and the requirements used throughout the project. The chapter will furthermore describe the initial experiment performed.

2 Preliminary Analysis

Section 2.1 will introduce the concept of streaming and introduce some important definitions and terms in order to describe the synchronization problem in detail.

2.1 Live streaming overview

Main source: [1]

Live streaming in it is most abstracted form can be seen in Figure 2.1. A recording is made consisting of either audio or video or both and this is fed into the source end of the stream. The source is connected to the sink end, possibly by a network, and at the sink end the stream is played.

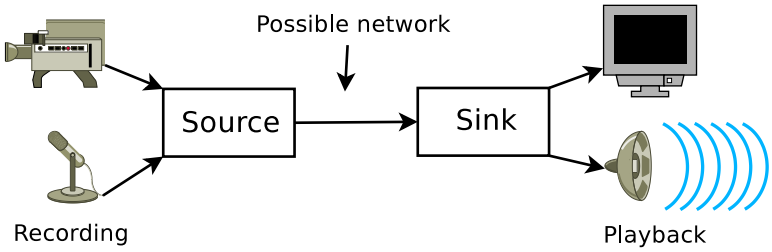

Figure 2.1: The figure shows how live streaming works. A recording is made with either audio, video or both, which is fed into the source. The source is connected to the sink possible via a network. At the sink the recording is played live.

A stream of either audio or video or both is characterized as time-dependant since the media objects which constitutes the stream are dependant on each other with relation to time. An example could be video and audio objects both recorded at a concert. These objects have a time-dependant relationship when they are recorded, and this relationship has to be preserved when the objects are played again.

A recorded stream is usually split up into a sequence of smaller information units. From now on the term Logical Data Unit (LDU) will be used about these information units. The size of these LDUs is dependant on the type of stream and the application creating them e.g. for CD-quality music, Pulse Code Modulation (PCM) coding without compression is used with a sample rate of 44100Hz, two channels and 16bit resolution per channel, which are combined to blocks of $\frac{1}{75}$seconds duration. LDUs can be classified as either closed LDUs or as open LDUs. A closed LDU have a predictable duration e.g. from continuous media like audio and video. The duration of open LDUs are not predictable and could be media objects that include user interaction. Examples of closed LDUs for video can be seen in Figure 2.2 where each picture/frame is packaged into one LDU.

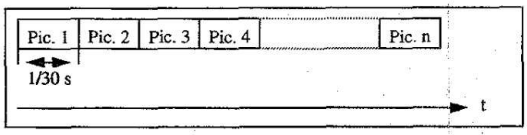

Figure 2.2: The figure shows an example of a video LDU. Each sample (picture/frame) is put into one LDU of 1/30s duration.

Since audio samples are much smaller than video samples, the LDUs for an audio stream usually contains a number of samples. Figure 2.3 shows an example where 512 samples is packaged into one LDU.

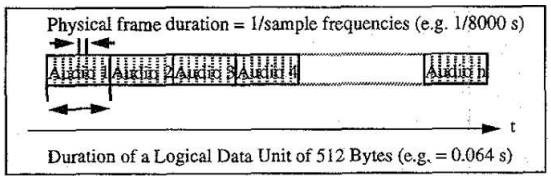

Figure 2.3: The Figure shows an example of an audio LDU. Since audio samples are small compared to video samples it is normal to package a number of these in each LDU. Her one sample is 1/8000s of audio, and 512 of these are packaged into one LDU.

A more detailed view of how the project use case streaming works is illustrated in Figure 2.4. The TV broadcast can be viewed as a kind of streaming which ends in the TV tuner as sink. From the TV to the headsets another streaming is performed with the TV as the live source and the audio capturing and transmitting device as the stream source. The audio streams is sent to the headsets which becomes the stream sinks.

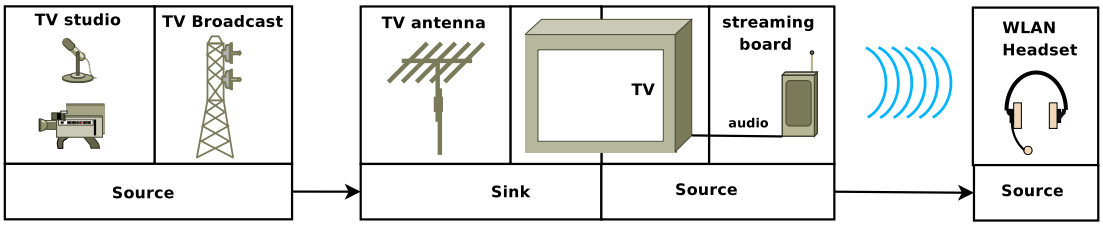

Figure 2.4: The figure shows a more detailed view of how the project use case streaming is done. A show is recorder or played from a TV studio and broadcast using the TV broadcast network to the project use case TV. This could be seen as one live stream. The TV tuner decodes the "TV stream" and the picture is shown on the TV while the audio is streamed using WLAN to a number of headsets.

The next sections will derive a set of requirements to the stream solution.

2.2 Synchronization Requirement

Splitting a TV signal/stream into two separate streams with audio and video, set up a requirement about the synchronization between these. This is called "Lip Synchronization" and refers to the temporal relationship between audio and video for the particular case of human speaking. The time difference between the audio and video LDUs is called skew. If the streams are perfectly in sync there is no skew (0ms difference). When the streams are not in perfect synchronization this can become a problem to the user experience, hence it is necessary to know how much skew is tolerable.

It is the human perception of the synchronization which is the measurement of whether two streams are "in-sync" or "out-of-sync". This is not an objective measurement because the perception of synchronization varies from person to person. Instead a heuristic criteria is needed. Experiments conducted at the IBM European Networking Center[13] gave the results shown in Figure 2.5. The region from -80ms (audio behind video) to 80ms was concluded to be the in-sync region since most of the subjects did not notice synchronization errors there. The region below -160ms and above 160ms was concluded to be out-of-sync since nearly all the subjects detected errors there. The area between -160ms and -80ms as well as between 80ms and 160ms was named the transient area since the subjects started to detect errors there. It was noted that when the resolution of the picture was better or the closer the speaker was to the camera, the easier it was to detect the errors. It was also noted that video ahead of audio could be tolerated better than audio ahead of video. The latter can be explained by the fact that this is a situation which is not unusual to humans since light travels faster than sound.

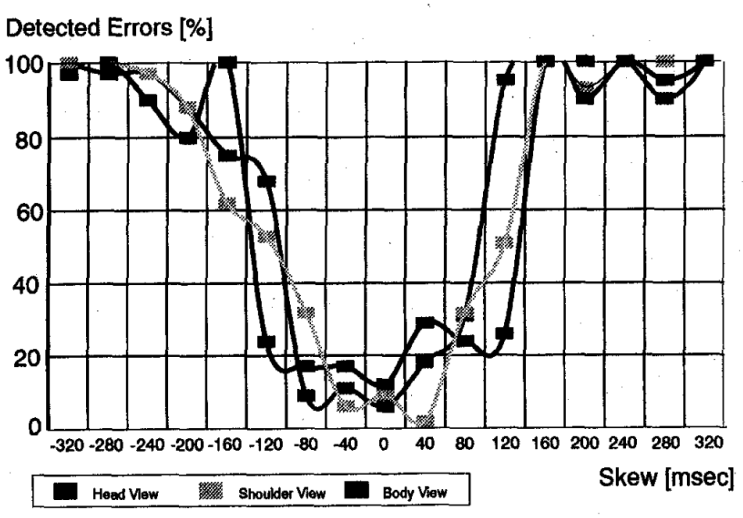

Figure 2.5: The figure shows how the skew between audio and video was detected by the subjects in the experiment. The area between -80ms and 80ms was concluded to be the in-sync area. The are beyond -160ms and 160ms was concluded to be out-of-sync and the are between in-sync and out-of-sync was called the transient area.

It is chosen to use the requirement of no more than -80ms to 80ms skew between the video and audio streams as the performance requirement for this project.

Fill Algorithm	Sex of Talker	Detect or Object	Loss Rate (%)
Silence	Male	Detect	1.51
Silence	Male	Object	0.67
Silence	Female	Detect	1.80
Silence	Female	Object	0.90
Noise	Male	Detect	3.58
Noise	Male	Object	1.87
Noise	Female	Detect	2.84
Noise	Female	Object	0.76
Repeat	Male	Detect	1.63
Repeat	Male	Object	0.33
Repeat	Female	Detect	3.43
Repeat	Female	Object	0.76

Table 2.1: The table shows the packet loss rates for detection and objection by the listener. The results er presented for each of the three different filling algorithms.

2.3 Packet Loss Requirement

Another parameter that needs consideration is the amount of packet loss a wireless link will experience during the transfer of the audio stream. A requirement regarding packet loss must be conducted to give a view of the quality of the transmission. In the ideal world the packet loss would be zero, but the wireless medium will always experience erroneous frames and packets. The requirement must thereby define the limit of which packet loss can be tolerated before the listener of the stream becomes unsatisfied.

An subjective analysis and evaluation of the listeners experience of voice in a telephone handset, when different fill algorithms are used and the transport stream are exposed to different means of packets loss were done by British Telecom Research Labs[15]. The analysis are done using a test processed with trained listeners. All listeners are told to mark their experience as high quality, detectable quality degradation or non acceptable quality degradation. The results of the experiment are presented in table 2.1.

The results from this experiment is used as an indication of the tolerated packet loss in an audio stream. It has been decided to use the silence fill algorithm as reference, since this seems to be the simple to implement and is widely used. This raises a requirement that the packet loss must not exceed 1.51 % for male talkers and 1.80 % for female talkers. For the rest of this report the requirement used is maximum 1.80 % uniformly distributed packet loss.

2.4 Initial Experiment

It is chosen to conduct a set of initial experiments in order to get an understanding of the issues which arises when trying to stream the TV audio to a number of headsets. An initial approach for streaming the audio from a TV could be to just record the audio at the audio output of the TV, and stream this to the clients.

This is illustrated in Figure 2.6, where a computing device is connected to a TV via a analog audio cable feeding the audio into the audio input device. Here the analog audio is sampled to digital audio and streamed over Wireless Local Area Network (WLAN) to a number of headsets. This setup can be emulated using two computers with one of them acting as a stream server streaming audio from a stored audio file, and the other acting as a streaming client (headset) receiving and playing the audio. Doing this would remove the delay from the TV audio output to the computing device audio input, but this delay is considered to be insignificant. This leaves the network end to end delay as the expected pitfall, together with a possible loss of packets due to the unreliability of the wireless medium. An experimental setup is created to try out these ideas.

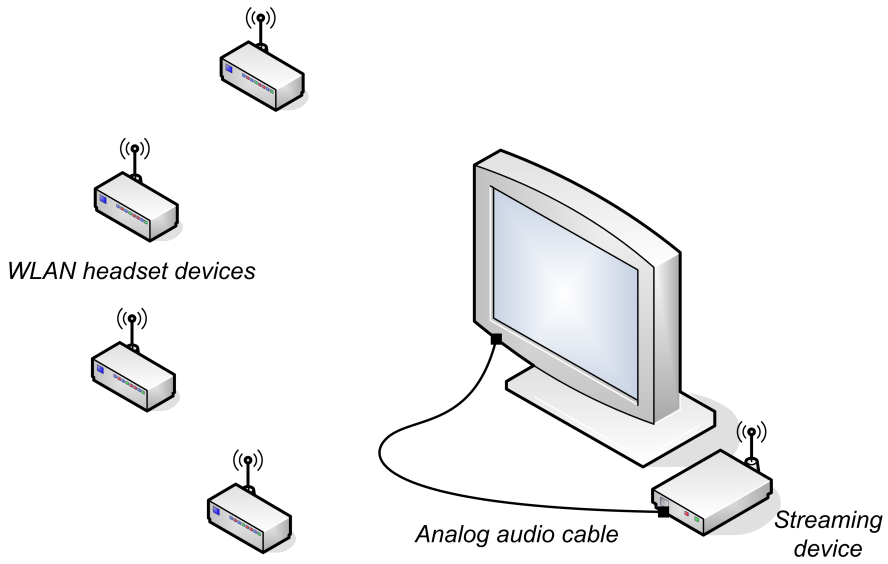

Figure 2.6: The figure shows how the streaming of audio could be done. Attaching a WLAN and audio recording capable computing device to the TV and record the audio from the TV audio output port. The audio is then streamed to a number of clients using WLAN.

The overview of the experimental setup is shown in Figure 2.7. In each scenario a MP3 file containing approximately 4 minutes of music is played on the stream server. The stream server is connected to a wired Ethernet network, which includes the Access Point (AP) in infrastructure mode. The audio track is streamed onto the network using PulseAudio[1], which streams the raw 16 bit PCM audio using Real-Time Protocol (RTP) to the receiving node.

[1] http://www.pulseaudio.org/

The experiment consists of five scenarios on respectively a wired LAN and Wireless LAN. The scenarios are named as follows:

- **lan:** is the scenario where the stream server and client are directly connected trough the wired Ethernet switch
- **no-disturbance:** is the scenario where the audio is streamed through the wireless network without disturbance
- **2mbit:** another run of the wireless scenario, this time with 2mbit UDP disturbance traffic on the AP
- **10mbit:** another run of the wireless scenario, this time with 10mbit UDP disturbance traffic on the AP
- **20mbit:** final run of the wireless scenario with 20mbit UDP disturbance traffic on the AP

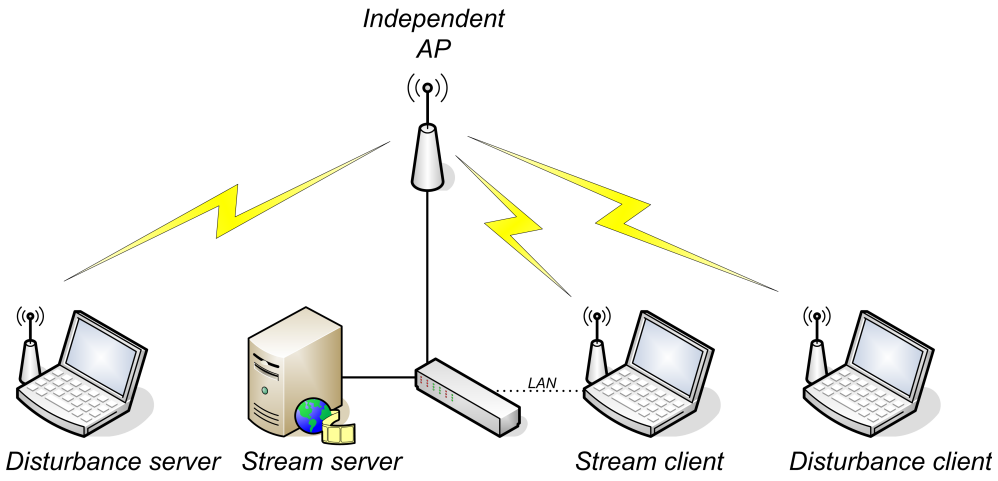

Figure 2.7: The figure shows the network and setup used for the initial experiments. The setup consists of three wireless laptops and a stream server. Two of the laptops performs disturbance to the wireless network, while the stream server and client handles the actual audio streaming of the scenario.

The RTP stream will in all scenarios, excluding the LAN reference scenario, traverse the wireless 802.11g link between the nodes. In parallel, two other wireless nodes are connected to the infrastructure AP. These two nodes are used to add a controlled amount disturbance traffic to the WLAN, in order to yield results that shows the influence from having their traffic in.

At first the stream client is connected to the Local Area Network (LAN) network by use of wired Ethernet. This is done to create a reference scenario for the wireless tests, since this scenario is considered as ideal. The scenario is repeated 5 times to increase significance of the results. When this scenario is finished, the Ethernet cable is disconnected and all following tests will use WLAN technology.

	Network Delay (mean) [ms]	Samples [.]	Confidence [ms]	Standard Deviation [ms]
lan	0,12	300	+/- 0,00	0,01
no-disturbance	2,35	300	+/- 0,20	1,62
2mbit	2,90	300	+/- 0,32	2,57
10mbit	20,27	270	+/- 2,02	15,55
20mbit	37,33	174	+/- 1,91	11,80

Table 2.2: The table shows the measured network end-to-end delay. The measurements are performed as a RTT Ping test, while the scenario is running. The table show the half RTT times, in order to give a meassure of the one way end-to-end delay.

The WLAN tests are performed with different levels of disturbance. The two disturbance nodes uses IPerf[2] as a traffic generator, to create an UDP stream between the server and the client. UDP is chosen to ensure that the rate of the disturbance is fixed, and unaware of e.g. sliding window mechanisms that controls the bandwidth. The IPerf server loads the network with 2, 10 or 20 Mbit per second of random data. This disturbance will likely affect the contention, collisions and processing times in the network. If this is true, it will lead to measurable degraded quality on the PCM audio stream. Each of these tests are, as with LAN, performed 5 times to gain higher confidence in the results.

2.4.1 Results

The results of the experiment are obtained by use of a Wireshark PCAP capturefile on each of the participating nodes, in combination with a simple ICMP ping to measure the network transmission delay. A set of five different scenarios are defined, each of them with different means of disturbance.

At first the mean network delay is considered, where the results are presented in table 2.2 and on figure 2.8 on the following page. Each result is obtained by sending 100 Ping packets from the stream client to the stream server during the scenario. It can be seen from the results in figure 2.8 on the next page that the network mean delay increases with the amount of disturbance traffic loading the wireless network, but it stays well below the 80 ms requirement. In the special case running on wired Ethenet LAN the network end-to-end delay is very near zero.

The results shown on figure 2.8 on the following page, represents a RTP analysis of the inbound stream of the receiving node. The RTP analysis is performed using the Wireshark Network Analyzer tool[3]. The following descriptions are used in the figure:

[2] http://sourceforge.net/projects/iperf
[3] http://www.wireshark.org/

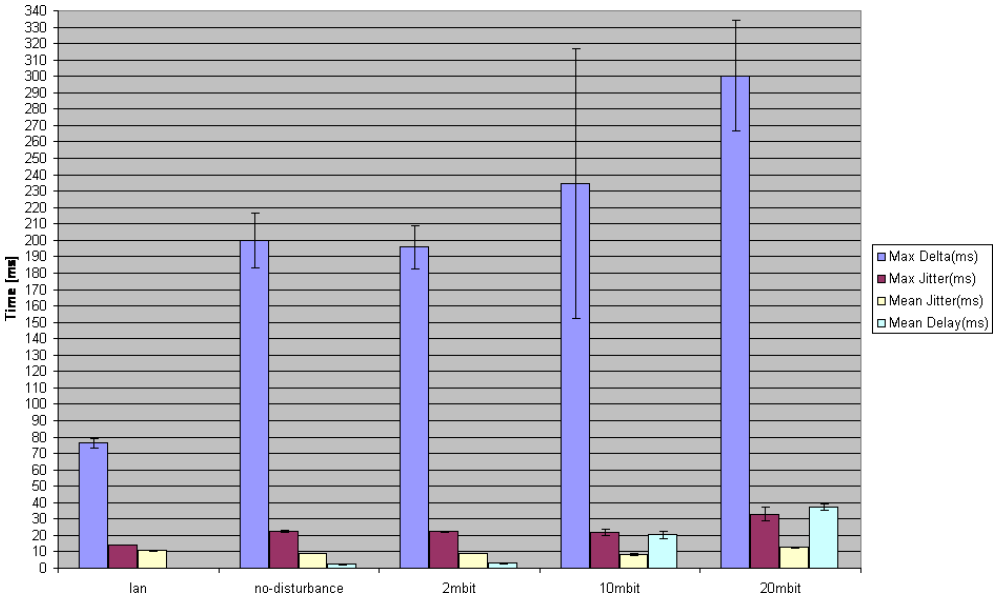

Figure 2.8: The figures shows the results of a Wireshark RTP analysis of the inbound stream on the receiving laptop node. For each of the five scenarios three different measurements are derived regarding the timing: Max Delta, Max Jitter and Mean Jitter. The RTP analysis results are plotted together with the results for mean network delay from Table 2.2. The errorbars indicates the 95% confidence interval of the dataset.

- Max Delta, is the maximum time between two successfuly recieved packets in the RTP stream
- Max Jitter, is calculated according to RFC3550 which describes the RTP protocol[4]
- Mean Jitter, is the mean of the jitter in the stream
- Mean Delay, is the mean of the delay extracted from the half of the Round-Trip-Time in the network

From the maximum inter arrival time (max delta) between the RTP packets is it possible to see the unreliability of the wireless medium. Comparing the LAN scenario, with the no-disturbance scenario it is clear that the inter arrival time increases, together with a small increase in jitter. The results obtained from the 2mbit disturbance scenario shows that a small amount of traffic on the network, together with the stream, do not affect the quality of the stream remarkable. But if this is compared to the LAN scenario, there is a considerable difference.

There seems to be a tendency in the results for 10mbit and 20mbits cases, which indicates that the Max Delta inter arrival time increases with higher loads, even though it cannot be concluded with this level of significance.

Figure 2.9 on the next page shows the packet count of transmitted and lost packets respectively, in each of the five test scenarios. It is seen that the LAN scenario gives a near ideal transmission, without packet

[4]Basically, the max jitter is a smoothed derivative of the inter arrival delta.

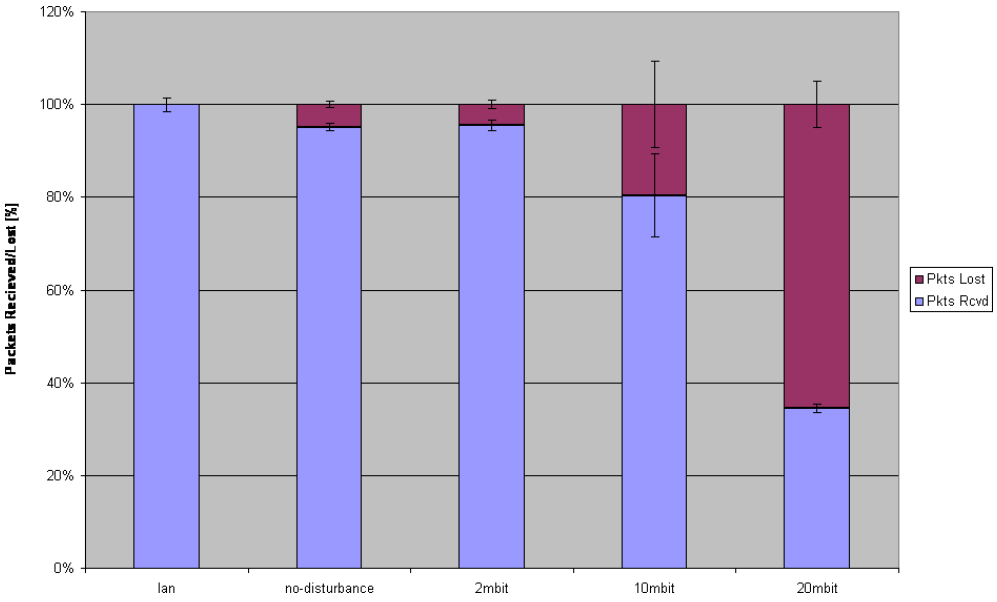

Figure 2.9: The figure shows the amount of transmitted and lost packets in each of the five test scenarios with different amounts of interference. The LAN scenario is the only one without any packet loss.

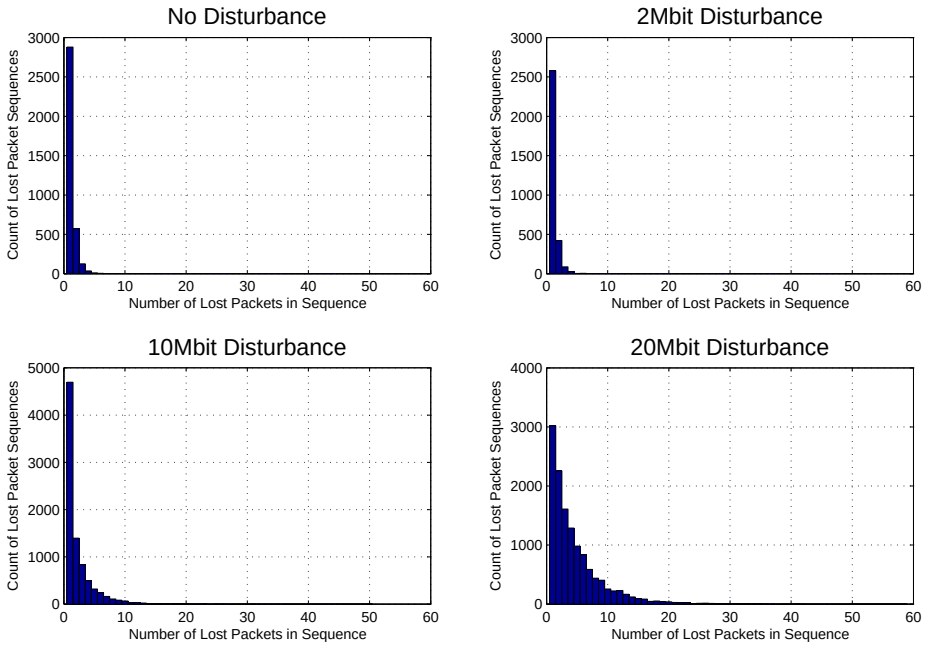

Figure 2.10: The figure shows a histogram of the lost packet sequences for each of the four wireless scenarios. The LAN scenario is excluded since there was no packet loss pressent.

loss and a high level of significance. Moving the stream to the wireless medium, increases the RTP packet loss from 0 pct. to 4,8 pct. at average. Comparing the no-disturbance case with the results of 2mbit disturbance traffic, it is seen that the difference between them is almost not existing. Actually, the packet loss for the 2mbit case is lower. Raising the disturbance to 10mbit and 20 mbit increases the packet loss essentially, even though the significance level for 10mbit is low.

2.4.2 Discussion

It is noticeable that the maximum inter arrival time (max delta) is very high compared to the actual network delay in each scenario. Even in the LAN case where the delay is very low, the inter arrival time is near the 80 ms limit. This is assumed to be coursed by checksum offloading in the wired Ethernet card. The Max Delta inter arrival time is in the RTP analyzer calculated as the difference in time between two successful packet arrivals. The Max Delta inter arrival time is therefore increasing with the packet loss shown in figure 2.9 on the preceding page.

The confidence intervals of the RTP analyzer results are in all cases quite good, except for the 10 mbit case. It is from the dataset concluded that this must be due to the heavy packet loss in one of the test runs.

It it surprising that even with a clear wireless channel without any clients contenting, a packet loss of 5 pct. is achieved. This is of course due to the unreliability of the wireless medium, but a much lower value was expected in this ideal setup.

2.4.3 Conclusions

The results of this experiment has shown that the transmission delay between the nodes is not really a problem. The largest network delay is seen for the 20mbit disturbance case, but still with a value of 37ms it is not near the 80ms limit. On the other hand, the inter arrival time (max delta) of the individual RTP packets introduces a problem.

This is coursed by the heavy packet loss, when loading the network with disturbance traffic. To solve those packet loss problems a more reliable transmission is need. This will consequently introduce a delay due to buffering, and add more complexity to the problem solution. These alternatives will be discussed further in the following section.

2.5 Packet Loss Recovery Techniques

Main source: [3]

Based on the conclusion on the initial experiment, the main Quality of Service (QoS) problem when trying to stream the audio from the TV directly is that of packet loss. This section will give an overview of possible solutions to this problem, detailing pros and cons for each solution.

In general packet loss recovery techniques can be split into two overall groups. There are the sender driven techniques and there are the receiver based techniques which should both be used to gain the best possible performance.

2.5.1 Sender driven

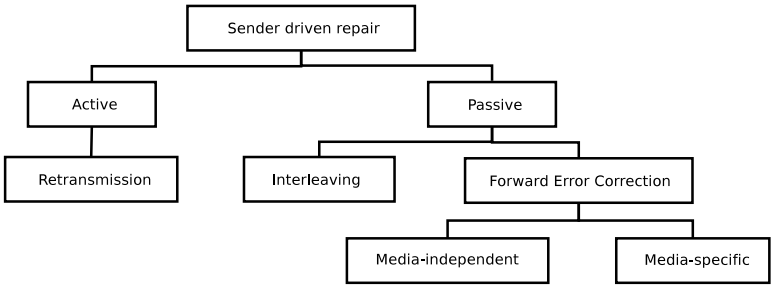

Figure 2.11: The Figure shows a taxonomy of the different sender based packet loss recovery techniques. Source:[3]

Figure 2.11 shows a taxonomy of the sender driven techniques for packet loss recovery. There are two major classes being Active Retransmission (AR) and Passive Channel Coding (PCC), where retransmission is the only active one. The passive ones are interleaving and Forward Error Correction (FEC) where forward error correction can be either media-independent or media-specific.

Forward Error Correction

Forward Error Correction (FEC) works by adding extra information to the stream in order to be able to recover lost packets. This can be done using a number of different techniques which can be either media-independent, meaning it acts below the application level, or media-specific which acts at the application level.

Media-independent FEC takes a codeword of k data packets and generates $n - k$ additional check packets for the transmission of n data packets over the network. Examples of media-independent FEC block code schemes are parity coding[12] and Reed-Solomon[6][11] coding. Reed-Solomon codes are renowned for their excellent error correcting properties and in particular their resilience against burst losses. The advantages of the media-independent FEC is that is does not depend on the contents of the packet and the repair is an exact replacement for a lost packet and that the computation required to derive the error correction packets is relatively small. The disadvantages are additional delay, increased bandwidth usage and difficult decoder implementations.

Media specific FEC in the simplest form works by transmitting each unit of the application stream data (e.g. audio) in multiple packets. If one packet is lost another packet containing the same unit will be able to cover the loss. The first transmitted copy of the stream data is called the primary encoding and the

subsequent transmissions are called the secondary encodings. The sender can choose it the secondary encoding scheme should be another than the primary one, usually a lower quality and thereby lower bandwidth encoding is used, which depends on the bandwidth requirements and encoding complexity trade-off. The advantage of media-specific FEC compared to the media-independent is that the transmission overhead can be reduced without affecting the number of losses which can be repaired. It also has the advantage that it only adds a single packet delay making it ideal where large end-to-end delays cannot be tolerated. If larger end-to-end delays can be tolerated it is possible to delay the redundant copy of the packet making it more robust to burst losses.

One problem of using FEC to protect against packet losses is if the reason for the packet loss is congestion in the network. Adding more data to the streams in the network will worsen the problem instead of preventing the packet loss, which again will add more FEC data to the streams etc.

Interleaving

When the stream data unit size is smaller than the network packet size interleaving can be used to reduce the effect of packet loss. The stream data units are re-sequenced before the transmission on the network, such that the originally adjacent units are separated with a guaranteed time distance. This means that if a packet is lost, there will only be small holes in the reconstructed stream instead of a larger hole because several adjacent data units are lost. The technique is illustrated in Figure 2.12.

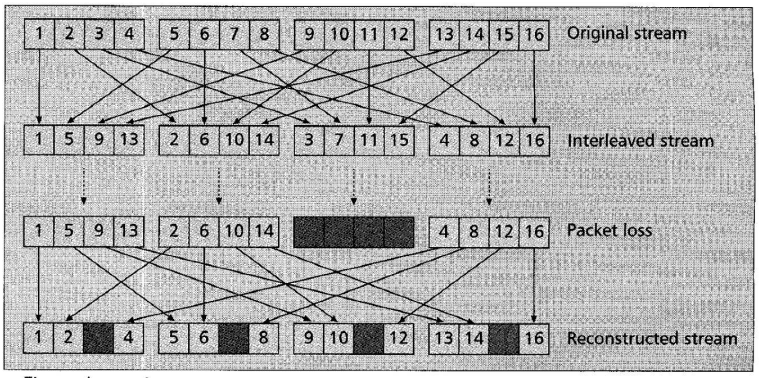

Figure 2.12: The Figure shows how interleaving works. The data units of the original stream is shuffled to the interleaved stream. If a packet is lost, only small data units will be lost in the reconstructed stream. Source:[3]

The smaller gaps in the streamed data means that the loss is spread such that only small parts of e.g. the phonemes in human speech is lost instead of loosing a whole phoneme. This makes it easier for the human listener to mentally "patch over" this loss[7], resulting in improved perceived quality. The majority of speech and audio coding schemes can have their output interleaved and may be modified to improve the effectiveness of interleaving. The advantage of interleaving is that it does not increase the bandwidth requirements while the disadvantage is that it increases latency.

Retransmission

In the experiment conducted in Section 2.4 on page 13 User Datagram Protocol (UDP) is used as transport protocol for the network communication. This is an unreliable protocol meaning packets can be dropped at lower layer resulting in missing data at the application layer. Changing the transport protocol to a reliable protocol can solve the problem of lost packets and there are a number of different protocols to choose from. Most reliable protocols works as a unicast protocol meaning that it has a stream of data from the server to each client in the network, but reliable multicast and broadcast protocols also exists.

Transport Control Protocol The most widely used reliable transport protocol is Transmission Control Protocol (TCP) which is used as transport protocol on the Internet. Using TCP at the transport level of the protocol stack gives a much lower probability for loss by using a 16bit field in the TCP header for calculating a checksum and by sending acknowledgement packets from the receiver to the transmitter informing about successfully transmitted packets. Using TCP as transport protocol between the stream server and the stream clients would remove the problem of lost packets, but at a certain cost. As TCP is a connection oriented protocol, a unicast stream of data will be created from the stream server to every stream client, which would add to the bandwidth usage for each stream client. Furthermore the acknowledgements sent for each TCP will take up bandwidth as well, causing higher delays as contention for the channel will increase at the data-link layer meaning that TCP scales poorly to larger streaming networks compared to UDP. To be able to actually resend lost packets, buffering of packets has to be enabled at both the transmitting and receiving side of the communication which will add additional delay to the traffic dependent on how large the buffers are chosen to be. As TCP is a more complex protocol than UDP it will also add to the network processing time needed at each node especially at the server since it has to handle a connection for each stream client.

Using TCP over a wireless link where packet loss is present has been shown to be a problem. TCP uses Acknowledgement (ACK) packets to decide when packets are lost, and many TCP implementations assumes that a packet loss is due to congestion somewhere in the path of the connection. This assumption is made since the Internet consisted of wired links only when TCP was invented. When packets are lost due to a unreliable wireless link, this means many TCP implementation incorrectly will try to avoid congestion by slowing down, instead of re-transmitting the packets instantly. Many different proposals have been made to solve this issue ranging from changing TCP to hiding the packet loss by re-transmitting lost packets in lower layers.

The required bandwidth for the audio stream can be made smaller by the usage of audio codecs to compress the audio. The experiment in Section 2.4 on page 13 streams the audio in the "raw" PCM format. Instead the audio could be encoded e.g. with the popular MP3 format which would require less bandwidth for a single stream. This would make it possible to have more nodes using unicast TCP streams.

Reliable Multicast In order to save bandwidth compared to using unicast streams it is often possible to use multicast instead. This is also used in the streaming experiment in Section 2.4 on page 13, but since the 802.11 channel is a shared medium the traffic sent to a multicast group will reach all nodes in the network anyway.

Reliable Broadcast It is also possible to use reliable broadcasting, where acknowledgement packets are sent from the receiver to the transmitter. The initial problem with such solution is the significant overhead which is added when all packets to all nodes are acknowledged. The HIDENETS project[5] proposed an algorithm combining different enhancement techniques for lowering the overhead[10, p.203] which through simulations was found to significantly reduce the acknowledgement as well as retransmission overhead in multi-hop ad-hoc wireless networks. As with any other reliable protocol this will add to the delay of the streamed audio since packets lost on lower layers needs to be retransmitted.

Retransmission summary Retransmission is typically not used for latency sensitive applications like streaming audio, since the retransmission of a packet will add considerably to the end-to-end delay, but depending on the specific requirements it can be a possibility. The main disadvantage of most reliable protocols e.g. TCP is that they do not bound the amount of retransmissions leading to an unbounded delay of the transmission of packets in the presence of packet loss. It is however possible to define retransmission schemes which bound the number of retransmissions, but these works best when the loss rates are relatively small. As the loss rates increase the overhead due to retransmission increases leading to a cross-over point where the use of FEC becomes more effective. Retransmission could also be used as a supplemental technique for recovering losses which cannot be repaired by FEC, or in a combination with FEC where a retransmission consist of a FEC packet which can repair multiple losses[8].

2.5.2 Receiver based

If the sender-based techniques cannot repair all losses or if the sender of a stream is unable to participate in the recovery, there exist a number of error concealment techniques which can be implemented by the receiver of the stream. An overview of the different receiver based techniques is given in Figure 2.13.

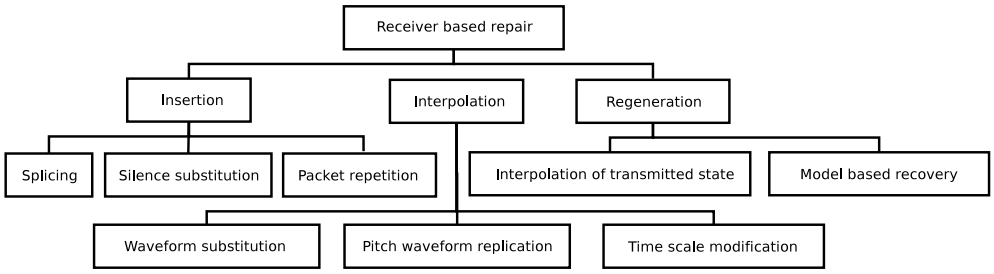

Figure 2.13: The figure shows an overview of the receiver based techniques for packet loss recovery. Source:[3]

[5] http://www.hidenets.aau.dk/

The three classes of receiver based error concealment techniques are insertion, interpolation and regeneration. The error concealment schemes rely on producing a replacement data unit to replace the lost packet, which can be done since audio signals exhibit large amounts of short-term self-similarity. These techniques works well for relatively small loss rates $(<\approx 15\%)$ and small packets $(4 - 40msaudio)$. When the loss length approaches the length of the phoneme the techniques break down since whole phonemes can be missed.

Insertion

The insertion based techniques works by deriving a replacement for a lost packet by inserting a simple fill-in. These techniques does not use the signal characteristics to aid the reconstruction makes them simple to implement but which also make them perform poorly.

Splicing In splicing the lost data units in the stream are simply disregarded and the stream is instead spliced together from both sides of the missing data. This means there are no longer a gap in the stream but the timing of the stream is disrupted. The performance of this technique is intolerable with loss rates above 3%[4]. A disadvantage to this technique is that it can interfere with the playout buffer, which is used to allow re-ordering of packets, removal of network timing jitter and retransmission, by step reducing the amount of data available in the buffer.

Silence substitution With silence substitution the lost data is replaced with silence in order to maintain the timing relationship in the stream. It is effective with short packets $(< 4ms)$ and low loss rates $(< 2\%)$[5] with performance degrading rapidly as packet sizes increase. The advantage of the solutions is the simple implementation which also means it is in widespread use of applications.

Noise substituion Instead of inserting silence into the stream, it is possible to insert noise/random stream data into the stream. It has been shown that the human brain is capable of subconsciously repair segments with random noise in speech data[9] while not being able to do it with silence substitution. The use of white noise has been shown to give both subjectively better quality[7] and improved intelligibility[9].

Repetition Another way of doing insertion based repair is to replace the lost units with a repetition of the last received unit. This has a low implementation complexity and performs reasonably well. The subjective quality of repetition can be improved by gradually fading the repeated units which is used by the Global System for Mobile communications (GSM) system.

Interpolation

Interpolation techniques attempt to interpolate from the packets surrounding a loss in order to produce a replacement. The advantage of these techniques compared to the insertion based techniques is that they account for the changing characteristics of the signal.

Waveform Substitution By using the audio before and optionally after the loss, templates can be used to locate suitable pitch patterns. This is used to generate a substitution signal to place instead of the lost packets. Two-sided schemes works better than one-sided schemes in terms of quality and both works better than silence substitution and repetition.

Pitch Waveform Replication A refinement on waveform substitution is pitch waveform replication which utilizes a pitch waveform detection algorithm on both sides of the loss. Losses during unvoiced speech segments are repaired using packet repetition while voiced losses are repaired using a waveform of appropriate pitch length. The technique works marginally better than waveform substitution.

Time Scale Modification It is also possible to stretch the audio on both sides of a loss such that it fills the gap. The technique is computationally demanding but appears to work better than both waveform substitution and pitch waveform replication.

Regeneration

It is also possible to use knowledge of the audio compression algorithm to derive codec parameters such that the audio in lost packets can be synthesized. These techniques perform well due to the large amount of state information used for the repair. A disadvantage to these techniques is that they are typically computationally intensive.

Interpolation of Transmitted State For codecs based on transform coding or linear prediction, the decoder can possibly interpolate between states. The advantage of codecs using this technique compared to recoding on both sides of the loss is that there are no boundary effects due to changing codecs and the computational load remains almost constant. The codecs where interpolation may be applied does however typically require more computational power.

Model-Based Recovery In model-based recovery a model is fitted to the speech on one or both sides of the loss, which is then used to generate speech to cover the period loss.

2.6 Conclusion

In the preliminary analysis a streaming experiment was conducted to get initial experience with the problems associated with streaming audio over a wireless link. The results from the experiment indicates that the main problem to be solved is that of packet loss due to the unreliable nature of the wireless link, while the delay requirement is satisfied.

A number of techniques for solving the packet loss problem was described, both sender driven and receiver based techniques. Each technique has advantages and disadvantages, which is listed in Table 2.6. The techniques does not need to be used independently, and is actually best used in combinations[3].

Sender driven		
Technique	**Advantage(s)**	**Disadvantage(s)**
FEC	low delay transparency	increased bandwidth congestion complexity
Retransmission	transparency congestion control	latency
Interleaving	bandwidth	latency
Receiver based		
Technique	**Advantage(s)**	**Disadvantage(s)**
Insertion	simple implementation	poor performance (quality)
Interpolation	performance (quality)	implementation complexity
Regeneration	good performance (quality)	computational complexity implementation complexity

Table 2.3: The table shows a comparison of advantages and disadvantages using different techniques for solving the packet loss problem.

One technique which is evaluated to work poorly in multicast scenarios on the Internet is that of retransmission[3]. The reason it performs poorly is that for delay sensitive applications, transmitting data over the Internet involves a relatively high end-to-end delay, since the data travels through a number of links and routers. This large end-to-end delay is not present in this project, since there is only one link the data should travel. This makes retransmission a possibly usable technique to use for this project why it is chosen to delimit the rest of the project to retransmission.

3 Problem Statement

The preliminary analysis concluded that one possible technique for solving the packet loss problem could be retransmission. It has therefore been chosen to work further with this technique in order to try to find an optimal retransmission protocol to use for the project use case.

3.1 Problem Statement

How is it possible to reduce the performance degradation caused by packet loss, in audio streaming over a wireless network using retransmission, within the 80 ms time bound of synchronization?

The objective of the rest of the project will be to evaluate different retransmission protocols in terms of their performance when used in the project use-case. This means evaluating each protocol according to the following metrics in hierarchical order:

1. Delay
   - The specified delay requirement of maximum 80ms skew must be fulfilled.
2. Robustness
   - The retransmission protocol has to be able to fulfil the specified packet loss requirement of maximum 1.80% [15]. However, if other reciever based techniques are used in combination with retransmission the stream could possible survive a higher packet loss.
3. Bandwidth scalability
   - Since the use-case specifies that a number of clients needs to be able to receive the streamed audio, the retransmission protocol must be able to scale in terms of bandwidth. Using the most scalable protocol will also make the solution most cost effective since a maximum number of clients can be served from one device.

In order to be able to evaluate the performance of different retransmission protocols a set of measurements will be done using a varying set of parameters. These measurements will make it possible to conclude whether an existing retransmission protocol can be used to fulfil the requirements. If this is not the case, the conclusions should give the tools for constructing a new protocol that could possible increase the performance using the existing wireless network.

Since it is beyond the scope of this project to work with audio compression and recording techniques a delimitation needed.

3.2 Delimitation

It is chosen to use an of the shelf audio stream codec for reference, and use the properties of this codec as input to the model of the system. The codec used from this point is PCM, since this equals the one used in the initial experiment and do not use compression. This will give a pure view of worst situation achievable, and keeps the focus on the retransmission problem.

The use-case poses the possibility to watch individual channels by use of split screen technologies, therefore it is chosen to keep the focus on unicast streams for the rest of the project. The project is thereby delimited from multicast and broadcast scenarios, which will be considered as future work in the view of this project.

4 Requirement Analysis

The requirements to the audio stream system which has been found through the preliminary analysis should be used to evaluate the performance of a set of retransmission techniques. One of these requirements are specifying the maximum skew which can be tolerated between the TV video and the TV audio.

This chapter proposes a new method of evaluating whether the skew requirement is fulfilled. The new method is needed since the traditional method of measuring the skew would be to measure the time difference directly on the network packets from transmission to reception. In order to measure the time difference, the clocks on the transmitting and receiving device need to be synchronized. Using standard equipment it is difficult to guarantee <ms accuracy, and trying to achieve it by using e.g. a tool like ntpdate[1] could result in wrong results since it causes a high system load when it is pushed to the accuracy limit.

4.1 Audio Stream Properties

This section will derive a set of properties of the audio stream used in the initial experiment which can be used to propose a method of measuring the skew given these specific properties.

The codec used in the initial experiment in Section 2.4 on page 13 is 16bit stereo Pulse Code Modulation (PCM) which means an application bitrate of:

$$44100\frac{samples}{s} \cdot 2\ channels \cdot\ 16bit = 1411200\frac{bit}{s} \tag{4.1}$$

The packet size used in the initial experiment is 1300Bytes. The number of packets per second generated from the stream is then:

$$\frac{1411200\frac{bit}{s}}{1300\ Byte \cdot 8\ bit} \approx 135.7\frac{packets}{s} \tag{4.2}$$

The length of the audio stream in each packet is:

$$\frac{1s}{135.7packets} \approx 0.0074\frac{s}{packet} = 7.4\frac{ms}{packet} \tag{4.3}$$

Using this information it is possible to model the audio streaming system as a queue system as illustrated

[1] http://linux.die.net/man/1/ntpdate

in Figure 4.1. There is a recording process which generates audio with a constant bitrate of 1411200 bit/s. The service rate of the recorder process is the arrival rate of the TCP/IP stack at the transmitter. The service rate of the TCP/IP stack is not known by exact value but can be assumed to be approximate constant and much higher than the arrival rate. The service rate of the TCP/IP stack becomes the arrival rate of the 802.11 MAC layer which has an unknown and variable service rate. This service rate becomes the arrival rate of the TCP/IP stack on the receiver side which again can be assumed to have an approximate constant service rate. At last the service rate from TCP/IP becomes the arrival rate to the audio stream player.

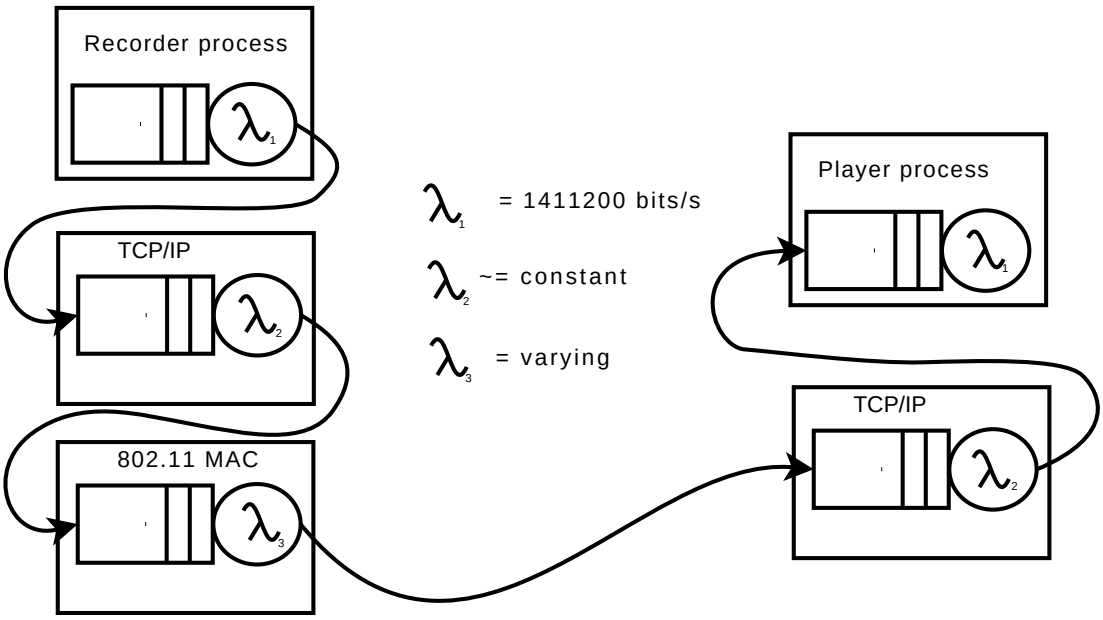

Figure 4.1: The figure illustrates a simplified queue model of the audio streaming use case.

The service rate which can become the bottleneck in this audio stream scenario is that of the 802.11 MAC layer. If the disturbance of the 802.11 link gets to high (noise, other traffic etc.) packets are dropped and delayed. These packet drops and delays can eventually make it impossible to fulfill the requirements.

4.2 Skew Requirement Checking Method

In this section a method to measure if the skew requirement is fulfilled is proposed. Instead of measuring the skew directly by recording the transmission time and the reception time for each packet, the skew

can be evaluated indirectly by controlling if the requirement is fulfilled and measuring how many times it is not fulfilled. To be able to explain the method some examples of how the audio stream would be played under different circumstances are presented.

If the audio stream player starts playing the audio as soon as the first packet arrives, the skew will be minimal, but if the delay of the transmission varies too much it is not possible to retain the required arrival rate. This is illustrated in Figure 4.2. If a packet arrives too late to meet the service deadline it will mean information was lost in the audio stream.

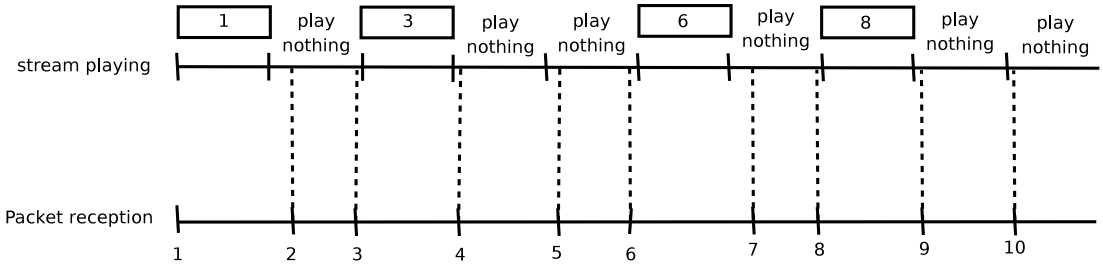

Figure 4.2: The figure shows an example of how the audio stream would look if the player starts playing the audio stream as soon as the first packet arrives and the network delay varies too much.

If the application instead uses a packet buffer to queue the packets in before the stream client starts playing, the network jitter can be averaged away. If two packets are buffered before the playing is started the stream from Figure 4.2 will instead look as shown in Figure 4.3

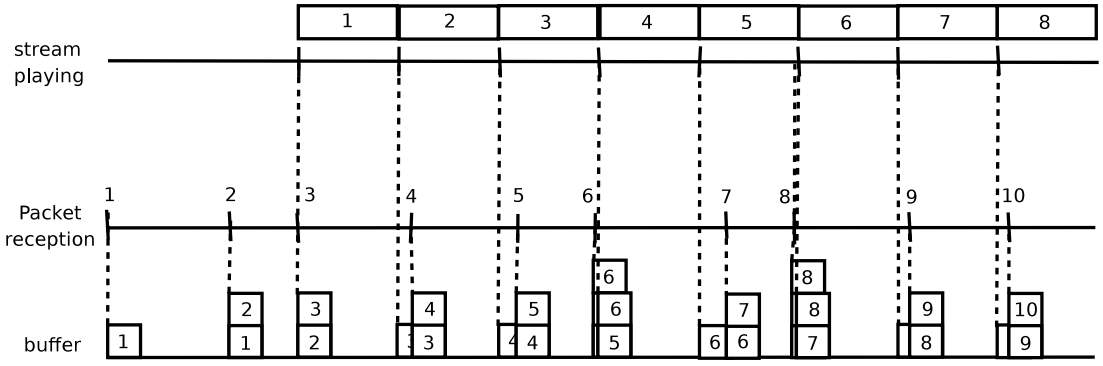

Figure 4.3: The figure shows an example of how the stream would be played with a buffer to smoothen the network jitter. The stream is first started when there are two packets in the buffer which creates an initial delay.

Even with the buffer in the application, data exhaustion can still happen. This is illustrated in Figure 4.4 where the average delay of the packets gets so large that eventually the buffer will be empty when a packet is needed to play the stream.

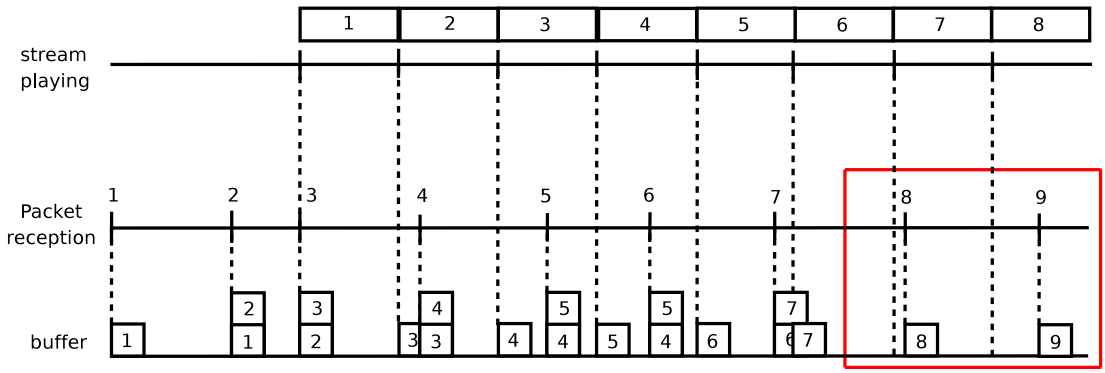

Figure 4.4: The Figure shows an example of what happens when the packets are generally delayed in the network. At some point there will not be a packet available in the buffer.

The example of buffer exhaustion in Figure 4.4 could be partially prevented, if the network jitter does not continue to be too large, by having a larger amount of packets in the buffer before starting to play the stream. The amount of packets that can be buffered is related to the skew requirement of 80ms, since the playback can not be delayed more than 80ms. This means that the length of the audio data buffered before the playback is started can not exceed 80ms.

The 80ms maximum skew includes the initial transmission time to generate and receive the first packet, which means this delay have to be subtracted from the 80ms in order to get the right buffer size. In order to get an idea of how this could be estimated Figure 4.5 illustrates the recording and transmission of one packet.

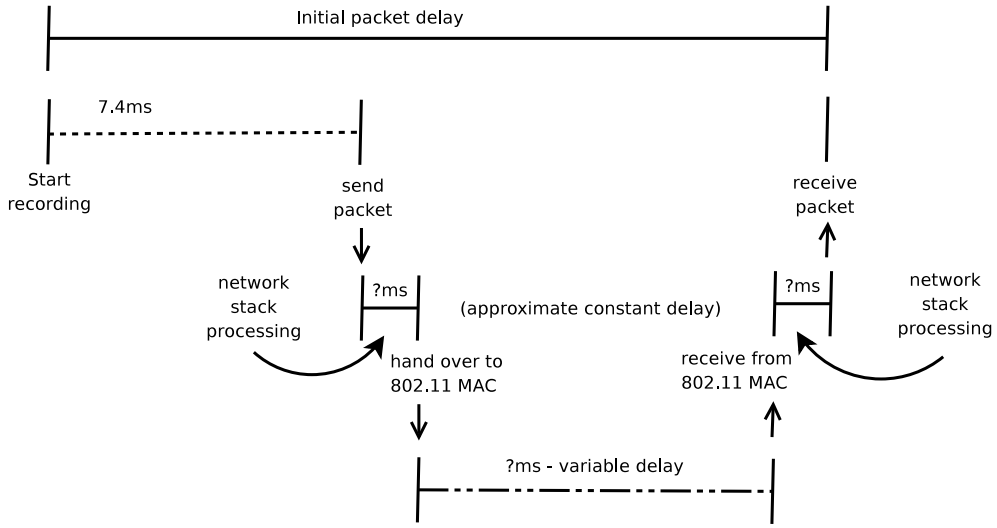

Figure 4.5: The figure shows an overview of the delay imposed on one audio stream packet. First the recording is started, and after 7.4ms enough audio data is available to generate one 1300Byte packet. The packet is processed by the transmitters network stack adding an approximate constant delay and handed over to the 802.11 MAC layer which will add an unknown variable delay to the packet. When the packet is recieved the receivers network stack processing will add an approximate constant delay after which the application receives the packet.

First the recording is started, and after 7.4ms enough audio data is available to generate one 1300Byte packet. The packet is processed by the transmitters network stack adding an approximate constant delay and handed over to the 802.11 MAC layer which will add an unknown variable delay to the packet. When the packet is recieved the receivers network stack processing will add an approximate constant delay after which the application receives the packet. From here on it is up to the receiving application to chose the strategy after receiving the first audio stream packet.

If the application chooses to start the playback of the audio stream immidiately after receiving the first packet, only the initial delay (see Figure 4.5 will be added to the audio stream, but it also means that the next audio stream packet has to be available exactly after 7.4ms when there is no more audio to play from the first packet. If the network link from the transmitter to the receiver was ideal it would only add a constant delay to each packet, meaning the packets could arrive to meet the required deadline. With WLAN this is not the case, and the application will have to handle a varying network delay.

If the wireless link is not saturated with traffic and estimate of the experienced delay is found in Section 2.4 on page 13 to be approximately 2ms. Adding this to the 7.4ms to create the packet and some processing time in the network stack approximately 10ms of initial delay is chosen. This means approximately 70ms of audio data can be buffered before the playback is started. The number of packets which can be buffered then becomes:

$$\frac{70ms}{7.4ms}\lfloor 9packets \rfloor \tag{4.4}$$

This means that by requiring 9 packets, $9 \cdot 1300$Bytes or $9 \cdot 7.4$ms of audio have to be received before the audio stream starts playing, and within no more than 70ms in total, this amount of data can be used to determine when the skew requirement is not fulfilled. By artificially introducing an application buffer to keep these packets in and evaluating if/when/how often there are no packets left in this buffer, it will be known if/when/how often the skew requirement is not fulfilled. This calculation will be used to evaluate the performance of the different retransmission protocols, and will from here on be named "buffer calculation".

A choice has to be made regarding what happens if the buffer gets empty in order to evaluate the different protocols from the same set of criteria. It is chosen that if/when this happens, the stream will have to be restarted.

If it is possible for the audio streaming application to detect when there has been a packet loss, it becomes a possibility to use recovery methods. This will influence how the buffer calculation should be performed and is therefore the subject of the next section.

4.3 Buffer Calculation Special Case

If the retransmission mechanism have to give up it can happen that a packet is lost. If there are no sequence numbers or other means, to let the application identify that a packet is lost, the application will not be able to react on this. This situation is illustrated in Figure 4.6 where the 5th packet is lost. The application will only notice that no packets arrive in a larger time span which influences the amount of available data in the buffer.

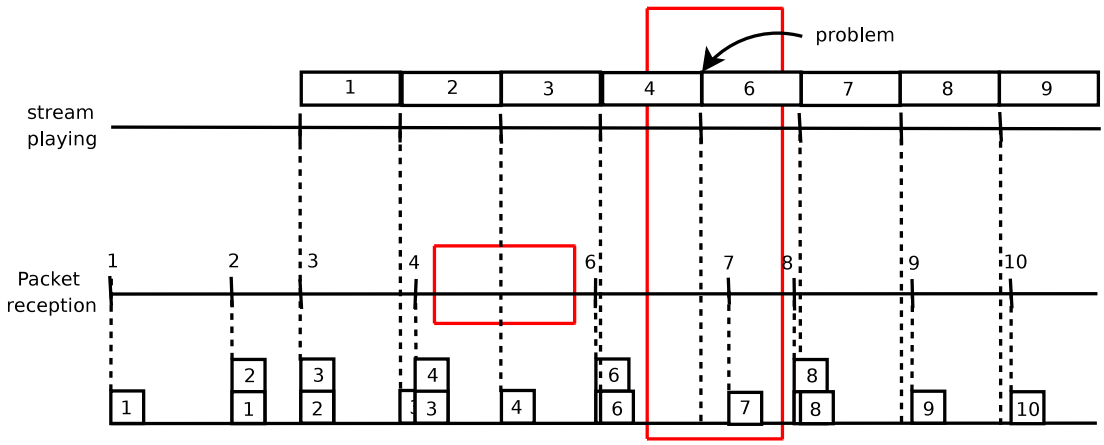

Figure 4.6: The figure shows how a packet loss affects the audio stream playback when there are no sequence numbers. Since the application can not tell that a packet is lost, it will just (wrongly) continue the stream from the next packet.

If sequence numbers are available in the network packets, the application can detect that a packet was missing when it gets the next packet. Instead of playing the wrong packet instead of the lost one, the

application will be able to use some of the recovery mechanisms discussed in Section 2.5.2 on page 22. This situation is illustrated in Figure 4.7 where the recovery is done by inserting silence into the stream. By doing this, the amount of stream data available in the buffer is not affected by the lost packet.

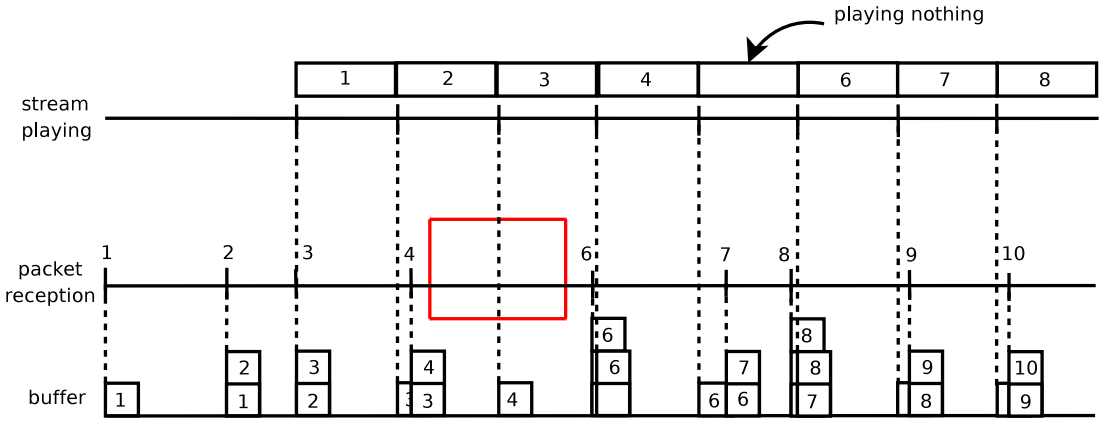

Figure 4.7: The figure shows how the application is able to react to a packet loss when it can detect this through sequence numbers. Note that the stream data available in the buffer is not affected by the loss since the lost packet is replaced.

In the example illustrated in Figure 4.6 the available data in the buffer will be reduced by a whole packet, while in the example illustrated in Figure 4.7 the available data in the buffer will not be affected by the packet loss. This means that when packet losses can be detected by the application, it will be easier to fulfill the skew requirement.

4.4 Summary

Instead of measuring the skew on the audio packets directly another method of checking whether the skew requirement of 80ms is fulfilled was proposed. The numbers used in this chapter is of course bound to the codec which is used, but replacing the audio codec with another codec will only change the bitrate delivered to the transport layer. The size of the buffer used to check if the requirement is fulfilled is specified in terms of time, which can be translated into a number of packets for any given codec. Even though the method proposed assumes an initial packet delay it is believed that it will give a good indication of how the different transport protocols are able to fulfill the skew requirement.

5 Performance Evaluation of 802.11 MAC retransmission

In the experiment described in Section 2.4 on page 13 the stream data is transmitted from the stream server to the stream client using multicast. Multicast traffic is transmitted as broadcast traffic on a WLAN without any retransmission using the basic access procedure and at the defined basic rate(a rate which must be supported by all WLAN clients in the network)[2, p.268,p.281]. The missing retransmission at the data-link layer has an impact on the experienced packet error rate at the transport layer. The experienced packet error rate should be higher when there are no retransmissions. In order to understand the impact of the data link layer retransmissions it has been chosen to conduct an experiment where this mechanism is switched on and off.

5.1 Experiment Setup

In the Linux operating system some WLAN drivers allow the user to configure the maximum number of data-link layer retransmissions through the wireless extension interface. In theory this is also possible with the access point used in the experiment in Section 2.4 on page 13, but trial experiments showed that this does not work as intended another experiment setup using only Linux laptops was chosen instead. As shown in Figure 5.1 four laptops equipped with WLAN interfaces where configured in Ad-Hoc mode in the same Independent Basic Service Set (IBSS). Two laptops was genereting disturbance traffic, and two laptops emulated the audio stream from the experiement in Section 2.4 on page 13 by sending UDP traffic with the same properties.

The packet delay is calculated at the receiver side of the communication only, since UDP packets does not contain any sequence numbers to differentiate packets and since this delay is what the delay requirement specifies. Calculating it like this neglects the delay added by the network before the first packet is received, which should be taken into account as well however an estimate of this delay is given in the experiment in Section 2.4 on page 13 to be 20ms with 10Mbit disturbance traffic. The delay is the inter packet delay between two correctly received packets which means lost packets will result in a larger delay. This is illustrated in Figure 5.2

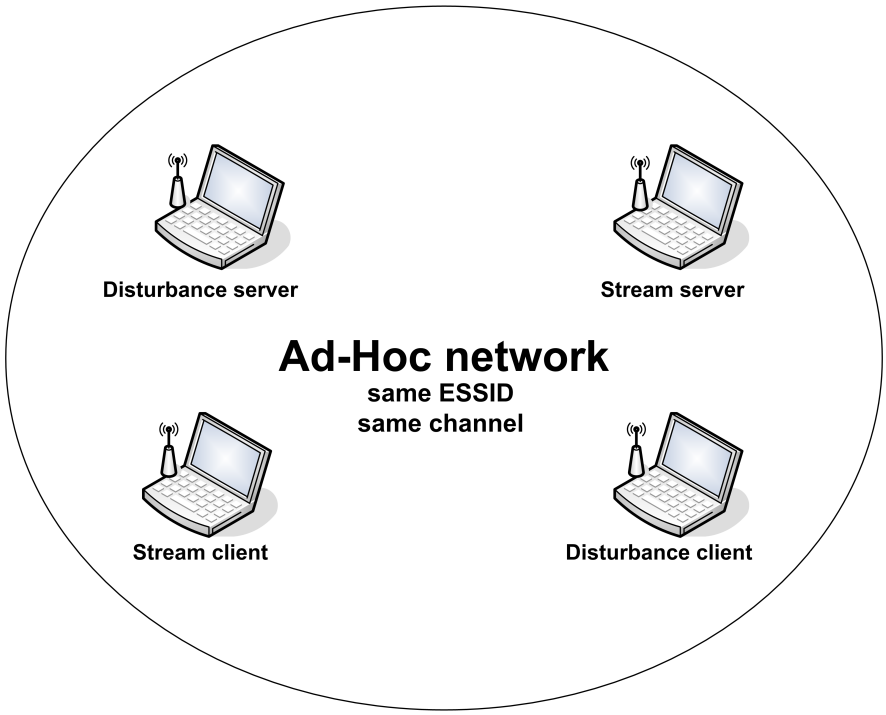

Figure 5.1: The figure shows the experimental setup for the data-link retransmission experiment. Four laptops are connected in an Ad-Hoc WLAN on the same channel with the same ESSID. Two laptops generates disturbance traffic by sending UDP packets from the server to the client while the other two laptops emulate the audio stream from the inital experiment using a packet generator.

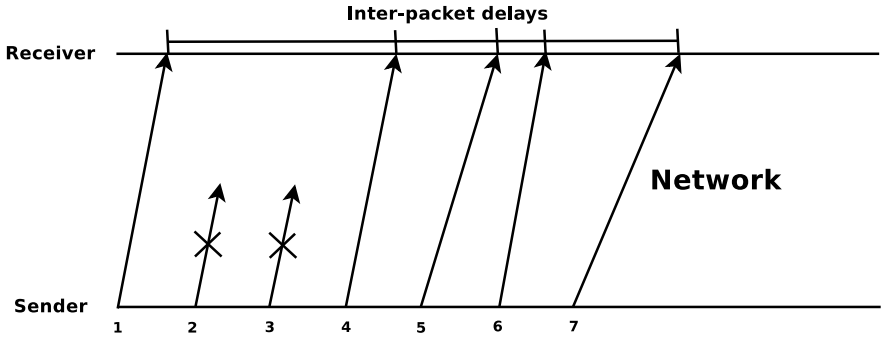

Figure 5.2: The figure shows how the packet delay is calculated. The delay is the inter packet delay at the receiver side of the communication, where lost packets will result in a higher delay e.g. between packet 1 and 4.

5.2 Results

It is expected that the mean inter-packet delay is smaller when the data-link retransmission mechanism is disabled, but because of the way the delays are calculated, the higher loss rate experienced when retransmission is disabled will increase the mean delay. It is also expected that the loss rate will decrease,

with a factor approximately the same as the maximum number of allowed retransmissions, when the retransmission is enabled.

The emulated audio stream was transmitted 30 times with a maximum retransmission value of 0 and 30 times with a maximum retransmission value of 7 (the default value) while the disturbing nodes transmitted UDP packets at a rate of 10Mbit/s. The resulting mean inter-packet delays, maximum inter-packet delays is shown in Figure 5.3.

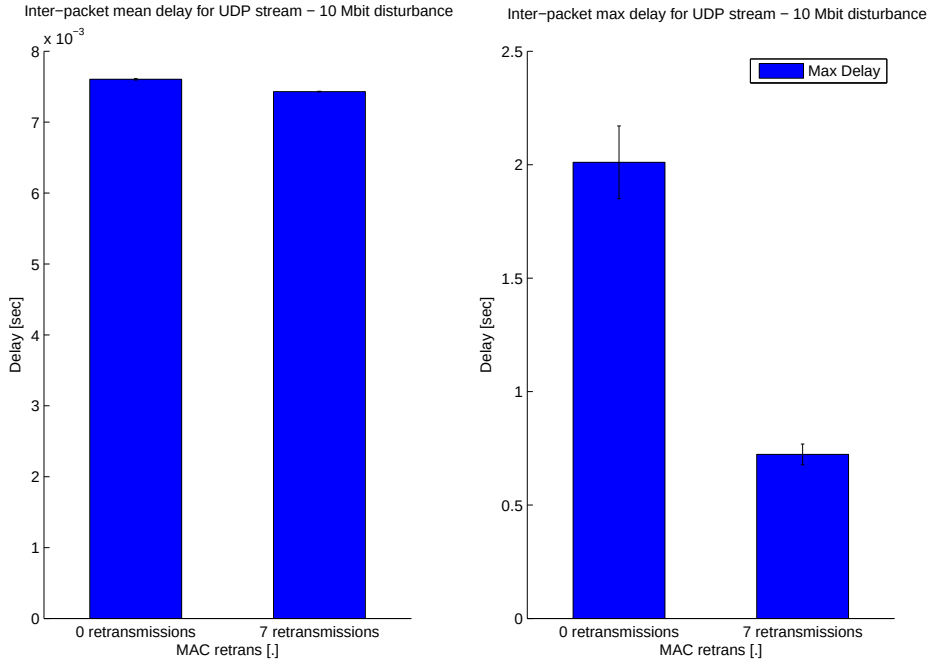

Figure 5.3: The Figure shows the mean and maximum delay for a maximum of 0 and 7 retransmissions at the data-link layer with a disturbance of 10Mbit UDP traffic

With data-link retransmission disabled (0 retransmissions) the mean inter-packet delay is 7.6ms and with maximum 7 retransmissions the mean inter-packet delay is 7.4ms. For 0 retransmissions the maximum inter-packet delay is 2010.3ms and for 7 retransmissions the maximum inter-packet delay is 723.3ms. This is as expected since the lost packets adds to the mean delays. The mean and maximum percentage packet loss is shown in Figure 5.4

The mean packet loss is 2.6063% for maximum 0 retransmissions and 0.3927% for maximum 7 retransmissions while the maximum loss for maximum 0 retransmissions is 3.4699% and 0.6316% for maximum 7 retransmissions.

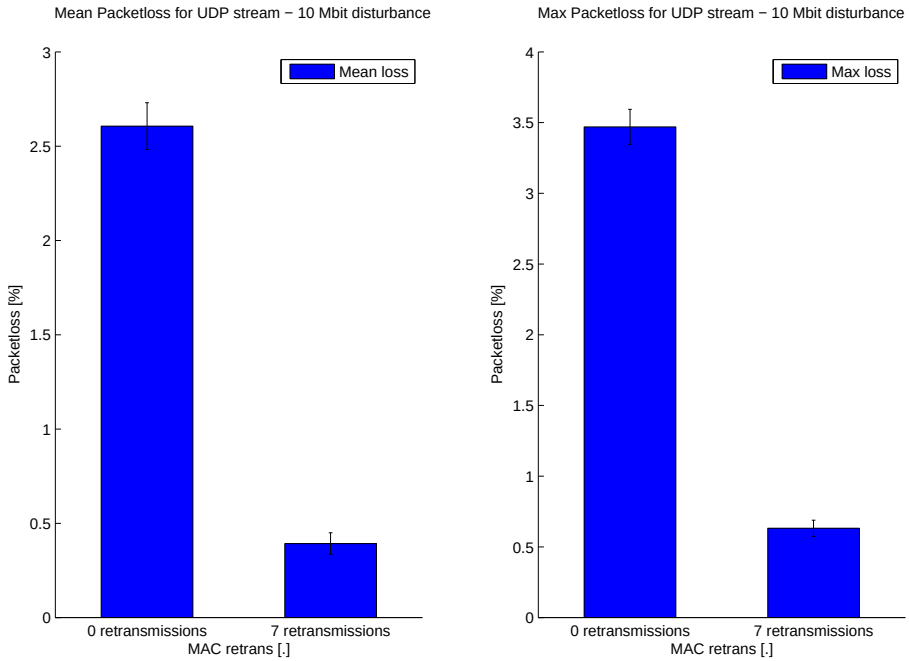

Figure 5.4: The Figure shows the mean and maximum packet loss in percent for maximum 0 and 7 re-ransmissions at the data-link layer with a disturbance of 10Mbit UDP traffic.

5.3 Conclusion

The results show that with a disturbance traffic rate of 10Mbit/s the 802.11 MAC retransmission mechanism in the default setting (7 retransmissions) is exhausted and packets is lost. This means that it can be assumed that the 802.11 MAC retransmission mechanism tries to re-transmit as many times as it is allowed to, and still the mean inter-packet delay is not increasing significantly. With a mean inter-packet delay of $\approx 7.4ms$ there is still time to make either more retransmissions at the 802.11 MAC layer or make retransmissions at higher levels in order to overcome the experienced packet loss.

6 Performance Evaluation of Transport Layer Protocols

Instead of, or in combination with, using the 802.11 MAC retransmission mechanism to prevent packet loss, higher layer protocols can be used as well. This chapter evaluates the performance of the well known reliable transport layer protocol TCP in terms of the delay imposed to the transferred traffic and the ability to handle packet loss at lower layers. In order to be able to evaluate the performance of TCP, it is chosen to evaluate the performance of UDP unicast, traffic and use this as a performance reference. This is mainly done since going directly from multicast UDP traffic to unicast TCP traffic will change the properties of the 802.11 MAC layer as described in Chapter 5 on page 35. Using unicast UDP keeps the 802.11 MAC retransmission active, and should thereby give a better performance in terms of packet loss.

It is chosen to emulate the properties of the audio stream used in the initial experiment in Section 2.4 on page 13 in order to be able to relate the obtained results to an actual audio stream. The experimental setup is shown in Figure 6.1

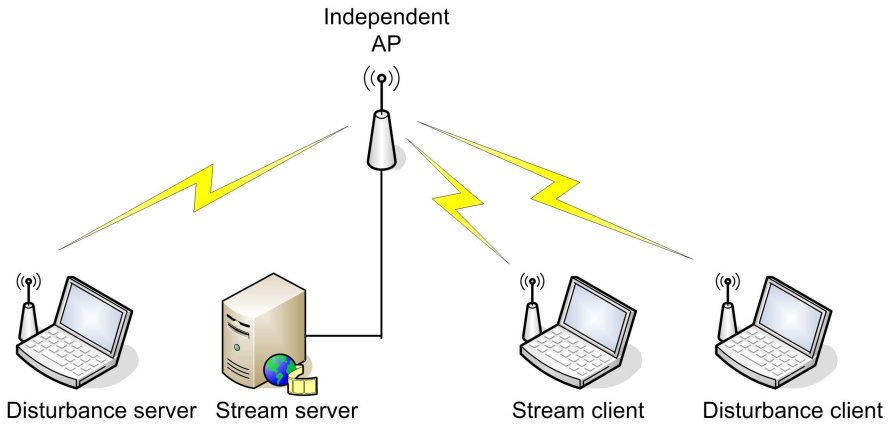

Figure 6.1: The figure shows the experimental setup used to evaluate the performance of UDP and TCP. The stream server is connected to an AP via LAN while both the stream client and the two disturbance nodes are associated to the AP.

The stream server is connected via LAN to an AP while the stream client and two disturbance laptops are associated with the AP. The audio stream is emulated using a packet generator tool called "PackGen"[1],

[1] http://raa.ruby-lang.org/project/packgen/

which can generate TCP and UDP traffic with different bandwidth and packet sizes. It is thereby possible to reconstruct the properties of the PCM audio stream used by Pulse Audio in the initial experiment.

6.1 UDP Performance Evaluation

UDP is a simple transport layer protocol which provides unreliable, and unordered delivery of packets (or datagrams) between applications. This means that if e.g. reliability is a concern, this needs to be handled by higher layer protocols, but also means that the overhead of error checking is not added at the transport level. Since there is some reliability added by the 802.11 MAC layer, evaluating the performance of UDP should reveal the best achievable delay performance from a transport protocol point of view, but also the worst performance in terms of transport protocol reliability. This should make it possible to make a better evaluation of the performance of a reliable transport protocol.

6.1.1 UDP - Experimental Setup

An emulated audio stream is sent from the stream server through the access point to the stream client using PackGen. The size of the UDP packets is chosen to be 1300 Bytes (UDP payload) and the transfer rate to be 1.4 Mbit/s, these values equals the ones measured in the initial experiment done with Pulse Audio. PackGen adds 8 bytes of information to the packet, therefore the configuration value must be 1292 Bytes to achieve 1300 Bytes UDP payload.

The bit stream is transfered in 3 minutes and 40 seconds, in order to emulate the stream length of the MP3 sound file played in the initial experiment. In parallel two laptops acts as disturbance nodes, by sending UDP traffic at different rates using the same access point. This is done to force a packet loss, even with the 802.11 MAC retransmission active. The configuration file to construct this scenario in PackGen is shown in listing 6.1. The keyword DSCP describes the Differentiated Services Code Point of the flow, which is needed in case of prioritization. This is not used in this scenario, but is required to be set. The "from..to:" specifies for duration that PackGen should run.

```
SEND:
  udp:
    -
      name: AudioTestStreamUDP
      host: 10.8.12.137:5002
      bandwidth: 1.4Mb
      packet_size: 1292B
      dscp: cs4
      from..to: !ruby/range 0.0..230.0
```

Listing 6.1: PackGen UDP Configuration file

6.1.2 UDP - Results

The emulated UDP audio stream was transferred 30 times with disturbance rates from 0 to 20Mbit/s UDP traffic in steps of 2Mbit/s. It is assumed that the 30 runs of the test are statistically independent, and thereby normal distribution can be assumed. The network traffic was captured at both the stream server and stream client using Wireshark[2] in order to be able to calculate inter packet delays and packet loss. The mean inter packet delay was calculated at the stream client with each disturbance rate with a 95% confidence interval. The results of the calculations are shown in Figure 6.2 and table 6.1.

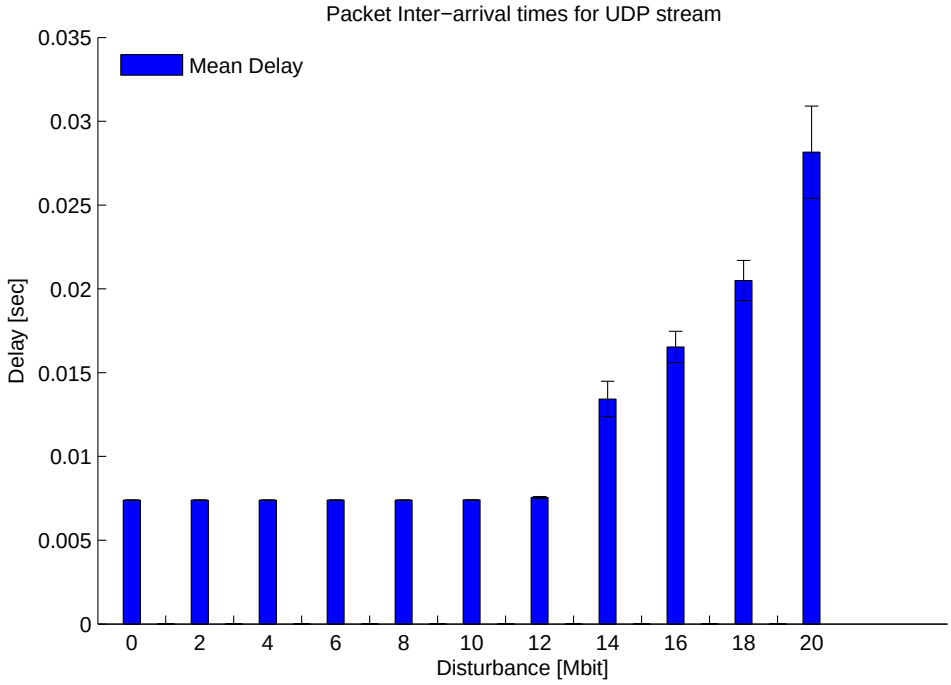

Figure 6.2: The Figure shows the mean inter packet delays for UDP with different disturbance rates with a 95% confidence interval (normal distribution)

[2]http://www.wireshark.org/

Disturb. [Mbit/s]	**0**	**2**	**4**	**6**	**8**
Mean Delay [s]	0,0074	0,0074	0,0074	0,0074	0,0074
95% Conf. +/- [.]	0,0000	0,0000	0,0000	0,0000	0,0000

10	**12**	**14**	**16**	**18**	**20**
0,0074	0,0076	0,0134	0,0165	0,0205	0,0282
0,0000	0,0001	0,0011	0,0009	0,0012	0,0028

Table 6.1: The table shows the values used to produce the graph in Figure 6.2

The mean maximum inter packet delays and the absolute maximum inter packet delays was calculated, using the data from the stream client, for each disturbance rate. The results of these calculations are shown in Figure 6.3 and Table 6.2.

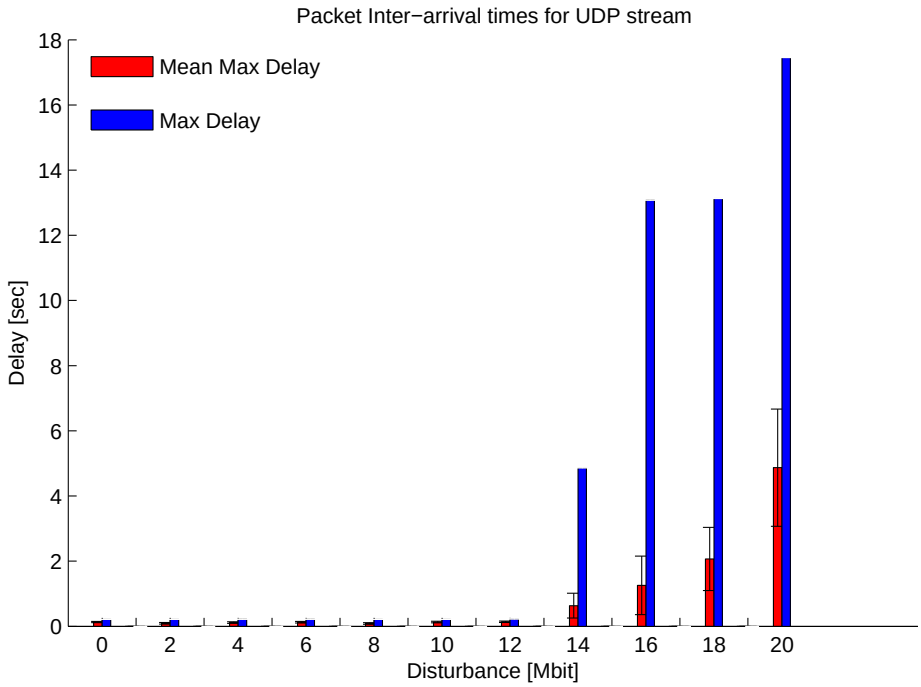

Figure 6.3: The Figure shows the mean of the maximum inter packet delays and the absolute maximum inter packet delays for UDP with XMbit/s disturbance with a 95% confidence interval (normal distribution).

Disturb. [Mbit/s]	**0**	**2**	**4**	**6**	**8**
Mean Max Delay [s]	0,1310	0,0907	0,1163	0,1249	0,0882
95 % Conf. +/- [s]	0,0201	0,0246	0,0243	0,0253	0,0229
Max Delay [s]	0,2132	0,2081	0,2136	0,2173	0,2129

10	**12**	**14**	**16**	**18**	**20**
0,1301	0,1398	0,6351	1,2574	2,0656	4,8700
0,0246	0,0228	0,3828	0,8974	0,9692	1,7992
0,2128	0,2287	4,8648	13,080	13,126	17,454

Table 6.2: The table shows the values used to produce the graph in Figure 6.3

The graphs in Figure 6.2 on page 41 and Figure 6.3 on the facing page shows that the stream is unaffected of the disturbance, when the disturbance is below 12 Mbit/s. In the average the packet inter-arrival time will be approximately 0.0074 seconds, when the link are experiencing no packet loss. This value is as expected from the calculations of the audio content of each packet, performed in chapter 4 on page 28. When the disturbance traffic exceeds 12 Mbit/s the packet inter arrival times increases due to packet loss on the link. It must be noticed that when a packet is lost, the measured time, is the time between two consecutive successful received packets. A lost packet will thereby increase the average inter arrival time significantly.

The mean packet loss was also calculated for the 30 samples with different disturbances with a 95% confidence interval. The results from these calculations are shown in Figure 6.4 and Table 6.3.

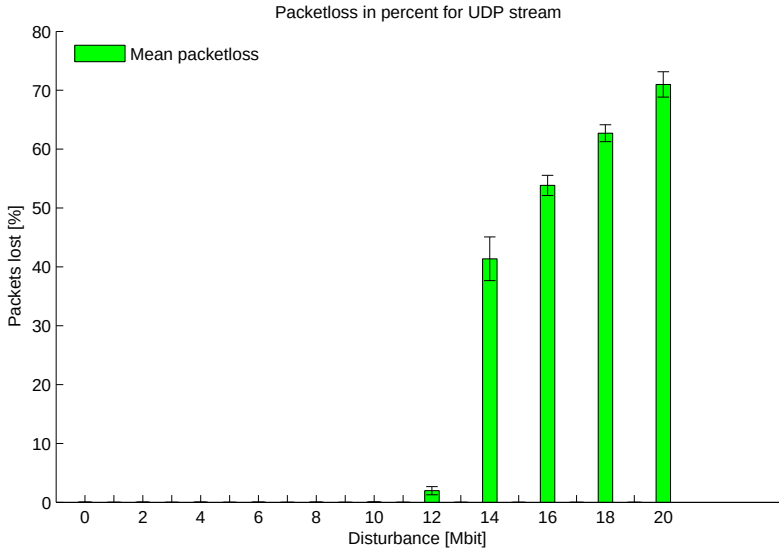

Figure 6.4: The Figure shows the mean packet loss for UDP with X Mbit/s UDP traffic as disturbance and a 95% confidence interval (Normal distribution).

It can be seen from the results in Figure 6.4 that the packet loss of the stream increases with the disturbance, even though the link seems unaffected until 12 Mbit/s disturbance. This is due to the active 802.11 MAC retransmission. When the quality of the link degrades, the 802.11 MAC layer starts retransmitting up to 7 times. When the disturbance raises above 10 Mbit/s, the MAC layer retransmission is exhausted.

Disturb. [Mbit/s]	**0**	**2**	**4**	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
Mean Packet Loss [%]	0,0082	0,0032	0,0035	0,0046	0,0132	0,0350	1,9792	41,369	53,836	62,716	70,989
95 % Conf. +/- [.]	0,0018	0,0000	0,0004	0,0008	0,0056	0,0101	0,6945	3,7101	1,7242	1,4345	2,1527

Table 6.3: The table shows the values used to produce the graph in Figure 6.4

The retransmission technique adds a delay from an end to end perspective. The number of packets which is delayed more than the requirement limit of 80ms was calculated for each level of disturbance, with a 95% confidence interval and is shown in Figure 6.5 and Table 6.4.

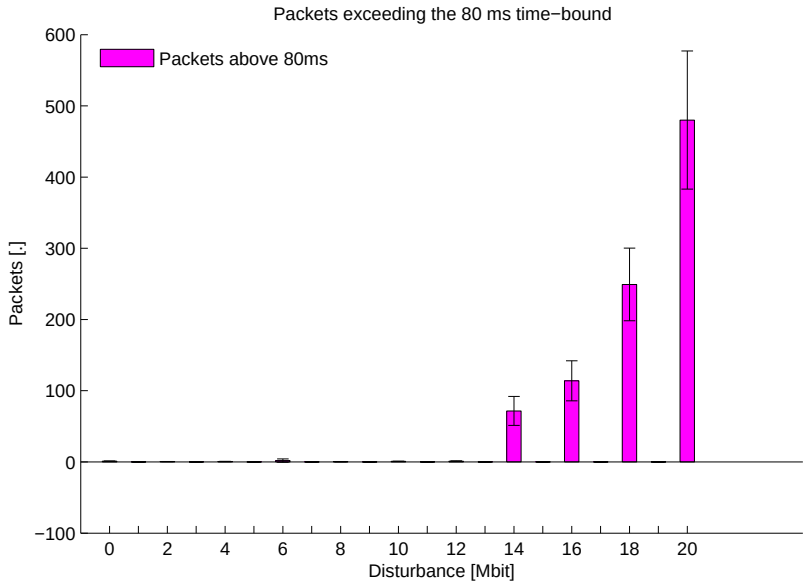

Figure 6.5: The Figure shows the mean number of packets above the requirement limit of 80ms for UDP with X Mbit/s UDP traffic as disturbance and a 95% confidence interval (Normal distribution).

Disturb. [Mbit/s]	**0**	**2**	**4**	**6**	**8**
Above Threshold [.]	1,0667	0,5333	0,6667	1,8333	0,4667
95% Conf. +/- [.]	0,2474	0,1816	0,1716	2,3155	0,2045

10	**12**	**14**	**16**	**18**	**20**
0,7667	1,0667	71,6	113,9667	249,233	480,0333
0,1804	0,3246	20,292	28,1163	51,0421	97,0125

Table 6.4: The table shows the values used to produce the graph in Figure 6.5

The results show that when the disturbance traffic exceeds 12Mbit/s the skew requirement can no longer be fulfilled.

6.1.3 UDP - Conclusion

From the experiments it is concluded that the 802.11 MAC retransmission is exhausted when the disturbance traffic exceeds 10 Mbit/s. The graphs gives a clear indication of retransmission in the range between no disturbance and 12 Mbit/s of disturbance, shown as the constant delay in both mean and maximum case as well as almost no packet loss.

It can furthermore from the results be concluded that the packet loss, and thereby the inter arrival time of the packets, grows as the disturbance traffic increases. At 12 Mbit/s the packet loss requirement is exceeded, with a measurement value of 1,97%. Overall this concludes that UDP cannot be used with disturbances larger than 12 Mbit/s.

6.2 TCP Performance Evaluation

When UDP has been evaluated it is possible to make an evaluation of TCP. TCP is in contrast to UDP a reliable transport protocol which means it guarantees reliable, in order, integrity checked delivery of data. The important aspect of TCP related to this project is the reliable delivery mechanism. TCP uses sequence numbers in the packet header as well as acknowledgement packets to ensure that packets are received correct and keeps retransmitting the packet if it is not. Retransmitting a packet introduces additional delay from an end to end perspective, and this delay as well as the robustness to the properties of the WLAN network will be investigated with a series of experiments.

6.2.1 TCP - Experimental Setup

The physical setup is the same as shown in Figure 6.1 on page 39. The PackGen packet generator is setup to send TCP traffic with a packet payload size of 1300Bytes and a rate of 1.4Mbit/s. This is done with the PackGen configuration show in Listing 6.2. The DSCP option describes the DiffServ prioritizing, which is not used in the scenario but has to be specified. The "from..to" range tells the packet generator to start at time zero and proceed until 230 seconds has elapsed.

```
SEND:
  tcp:
    -
      name: AudioTestStreamTCP
      host: 10.8.12.137:5002
      bandwidth: 1.4Mb
      packet_size: 1300B
      dscp: cs4
      from..to: !ruby/range 0.0..230.0
```

Listing 6.2: PackGen TCP Configuration file

6.2.2 TCP - Calculations

Calculating the packet delay for TCP is different from that of UDP in a number of ways. First of all TCP makes the data available to the receiving application as a stream of bytes instead of packets as with UDP. The data does however still arrive as packets, but these packets can and will vary in size. Another aspect of TCP is that it will continue to retransmit lost packets until they arrive at the receiver or until some maximum time-out is reached. When the maximum time-out is reached the connection is seen as broken and the connection will have to be set up again. Since TCP hand over the received data as a stream of bytes, the order of bytes has to be preserved. This means that some bytes which have already arrived can not necessarily be handed over to the application if some bytes are missing which are being retransmitted. These aspects has to be taken into account when it is calculated what the delay on the data is.

The packets transmitted was captured at the stream client. A sample of the packet trace run through TShark and filtered to only shown the packets going from the stream server to the stream client is shown in Listing 6.3

```
1 918719 7046.264921   10.8.12.59 -> 10.8.12.137  TCP 42338 > rfe [ACK] Seq=43641 Ack=1
      Win=92 Len=1448
2 918721 7046.304293   10.8.12.59 -> 10.8.12.137  TCP 42338 > rfe [ACK] Seq=45089 Ack=1
      Win=92 Len=1448
3 918722 7046.304669   10.8.12.59 -> 10.8.12.137  TCP 42338 > rfe [ACK] Seq=46537 Ack=1
      Win=92 Len=1448
4 918724 7046.324624   10.8.12.59 -> 10.8.12.137  TCP 42338 > rfe [ACK] Seq=50881 Ack=1
      Win=92 Len=1448
```

Listing 6.3: The listing shows the output from the TShark application. TCP packets are sent from the stream server to the stream client. The first number is the packet number set by TShark, then the reception timestamp. The next fields are the IP address of the transmitter and the receiver. Then there is a protocol flag and a transmission and reception port number. At last there are TCP specific options like sequence numbers and length information.

The first number is the packet number set by the packet capture program (TShark), followed by a timestamp in seconds. Next is the transmitter IP address and the receiver IP address, then a protocol identifier, the source port number and the destination port number. The last fields shows the TCP packet specific information e.g. whether the acknowledgement flag is set, what sequence number this packet starts with, the current windows size and the length of the payload. An example of how the seq/ack mechanism of TCP is shown in Figure 6.6 on the next page[14].

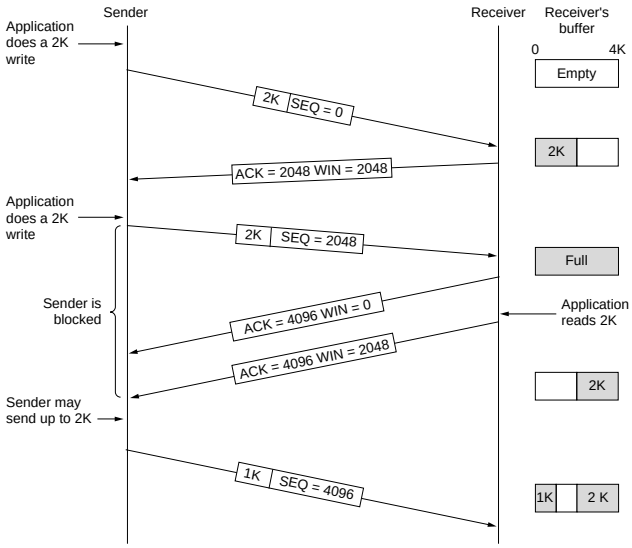

Figure 6.6: The figure shows how the sequence numbers and acknowledgement numbers are used by the TCP protocol. The sequence number is the first byte in the packet.[14, p. 544]

The sequence number informs of the first byte present in the received packet, while the length informs about how many bytes are in one packet. In the snippet shown in Listing 6.3 some packets are lost. This can be seen from the fact that the sequence number plus the length of the second-last packet does not equal the sequence number of the last packet. When some packets are lost there will usually be other packets arriving before these lost packet are retransmitted and finally arrives. The packets which arrives in between can be marked with an inter packet delay of zero since they will be immediately available when the lost packets finally arrive.

A python script was used to calculate the delay of each packet. An activity diagram showing how the script works is illustrated in Figure 6.7

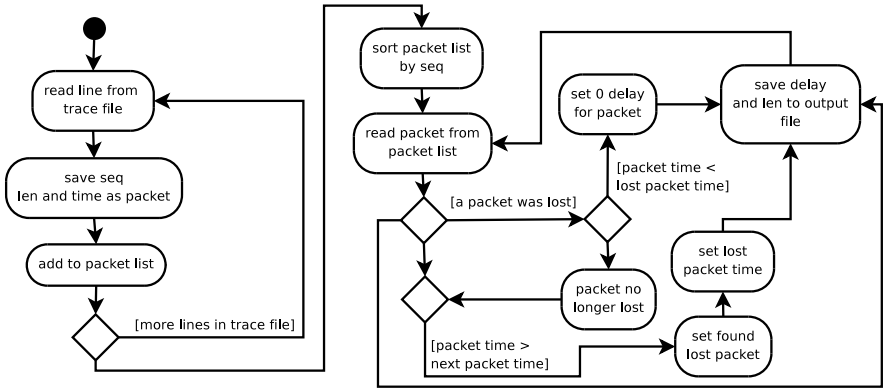

Figure 6.7: The figure shows an activity diagram of the tcp inter packet delay calculation script. First the data for each packet is saved in a list. Then this list is sorted by sequence numbers. Then for each packet a packet delay is calculated and saved with the length of the packet. If a packet is received while another packet was being retransmitted it will get a zero delay.

6.2.3 TCP - Results

The emulated audio stream was transferred 30 times using TCP as transport protocol with disturbance rates from 0 to 12Mbit/s UDP traffic in steps of 2Mbit/s. The network traffic was captured at both the stream server and stream client using Wireshark in order to be able to calculate packet delays and packet loss. The mean inter packet delay was calculated at the stream client with X Mbit/s of disturbance with a 95% confidence interval and the results are shown in Figure 6.8.

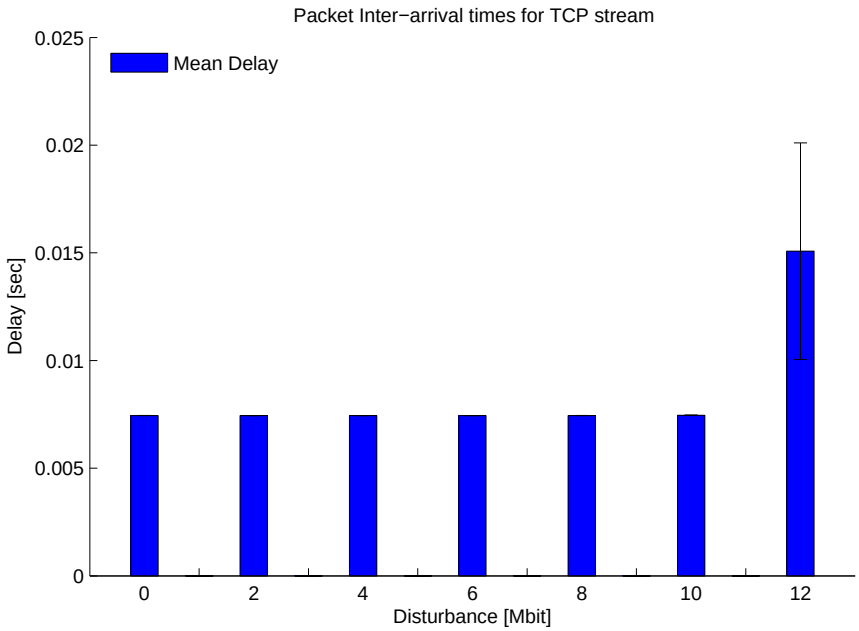

Figure 6.8: The figure shows the mean inter packet delays of the emulated audio stream packets when using TCP as transport protocol with disturbance rates from 0 to 12Mbit/s UDP traffic in steps of 2Mbit/s. The results are plotted with 95% confidence interval (normal distribution).

When the disturbance rate was set to 14Mbit/s and above the TCP connection would be reset at some point in the stream, probably since the maximum TCP timeout value is reached. Since it was not possible to run the full set of samples it was chosen to leave out the higher disturbance rates from the TCP experiment.

Disturb. [Mbit/s]	**0**	**2**	**4**	**6**	**8**	**10**	**12**
Mean Delay [s]	0,0074	0,0074	0,0074	0,0074	0,0074	0,0075	0,0151
95% Conf. +/- [s]	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,0050

Table 6.5: The table shows the values used to produce the graph in Figure 6.8

The mean maximum inter packet delay and the absolute inter packet delays were calculated at the stream client with X Mbit/s of disturbance with a 95% confidence interval. The results from the calculations are shown in Figure 6.9.

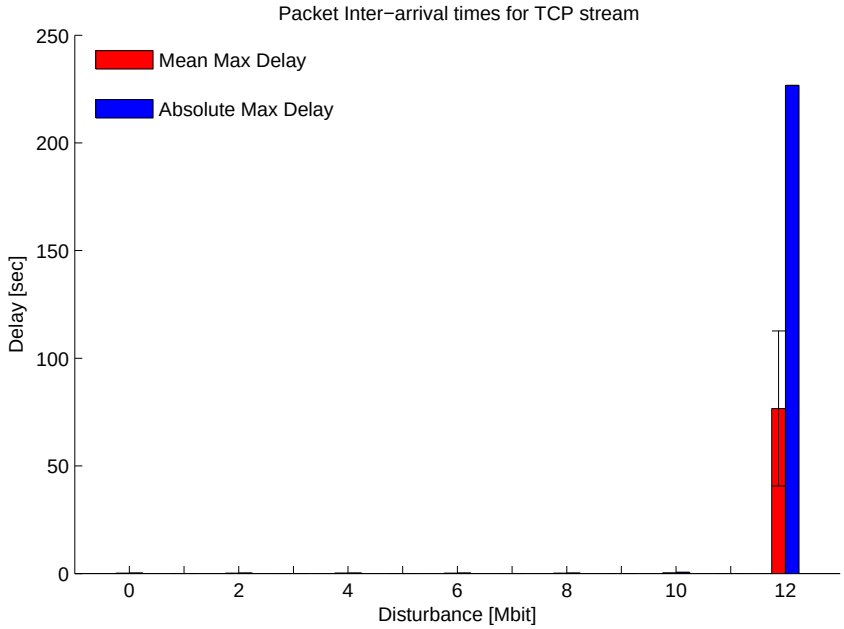

Figure 6.9: The figure shows the mean and absolute maximum inter packet delays of the emulated audio stream packets when using TCP as transport protocol with disturbance rates from 0 to 12Mbit/s UDP traffic in steps of 2Mbit/s. The results are plotted with 95% confidence interval (normal distribution).

Disturb. [Mbit/s]	**0**	**2**	**4**	**6**	**8**	**10**	**12**
Mean Max Delay [s]	0,1241	0,1048	0,1367	0,1238	0,1228	0,2168	76,6501
95% Conf. +/- [s]	0,0252	0,0234	0,0248	0,0225	0,0218	0,0387	36,0229
Max Delay [s]	0,2131	0,2077	0,2125	0,2127	0,2129	0,5861	226,7051

Table 6.6: The table shows the values used to produce the graph in Figure 6.9

The results show that when the disturbance exceeds 10Mbit/s TCP can no longer transmit with the required rate. The maximum delays gives an idea about how long it takes to continue to retransmit every packet until it is received correctly like TCP does.

The mean number of inter packet delays exceeding 80ms (the skew requirement) was calculated to get a first impression of whether the skew requirement is fulfilled. When the inter packet delay exceeds 80ms the requirement is not fulfilled and as shown in Figure 6.10 and Table 6.7 this happens for TCP when the disturbance rate becomes 10Mbit/s.

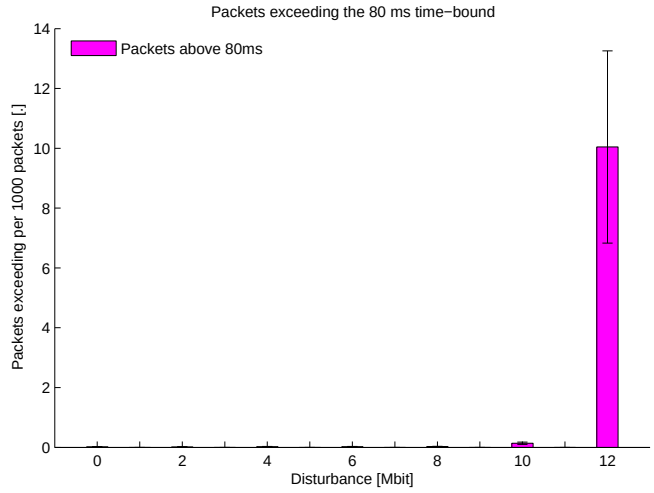

Figure 6.10: The figure shows the mean number of inter packet delays exceeding 80ms with disturbance rates from 0 to 12Mbit/s UDP traffic in steps of 2Mbit/s.When the inter packet delay exceeds 80ms the skew requirement can not be fulfilled. The results are plotted with 95% confidence interval assuming normal distribution.

Disturb. [Mbit/s]	**0**	**2**	**4**	**6**	**8**	**10**	**12**
Above Threshold [.]	0,0204	0,0162	0,0247	0,0247	0,0279	0,1395	10,043
95% Conf. +/- [.]	0,0061	0,0063	0,0054	0,0063	0,0097	0,0386	3,2133

Table 6.7: The table shows the values used to produce the graph in Figure 6.10

Finally an application buffer was simulated (as described in Chapter 4 on page 28) using a Python script to evaluate exactly how many times the skew requirement was not fulfilled. Figure 6.11 shows an activity diagram of the script. The input to the script is the calculated inter packet delays and the length of the received packets in bytes. For each packet the time value and the length is read. The time value is added to the overall time and the length is added to the buffer. If the stream is starting up it is checked if the overall time value is larger than 70ms since that would mean the skew requirement is not fulfilled. If the overall time value is larger than 70ms the stream is restarted. Otherwise it is checked if the buffer is full, and if it is the stream playing can be started, by removing the first packet and setting the next deadline.

When the stream is playing it is checked whether there was any deadlines since last packet reception. If there were, then packets are removed and the deadline is incremented until it is higher than the overall time value. If the buffer gets below 0 Bytes, it means a deadline could not be reached, and the stream is restarted meaning the skew requirement was not fulfilled.

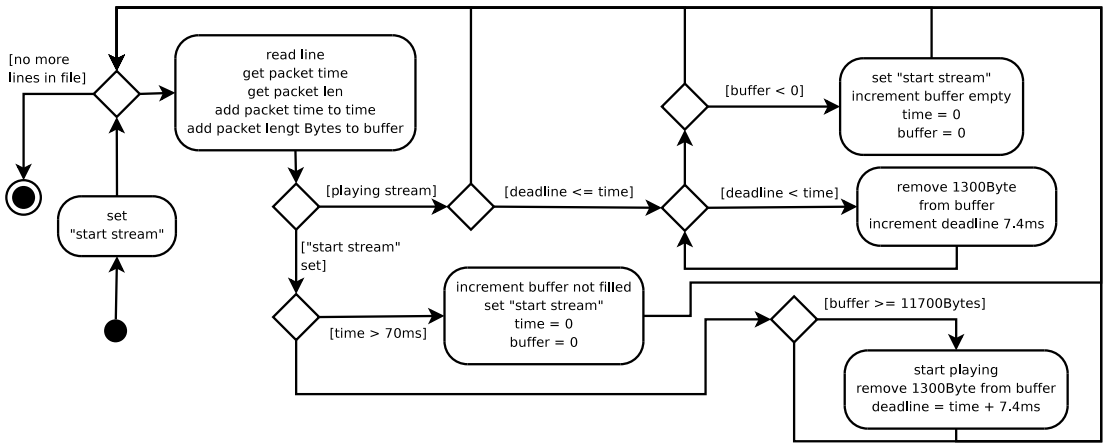

Figure 6.11: The figure shows an activity diagram of the script used to do the buffer calculations. Two incidents are important. If the buffer cannot be filled within 70ms the stream is restarted, and if the buffer becomes empty the stream is restarted. These incidents happen when the skew requirement is not fulfilled.

The results are shown in Figure 6.10 on the preceding page and Table 6.8.

Disturb. [Mbit/s]	**0**	**2**	**4**	**6**	**8**	**10**	**12**
Buffer Empty [.]	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,9918
95% Conf. +/- [.]	0,0000	0,0000	0,0000	0,0000	0,0000	0,0000	0,5074
Buffer Not Full [.]	0,0086	0,0000	0,0000	0,0000	0,0000	0,0064	7,1392
95% Conf. +/- [.]	0,0052	0,0000	0,0000	0,0000	0,0000	0,0063	4,1841

Table 6.8: The table shows values used to produce the graphs in figure 6.12

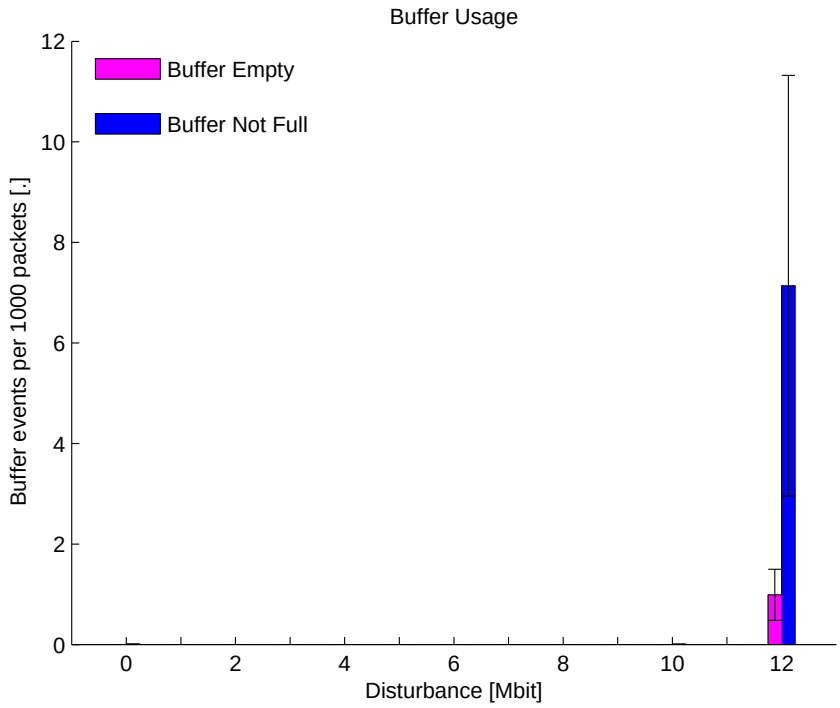

Figure 6.12: The figure shows how many times the simulated application buffer became empty or did not become full, which leads to a skew above 80 ms.

The results shows that the number of times the skew requirement could not be fulfilled is approximate 8 times (buffer not filled and buffer empty) per 1000 packets when the disturbance rate becomes 12Mbit/s. Compared to the number of packets exceeding 80ms in inter packet delay, which is approximately 10 times per 1000 packets, it becomes clear that the exact number of times the skew requirement is not fulfilled is higher than what the packets exceeding 80ms inter packet delays shows.

6.2.4 TCP - Conclusion

When the disturbance rate becomes 10Mbit/s and above the 802.11 MAC retransmission becomes exhausted and packets are lost which TCP needs to retransmit. When the number of lost packets are low TCP is able to retransmit the packets fast enough to be able to fulfil the skew requirement of 80ms, but as the disturbance rate increases to 12Mbit/s TCP can no longer get the packets retransmitted fast enough. When the disturbance rates become more than 14 Mbit/s TCP is not even capable of maintaining the connection.

There are a number of possible reasons for why TCP does not perform very well. First of all TCP uses rate control algorithms to avoid congestion in the network, and these algorithms uses packet loss as an indication of congestion in the network. This is a reasonable assumption on wired links where

the bottleneck in the network most often is the routers forwarding the traffic, but this is not the case for a wireless link. When packets are lost, TCP will slow down instead of retransmitting the packet immediately which means it will be harder to meet the deadline. Secondly TCP delivers an ordered byte stream of data to the application. This means that if just one byte is lost, this byte will have to be retransmitted and correctly received before the following bytes can be handed over to the application. In situations where it would be possible to fulfil the skew requirement by not waiting for a retransmit of a relative small amount of data, TCP will not be able to since it will either deliver the data in order or break the connection when the maximum time-out is reached.

Using TCP instead of UDP as a transport protocol removes the packet loss experienced by UDP ($\approx 2\%$ for 12Mbit/s disturbance) but since the skew requirement is not fulfilled, it would probably give a better user experience to accept the loss of UDP than having to restart the stream numerous times. Based on this and the fact that TCP does not even work at higher disturbance rates it has been concluded that TCP is not usable for the project use-case.

It is assumed that the first packet of the stream is received with a delay of exactly 10ms. This will not reflect reality especially for high disturbance rates where the mean delay will be higher. This means that for high disturbance rates the buffer calculations will actually yield results in favor of TCP. This impact is however not considered large enough to influence the conclusion.

7 A Simple Retransmission Protocol

The previous chapters has evaluated the performance of 802.11 MAC retransmission as well as TCP and UDP transport protocols in terms of inter packet delay and packet loss. Throughout the performance evaluation several observations and considerations has been made, from which the design and implementation of a simple retransmission protocol is derived. This chapter describes the design and implementation of this simple retransmission protocol and evaluates the performance of this compared to the results for TCP.

7.1 Design

Implementing a complete transport protocol in e.g. Linux would mean programming at the kernel level of the operating system. To avoid the difficulties of working at that level it is instead chosen to implement a retransmission protocol at the application level. This means using a already existing transport protocol and implement a retransmission protocol on top of this. UDP has been chosen since this is most widely available low overhead transport protocol there is. UDP only handles the addressing of the service running at the receiving side of the communication (also called application multiplexing) as well as integrity verification of packet header and payload.

7.1.1 Reliability

In order to be able to detect if a packet is lost, some kind of unique packet identification has to be embedded into the packets. Usually this is done by having a sequence number in the packet which can be used by the receiver to send back a packet which acknowledges that the packet is correctly received. It has been chosen to use a simple sequence number scheme where the first packet is numbered 1, the next is number 2 etc. This continues as long as the communication is going on.This sequence number will be embedded in each transmitted packet as the UDP payload followed by the audio stream data. When a packet is received, an acknowledgement packet containing only this sequence number is returned to the transmitter.

It is chosen to specify in the design, that the sequence number must be implemented as a sequence of characters instead of e.g. one XByte binary value. Embedding the sequence number in the packet as a string, allows the use of the TShark application to parse the UDP payload data and read the sequence number. This is illustrated in Listing 7.1 where the output from TShark on a UDP packet capture is shown.

```
935   3.455356   10.8.5.165 -> 10.8.5.160   UDP Source port: 60209  Destination port:
     commplex-main

0000  00 19 cb 85 fd a2 00 11 25 4b 7b 20 08 00 45 00    ........%K{ ..E.
0010  05 30 aa 8a 40 00 40 11 6b de 0a 08 05 a5 0a 08    .0..@.@.k.......
0020  05 a0 eb 31 13 88 05 1c 2a 8b 73 65 71 3a 34 36    ...1....*.seq:46
0030  38 20 50 41 59 4c 4f 41 44 3a                      8 PAYLOAD:

936   3.455422   10.8.5.160 -> 10.8.5.165   UDP Source port: commplex-main
    Destination port: 60209

0000  00 11 25 4b 7b 20 00 19 cb 85 fd a2 08 00 45 00    ..%K{ ........E.
0010  00 23 00 00 40 00 40 11 1b 76 0a 08 05 a0 0a 08    .#..@.@..v......
0020  05 a5 13 88 eb 31 00 0f a8 ed 61 63 6b 3a 34 36    .....1....ack:46
0030  38                                                 8
```

Listing 7.1: The Listing shows output from the Tshark application when it parses the UDP payload. The first packet shows the sequence number 468 in the second column, followed by a string "PAYLOAD:". The next packet is the acknowledgement packet returned to the transmitter, acknowledging the received sequence number.

Having the sequence number available to the application layer has a positive effect on the streaming use case because the ordering can be handled by the application layer. This is an advantage in the sense that if packets are missing in between received packets, this will not force the player to stop playing. Instead it can just choose to ignore packets that are not delivered in time or otherwise try to repair the missing data. As described in chapter 4 on page 28 this means the skew requirement can be fulfilled by e.g. playing silence, as long as the packet loss requirement is still fulfilled.

7.2 Retransmission algorithm

There are different methods to decide whether and when a packet should be retransmitted. One possibility is to consider a packet as lost when an acknowledgement is successfully received for the next packet in the sequence. The drawback is that this retransmission technique has to wait for the next packet to be recorded, transmitted and acknowledged before a retransmission of the first packet can take place. This introduces a potentially avoidable delay making it harder to fulfill the skew requirement.

Another solution for deciding whether a packet is lost is to record the time when a packet is transmitted. It is then possible to evaluate the time elapsed and compare this to a timeout value. This timeout must be set to a value near two times the expected end-to-end delay of the transmission path. The receiving end of the transmission must, in this scenario as well, acknowledge the received packet immediately after reception. If this acknowledge is not received by the sender, within the timeout range, a retransmission initiates. This solution does not affect the timing of the original packets and thereby introduces no extra delay to the stream. It does however introduce a bandwidth usage overhead since packets can be retransmitted if the acknowledgement is delayed too much. This will result in duplicate packets received which means unnecessary bandwidth usage.

The audio stream use case for this project has a hard real time skew requirement. If a packet does not arrive at the application layer within the maximum allowed skew bound of 80 ms, there is no need for retransmitting it, since it will be considered lost anyway. It is therefore chosen to use the first algorithm for retransmission only since this potentially yields the best results in terms of delay.

A set of parameter values for the retransmission algorithm has to be chosen. This is a maximum timeout value for a packet after which no more retransmissions will be performed and the retransmission timeout value which decides how often a retransmission is performed. It is decided to use a retransmission timeout of 20ms and let the maximum timeout value be decided from the amount of data buffered at the receiver. Since 70ms are buffered at the receiver this is the amount of time it is possible to retransmit a packet until it will be declared lost anyway. With the chosen retransmission timeout of 20ms this gives the ability of 3 retransmissions.

The transmitter side of the chosen retransmission algorithm is illustrated in Figure 7.1 as an activity diagram. To simplify the implementation it is chosen to reflect the emulated stream in the design as well.

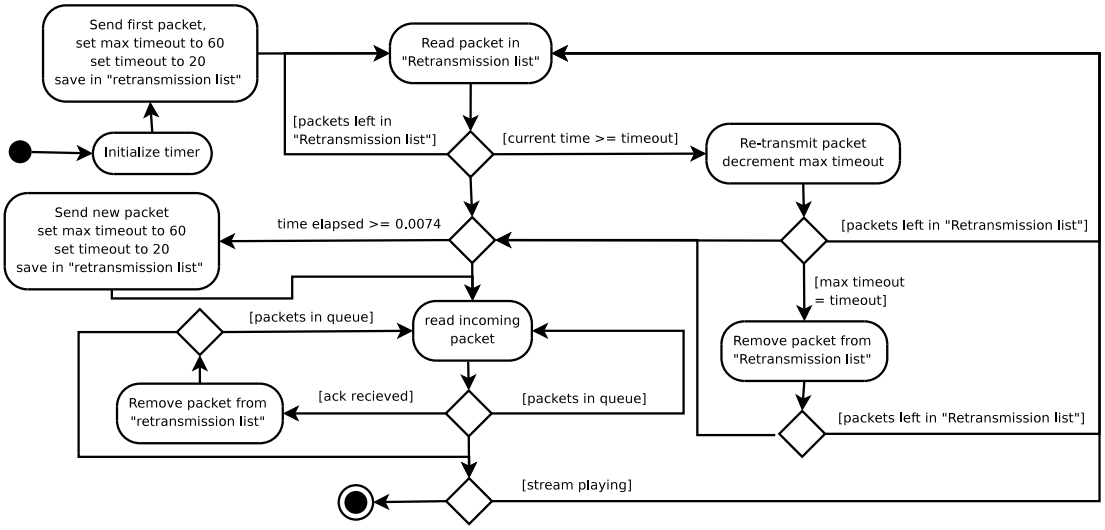

Figure 7.1: The figure an activity diagram of the chosen algorithm for the simple retransmission protocol. A retransmission list is used in combination with a timer to keep track of which packets are not correctly received.This list is looped through to decide whether it is time to retransmit a packet. If a packet is retransmitted the maximum timeout value is decremented to keep track of then a packet should be taken out of the list. If the elapsed time is more than or equal to 0.0074s a new packet is transmitted. The incoming queue is then emptied for received acknowledgements and the retransmissions list is updated accordingly.

A retransmission list is used in combination with a timer to keep track of which packets have to be retransmitted, as well as when it is time to send a new stream packet. First the timer is started and the first stream packet is sent. Information about the first packet is added to the retransmission list, which is the sequence number, acting as the unique packet identifier, the maximum timeout value and the value of the next timeout. The the main loop of the algorithm is entered.

The retransmission list is looped through to decide whether it is time to retransmit a packet. If a packet is retransmitted the maximum timeout value is decremented to keep track of then a packet should be taken out of the list. If the maximum timeout becomes lower than the timeout value the packet is removed from the retransmission list.

If the elapsed time is more than or equal to 0.0074s a new packet is transmitted to emulate the rate of the audio stream. The incoming queue is then emptied for received acknowledgements and the retransmissions list is updated accordingly. The main loop continues as long as there are more audio stream data.

The receiver side of the chosen retransmission algorithm is shown in Figure 7.2 as an activity diagram.

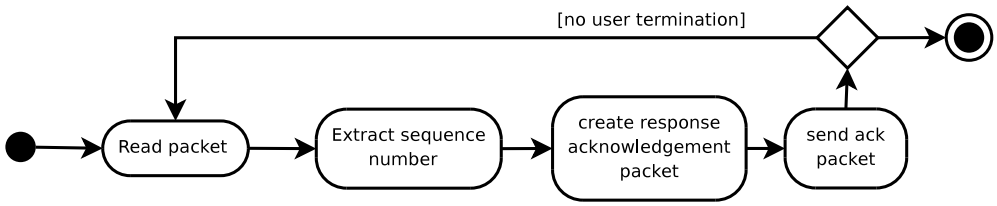

Figure 7.2: The Figure shows an activity diagram of the receiver side of the simple reliable protocol. The receiver enters a loop which is only terminated on user request. When a packet is received the sequence number is extracted and an acknowledgement packet is constructed an returned to the transmitter.

The receiver enters a loop which is only terminated on user request. When a packet is received the sequence number is extracted and an acknowledgement packet is constructed an returned to the transmitter.

7.3 Implementation

It is chosen to use Python to implement the simple retransmission protocol since this is well suited for rapid prototyping and has solid support for network programming through a socket library which wraps around the UNIX socket API.

It is chosen to describe the implementation by showing the actual code in snippets and explaining how it works, since the implementation is not very many lines of code. Listing 7.3 shows how the needed python libraries are imported and the variables needed to control the timing of the retransmission is set. `Mttl` is introduced as an easier way of keeping track of when to stop retransmitting a packet. When the `Mttl` value reaches 1 the packet is removed from the retransmission list. The next set of variables is used to control the timer for emulating the correct stream rate and to use for checking if packets timeout values have been reached. Then some packet payload date is constructed as a string. At last an empty dictionary `airPackets` is created which becomes the retransmission list. A dictionary is chosen since it stores key,value pairs allowing easy look up by using the sequence number as key to look up the necessary packet information.

```
#!/usr/bin/python
import socket,sys,time

tur = 0.020 # Time Until Re-transmit - 20 milliseconds
maxTimeout = 0.060
Mttl = int(maxTimeout/tur) #Max Time To Live - how many retransmission we can do

oldTime = 0
timeVal = time.time()
elapsed = 0

payload=' PAYLOAD::'*130 # 1 char 1 Byte #Create some payload data for packets
airPackets = {}
```

In Listing 7.2 a non-blocking UDP socket (`SOCK_DGRAM`) is created by setting a timeout for the socket. This means that when data is read from the socket it will not block longer than maximum 1ms before it will continue if there is not enough data available. Creating the socket like this makes it possible to implement both the retransmission and the emulated stream as one loop. The first commandline argument is used as IP address to send data to, while the second commandline argument is used as the port number. By calling connect on a UDP socket, even though it is not connection oriented, it is not necessary to specify the destination host and port each time a packet is transmitted. The stream packet sequence number is controlled by the loop variable `i` and converted to a string in the variable `seq`. Finally the first packet is transmitted by appending payload data to the sequence string until a 1300Byte packet is created. Then the packet is saved in the retransmission list with the timeout value and the maximum time-to-live value.

```
s = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
s.settimeout(0.001)
host = sys.argv[1] # server address from cmdline
port = int(sys.argv[2]) # server port from cmdline
s.connect((host, port))

i = 1
seqnr = 'seq:'+str(i)

#send first packet and save info for retransmission
s.send(seqnr+payload[:(1300-len(seqnr))])
airPackets[i] = ((timeVal+tur),Mttl)
```

Listing 7.2: Initialize non blocking socket,loop control and retransmission list and send the first packet

When the initialization is done the main loop is entered as shown in Listing 7.3. This loop controls how long the emulated audio stream should be transmitted, here a value of 4055 packets is approximately 30s. First the timer information is updated before the retransmission list is traversed to look for possible retransmissions. The timeout value of each packet in the retransmission list is evaluated against the current time and the packet is retransmitted if the timeout is reached. If a packet is retransmitted its

time-to-live value is decremented and if this value reaches one it is removed from the retransmission list.

```
while(i < 4055):
    oldTime = timeVal
    timeVal = time.time()
    diff = timeVal-oldTime
    elapsed += diff

    for packet in airPackets[:]:
        timeout,ttl = airPackets[packet]
        if(timeVal >= timeout): #re-transmit
            seqnr = 'seq:'+str(packet)
            s.send(seqnr+payload[:(1300-len(seqnr))])
            if(ttl == 1):
                airPackets.pop(packet) #don't re-transmit this again
            else:
                airPackets[packet] = ((timeout+tur),(ttl-1)) #update packet
```

Listing 7.3: Enter the transmission loop until the stream ends. Update time values and loop through the retransmission list to check for retransmissions.

The last part of the implementation controls the rate of the emulated stream by sending a packet if/when 7.4ms have elapsed as shown in Listing 7.4. Finally it is tried to read an ack packet from the socket and if an acknowledgement packet is received this packet is removed from the retransmission list.

```
    if(elapsed >= 0.0074): #send next packet according to rate
        i += 1
        seqnr = 'seq:'+str(i)
        s.send(seqnr+payload[:(1300-len(seqnr))])
        airPackets[i] = ((timeVal+tur),Mttl) #save info for retransmission
        elapsed = 0
    try:
        data,addr = s.recvfrom(1500) # try to read ACK
    except socket.error, msg:
        continue
    if(data):
        ack = int(data[4:])
        if ack in airPackets: #remove from retransmission list
            airPackets.pop(ack)
s.close()
```

Listing 7.4: The emulated stream rate is controlled with the elapsed time, sending a packet for each 7.4ms. The last part reads in acknowledgement

Since the implementation of the receiver is very simple, a while loop that read packets and returns the sequence number in an acknowledgement packet, it is chosen not to include the code from that in the report.

7.4 Performance Evaluation

This section describes the performance evaluation done for the simple retransmission protocol. The protocol is evaluated by using the same calculations as for TCP in order to be able to compare the results.

The experimental setup used to perform measurements on the simple retransmission protocol is the same as shown in Figure 6.1 on page 39. The transmitter side of the simple retransmission protocol is used to emulate the stream server and the receiver side to emulate the stream client. A series of experiments was conducted with UDP disturbance traffic from 6 to 20 Mbit/s in 2Mbit/s intervals, where the emulated audio stream was transmitted from the stream server to the stream client. It was chosen to cut down the audio stream time from 3minutes and 40seconds in the initial experiment to 30 seconds in order to keep the computational time of the calculations at a tolerable level and since it is believed that 30 seconds for each sample is enough to show the important results. It was also chosen to not conduct experiments for disturbances below 6Mbit/s since these experiments should not add new information because the disturbance is too low to trigger the simple retransmission protocol. For each interval 30 samples where recorded in order to be able to assume normal distribution when calculating the confidence intervals.

7.4.1 Results

Since the simple retransmission protocol is specifically designed with the audio stream use case in mind, it is expected that it will outperform TCP in terms of fulfilling the skew requirement. Since it does not retransmit packets more than maximum 3 times, it is expected that more packets will be lost as a tradeoff with performing better for the skew. Since there are no bandwidth control in the simple retransmission protocol, it will not back off to reduce the bandwidth usage but resend packets whenever they are concluded to be lost. This will most likely result in a considerable increase in the bandwidth used to transmit the audio stream when the disturbance rate increases.

The first calculations done for the simple retransmission protocol concerns the mean inter packet delays experienced while UDP disturbance traffic was transmitted from 6 to 20Mbit/s in 2Mbit/s intervals. The results are shown in Figure 7.3 and Table 7.1 with a 95% confidence interval.

Disturb. [Mbit/s]	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
Mean Delay [s]	0,0076	0,0076	0,0075	0,0096	0,2698	3,5883	2,8666	3,6917
95% Conf. [s]	0,0001	0,0002	0,0000	0,0001	0,5044	1,3920	1,3639	1,4293

Table 7.1: The table shows the values used to produce the graph in Figure 7.3

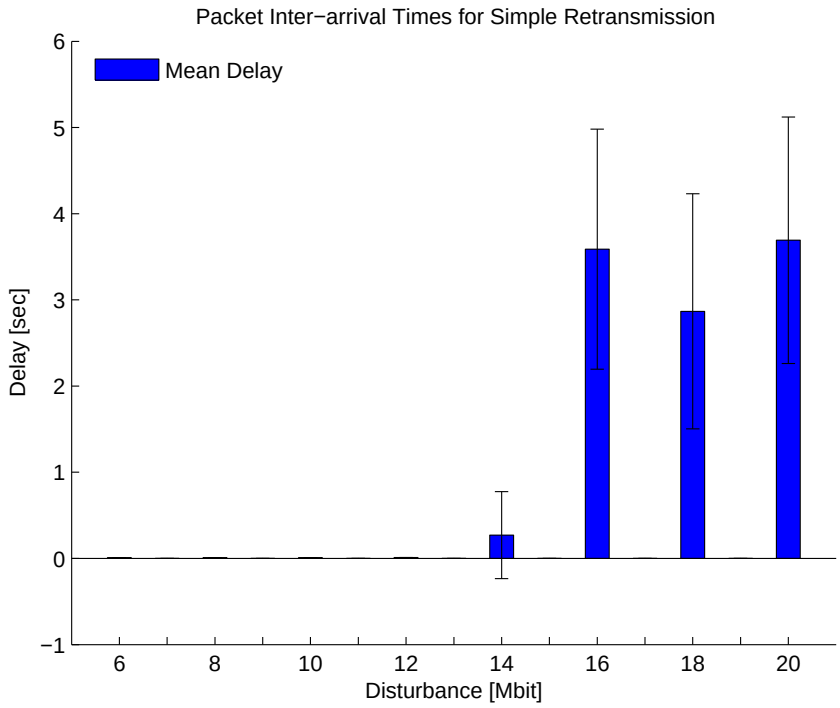

Figure 7.3: The figure shows the mean inter packet delays of the emulated audio stream packets when using the developed simple retransmission protocol with disturbance rates from 6 to 20Mbit/s UDP traffic in steps of 2Mbit/s. The results are plotted with 95% confidence interval assuming normal distribution.

Calculations was also done for the mean and absolute maximum inter packet delays experienced while UDP disturbance traffic was transmitted from 6 to 20Mbit/s in 2Mbit/s intervals. The results are shown in Figure 7.4 and Table 7.2 with a 95% confidence interval.

Disturb. [Mbit/s]	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
Mean Max Delay [s]	0,6632	0,0879	0,0195	0,3317	1,7696	14,821	11,840	15,187
95% Conf. +/- [s]	0,3323	0,1267	0,0020	0,2206	1,9955	5,4503	5,3675	5,5066
Max Delay [s]	1,9738	1,9608	0,0386	1,9676	30,901	31,234	31,299	31,564

Table 7.2: The table shows the values used to produce the graph in Figure 7.4

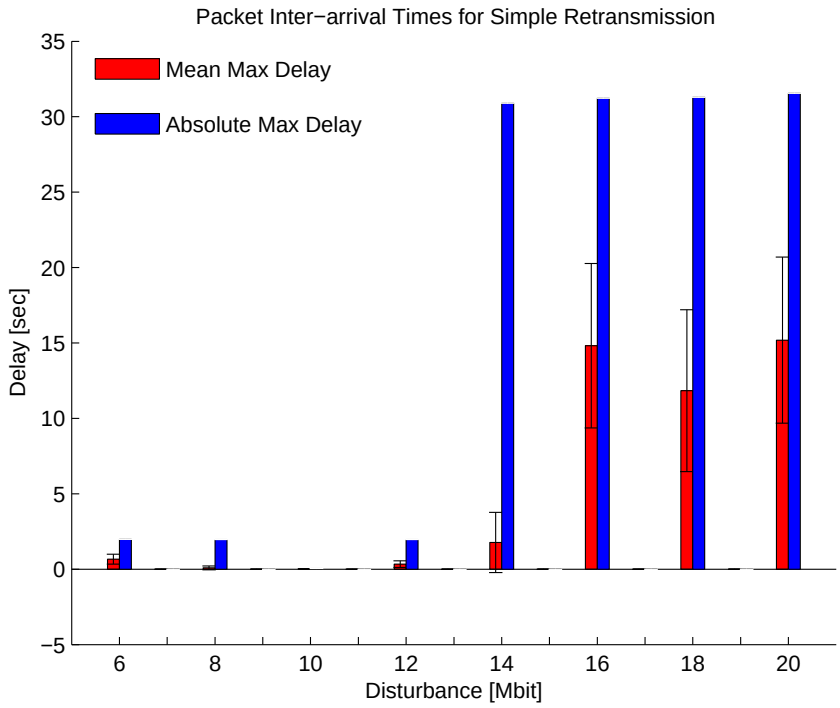

Figure 7.4: The figure shows the maximum inter packet delays of the emulated audio stream packets when using the developed simple retransmission protocol with disturbance rates from 6 to 20Mbit/s UDP traffic in steps of 2Mbit/s. The results are plotted with 95% confidence interval assuming normal distribution.

The mean delay becomes more than 7.4ms when the disturbance becomes 12Mbit/s and above. For 12Mbit/s it is only slightly higher than the expected average of 7.4ms. These results needs to be compared to the packet loss percentage since each lost packet will contribute to the mean inter packet delay. The results indicates that the simple retransmission protocol becomes exhausted around 14Mbit/s since the maximum delay increases a lot and the mean delay also becomes significantly larger.

The amount of inter packet delays exceeding 80ms was also calculated with a 95% confidence interval in order to get a first impression of how many times the skew requirement is not fulfilled. The results for these calculations are shown in Figure 7.5 and Table 7.3.

Disturb. [Mbit/s]	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
Above Threshold [s]	0,0822	0,0658	0,0000	1,2662	37,4198	432,4371	335,1916	399,5971
95% Conf. +/- [s]	0,0423	0,1009	0,0000	0,2461	63,4082	164,1487	150,7542	142,1308

Table 7.3: The table shows the values used to produce the graph in Figure 7.5

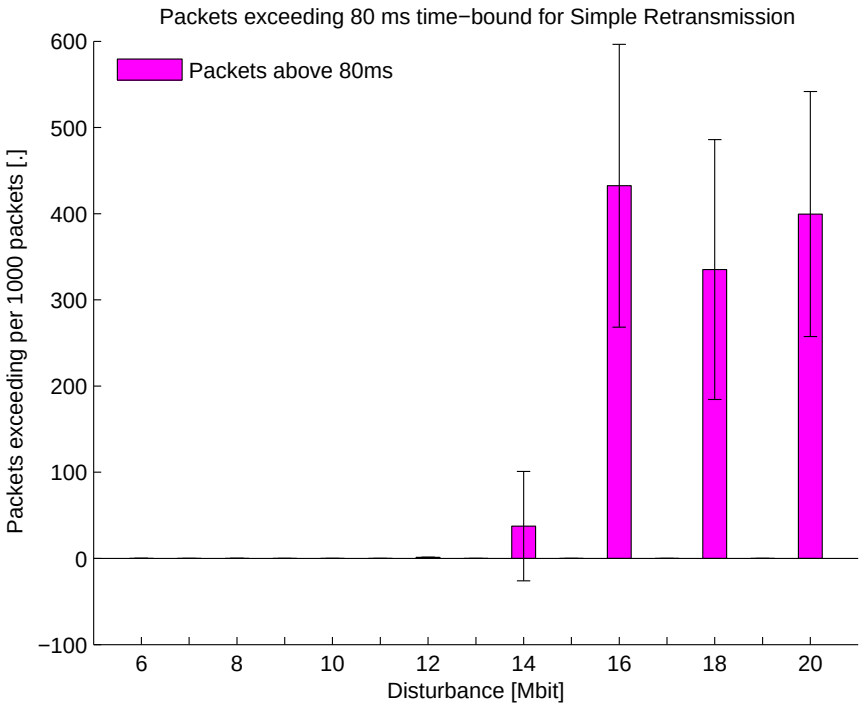

Figure 7.5: The figure shows the mean number of inter packet delays exceeding 80ms with disturbance rates from 6 to 20Mbit/s UDP traffic in steps of 2Mbit/s. When the inter packet delay exceeds 80ms the skew requirement can not be fulfilled. The results are plotted with 95% confidence interval assuming normal distribution.

The results show that at 12Mbit/s disturbance the skew requirement is not fulfilled at least 5 times and at higher disturbances it gets worse.

The buffer calculation, as explained in Chapter 4 on page 28, was also made to determine whether the skew requirement was fulfilled. Figure 7.6 shows an activity diagram of how the calculations was done for the simple retransmission protocol. The input to these calculations is the calculated inter packet delays with packet sequence numbers.

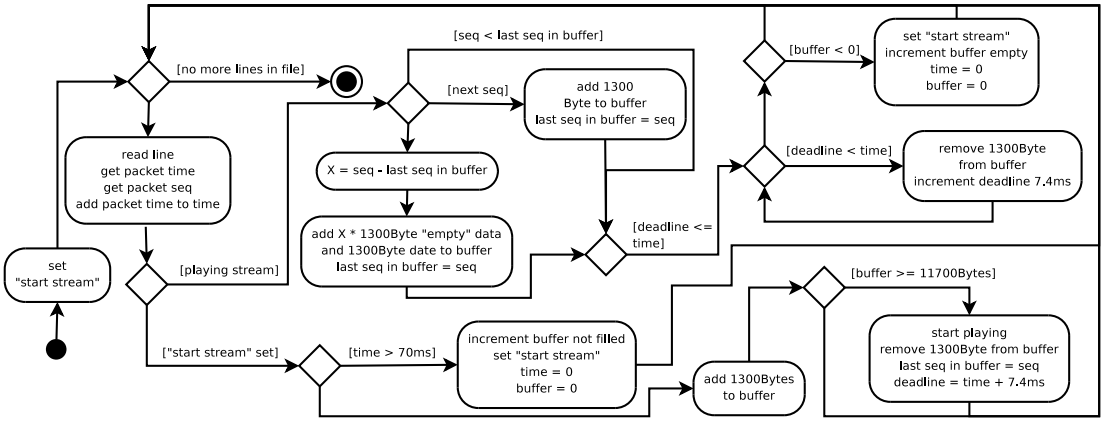

Figure 7.6: The figure shows an activity diagram of how the script to make the buffer calculations works.

The data for a received packet is read from the input file and the inter packet delay is added to the time variable. If the audio stream is starting up 9 packets or 11700Bytes of data has to be in the buffer before 70ms has passed otherwise the stream is restarted. If the buffer becomes full in time the stream can start playing and the first packet is removed from the buffer while the deadline is updated to be 7.4ms later than the time variable. When the stream is playing the sequence number is compared to the last sequence number in the buffer in order to determine if empty packets should be placed in the buffer. If the received sequence number is one higher than the last sequence number in the buffer, 1300Bytes is added to the buffer and the last sequence number in the buffer is incremented. If the received sequence number is smaller than the last sequence number in the buffer, an already added empty packet is replaced by a real packet which means nothing gets added to the buffer. If the received sequence number is higher than the last sequence number plus one, the difference times 1300Bytes plus 1300Bytes is added to the buffer since the gap in sequence number can be filled with empty data and the last sequence number in the buffer is updated. When the data has been added to the buffer the time is checked against the deadline. If the deadline is less than the time, 1300Bytes is removed from the buffer and the deadline is incremented with 7.4ms. This is repeated until the deadline is higher than the current time. After packets have been removed from the buffer to fulfill the deadlines, it is checked whether more data has been taken from the buffer than there was available. If this happens the skew requirement is no longer fulfilled and the stream is restarted with time and deadline values reset.

The results from the calculations are shown in Figure 7.7 and Table 7.4.

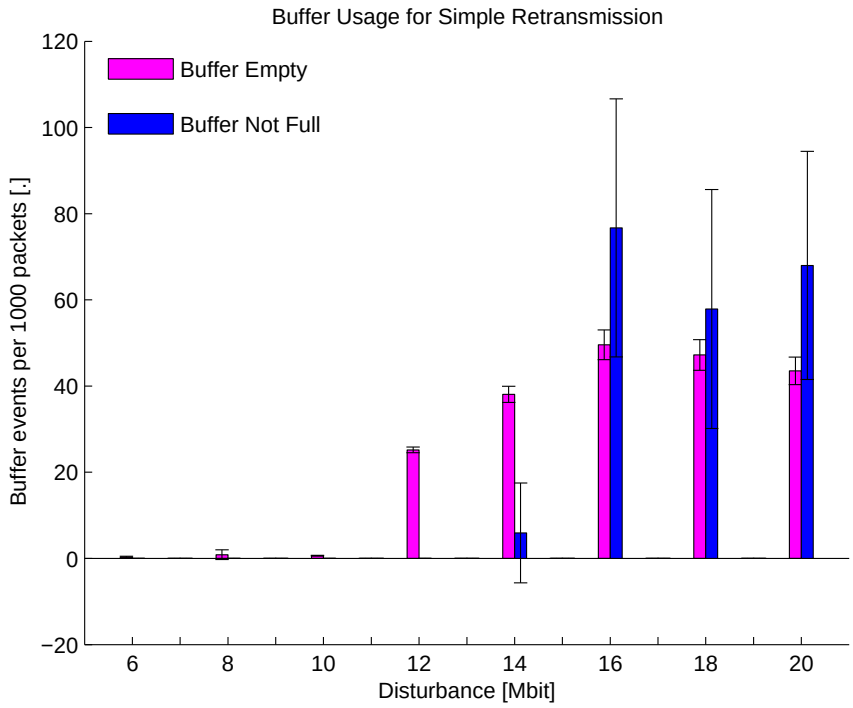

Figure 7.7: The figure shows how many times the simulated application buffer exceeded the limit per 1000 transmitted application packets, which leads to a skew above 80 ms. The results are plotted with 95% confidence interval assuming normal distribution.

Disturb. [Mbit/s]	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
Buffer Empty [.]	0,3371	0,8633	0,6331	25,1850	38,0858	49,5889	47,2126	43,5290
95% Conf. +/- [.]	0,1818	1,1292	0,0721	0,6480	1,8670	3,4514	3,5741	3,1912
Buffer Not Full [.]	0	0	0	0	5,9119	76,7061	57,8770	67,9905
95% Conf. +/- [.]	0	0	0	0	11,5872	29,9509	27,7199	26,4655

Table 7.4: The table shows the values used to produce Figure 7.7

When the disturbance exceeds 10Mbit/s the skew requirement is no longer fulfilled. The number of times the requirement is not fulfilled seems to settle at maximum 50 times per 1000 audio stream application packets. The results also show that for disturbances above 12Mbit/s it gets difficult to even restart the audio stream, since this fails 6 times at 14Mbit/s and more for higher disturbances.

The number of duplicate packets received at the stream client as well as the number of packets received out of order was also calculated. The results are shown in Figure 7.8 and Table 7.5.

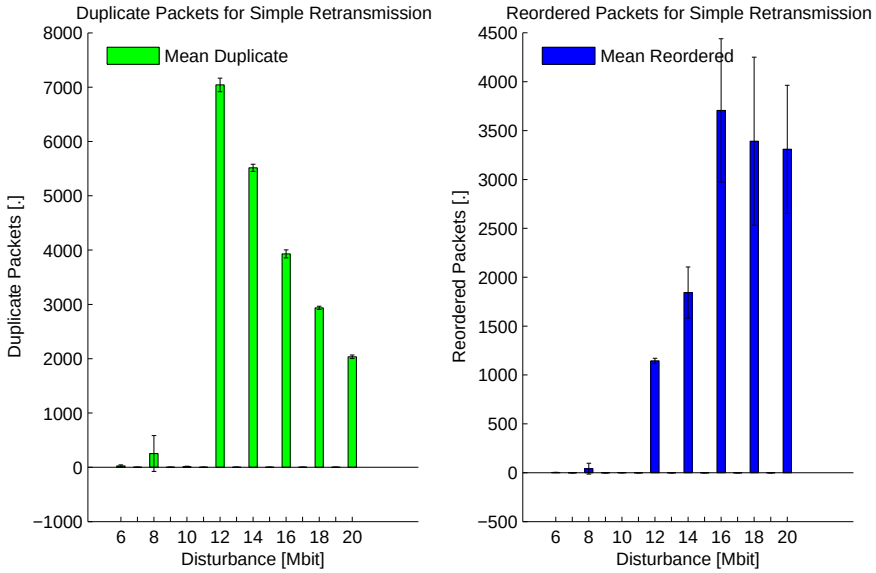

Figure 7.8: The right side of the figure shows the amount of duplicate packets received when using the developed simple retransmission protocol as transport protocol. The left side of the figure shows the amount of reordered packets at varying disturbance rates from 6 to 20 Mbit/s. The results are plotted with 95% confidence interval assuming normal distribution.

Disturb. [Mbit/s]	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
Dup. Packets [.]	26,700	253,20	8,5000	7042,7	5514,8	3930,9	2935,3	2035,3
95% Conf. +/- [.]	16,543	330,05	10,273	124,23	66,428	73,870	29,814	33,895
Reord. Packets [.]	2,2000	41,200	0,5000	1144,8	1843,5	3706,5	3391,3	3309,1
95% Conf. +/- [.]	0,0000	0,0711	0,0000	0,2375	6,9438	4,8596	6,1688	10,732

Table 7.5: The table shows the values used to produce the graphs in Figure 7.8

The number of duplicate packets received gives information about how much network bandwidth is wasted using the simple retransmission protocol, since duplicate packets are packets which are retransmitted even though the packet were already received. The number of packets transmitted to emulate the audio stream was 4055. The results shows that when the simple retransmission protocol starts retransmitting packets ($>=$ 10Mbit/s) almost twice the amount of packets are transmitted as duplicates. The amount of duplicate packets decrease when the disturbance becomes higher which is expected since the packet loss gets higher.

The number of packets which was received out of order (reordered) indicates how many successful retransmissions was done by the simple retransmission protocol. When the disturbance becomes more than 10Mbit/s approximately 25% of the original packets are successfully retransmitted and the number

increases until the disturbance becomes so high that more retransmissions are lost.

The amount of packets which never arrives at the stream client will be the packet loss experienced even with the simple retransmission enabled. This was also calculated and the results are shown in Figure 7.9 and Table 7.6.

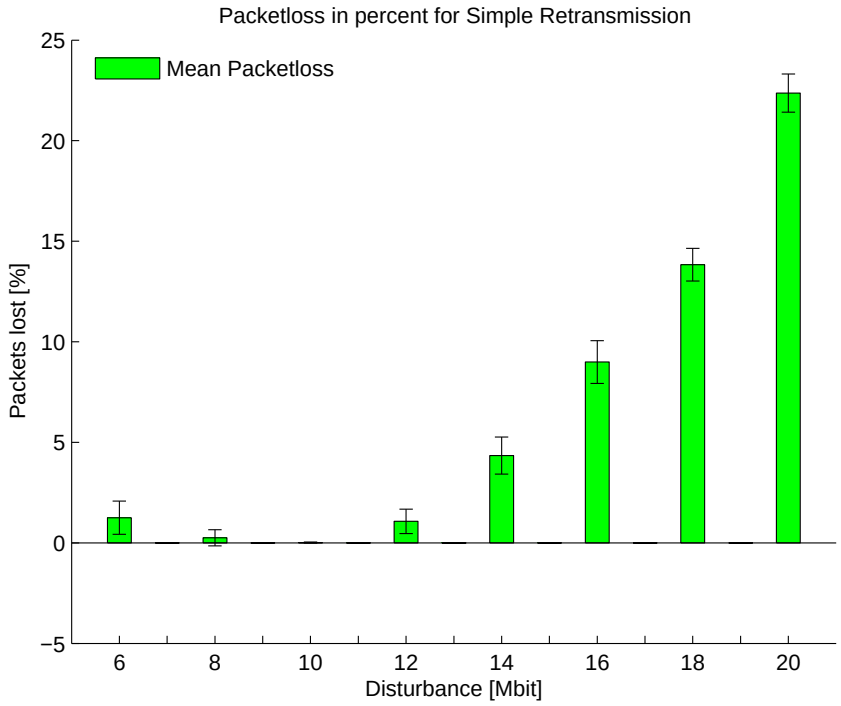

Figure 7.9: The figure shows the mean packet loss for UDP with X Mbit/s UDP traffic as disturbance. The results are plotted with 95% confidence interval assuming normal distribution.

Disturb. [Mbit/s]	**6**	**8**	**10**	**12**	**14**	**16**	**18**	**20**
Lost Packets [%]	1,2503	0,2532	0,0140	1,0727	4,3436	8,9930	13,833	22,36
95% Conf. +/- [%]	0,8249	0,3994	0,0274	0,6071	0,9196	1,0605	0,8140	0,948

Table 7.6: The table shows the values used to produce the graph in Figure 7.9

The results show that for disturbances above 12Mbit/s the packet loss requirement can not be fulfilled. The packet loss is high for 6Mbit/s disturbance compared to the UDP experiments, which probably is due to some WLAN interference in that experiment. When compared to the UDP experiment the simple retransmission protocol performs almost 100% better at 12Mbit/s and considerably more for higher disturbance rates. TCP does not have any packet loss but does not work for higher disturbance rates than 12Mbit/s.

7.4.2 Discussion

The results from the experiments conducted for the simple retransmission protocol was expected to show an increased performance in terms of fulfilling the skew requirement when compared to TCP. TCP performs worse in terms of times where the streams fails a restart (buffer can not be filled), but performs much better in terms of times the stream gets out of sync (buffer becomes empty). The requirement is not fulfilled which means the audio stream use case could not be successfully executed with a disturbance rate higher than 10Mbit/s. Comparing this result to the results for UDP it is clear that disturbance rates above 10Mbit/s is where the 802.11 MAC layer retransmission gets exhausted.

In order to try to maximize the retransmission performance in terms of the skew requirement it was chosen to try to retransmit based on timeouts alone. The results show that the number of duplicates packets received at the stream client is high which means a lot of network bandwidth is wasted. This could be a result of using a too low timeout value, since this would mean that a packet could be retransmitted before the acknowledgement packet was received. The timeout value setting then becomes a tradeof between how much network bandwidth that is wasted and how fast a packet is being retransmitted. It is clear from the bandwidth usage calculations for TCP that not very much extra bandwidth is used for retransmissions. This is also due to the fact that TCP tries to adjust the transmission rate to the capacity in the network.

Another possible improvement to the protocol concerns the reception of acknowledgement packets from the stream client. Since all the protocol and audio stream emulation capability is solved in one large loop it is possible that some acknowledgement packets are not read from the incoming queue fast enough and therefore packets are wrongly retransmitted. The large number of duplicate packets received could be an indication of such a problem, but further experiments would have to be made to determine whether this holds.

It is also possible, that since the simple retransmission protocol is implemented in a high level language like Python and is executed through the Python interpreter, it becomes difficult to keep the timers running fast enough. If the loop does not execute fast enough, the algorithm will not work as expected. It is however not believed to be an issue with the equipment that was used for the experiment, but in order to determine this for sure, further experiments are needed.

The direct comparison between the simple retransmission protocol and TCP is not completely correct since TCP is located at the transport layer and implemented in the operating system kernel in Linux.

This means TCP has timing advantages since it avoids expensive scheduling of processes and other disturbances from the application level.

The simple retransmission protocol will use more bandwidth when packet losses occurs, since it will retransmit packets only based on timeouts. This becomes a tradeof since using more bandwidth for retransmissions will have an impact on the contention at the 802.11 MAC level, which again would cause more packet losses. Since it seems that using more bandwidth for retransmissions doesn't even improve the performance in terms of the skew requirement, it is hardly worth to waste the bandwidth.

7.4.3 Conclusion

The simple retransmission protocol does not perform better than TCP since TCP is better at fulfilling the skew requirement at 12Mbit/s disturbance. This means that the skew requirement still can not be fulfilled at 12Mbit/s disturbance rate, which means it does not seem to be a solution to the packet loss experienced when the 802.11 MAC retransmission mechanism is exhausted. Since the simple retransmission protocol probably can be improved, it is possible that it could be able to fulfill the skew requirement eventually. It is therefore not possible to conclude that a simple retransmission protocol can not be used to improve the performance of the audio stream use case described in this project. It is however believed that it will be difficult to improve the protocol enough to be able to fulfill the requirement, especially at higher disturbance rates than 12Mbit/s.

8 Conclusion

This project considers using Wireless LAN as transport medium for live audio streaming, while keeping the lip synchronization between a television picture and the corresponding audio track. The use-case of the project deals with people working in open office environments, watching the news during the working day. The preliminary analysis investigated requirements for synchronization and packet loss on an audio stream, in order to conclude when the listener finds the skew between audio and video non satisfying.

An initial experiment was conducted, using a stream server and a stream client running Pulse Audio, together with two disturbance nodes. This was done to gain experience with streaming audio over Wireless LAN and UDP multi-casting. Analysis of the network trace was performed using Wireshark, and showed that the packet loss increasing with the disturbance was the primary problem to solve.

A set of packet loss recovering techniques were described as possible solutions to reduce the impact of the packet loss. Two possible directions were found, active and passive recovering techniques. It was chosen to focus on the active sender driven repair of the audio stream, which introduces retransmission of lost segments as the method.

It was not possible to measure the skew of the audio stream directly. Therefore a new method for measuring whether the skew requirement is fulfiled was developed, as a simulation of a fictive buffer at the application layer. It is concluded that measurements of the load of the buffer will give the ability to decide whether the skew requirement of 80 ms is fulfilled or not.

Experiments with adjusting the maximum retransmission limit for 802.11 MAC, showed that MAC-layer retransmission can handle 10 Mbit/s of disturbance traffic traversing the access point, when the MAC retransmission is active. If the MAC layer retransmission is turned off, the performance of the network is degraded remarkable.

A performance evaluation of two transport layer protocols, UDP and TCP, has been performed. It is concluded that UDP protocol exceeds the packet loss requirement of 1,80%, when the disturbance traffic raises above 12 Mbit/s and the MAC retransmission is exhausted. The results shows furthermore, that the UDP protocol keeps an almost constant packet inter arrival time, when the MAC retransmission is active. Overall it is concluded, that UDP can not be used with disturbances above 12 Mbit/s.

In the TCP performance evaluation, the MAC layer retransmission is exhausted at 10 Mbit/s disturbance, and the TCP retransmission starts reacting. It is seen from the results that TCP is able to survive a packet loss of 2% at 12 Mbit/s disturbance, but at 14 Mbit/s disturbance the connection closes down due to timeout. TCP was not able to fulfil the skew requirement for 12 Mbit/s disturbance.

A simple application layer retransmission protocol has been developed, introducing reliability on top

of the UDP protocol, in order to combine the best of the two previously evaluated protocols, with an expectation of a performance gain. The implementation was compared to the results of the TCP and UDP experiments, which showed that the *Simple Retransmission Protocol* implementation performs worse than TCP at fulfilling the requirements at 12 Mbit/s disturbance. It is thereby concluded that the simple retransmission protocol can not fulfill the requirements in its current implementation. There is still room for improvement though, which will be discussed further in the project perspectives.

8.1 Discussion of main Assumption

The disturbance traffic was used to make sure that packets were lost in order to test how the different protocols react to this mechanism. The generated disturbance is however not necessarily a perfect emulation of the disturbance on a WLAN network in general. When the disturbance is created by sending traffic through the same AP as the audio stream, the disturbance will have a high influence on the 802.11 MAC scheduling mechanism while e.g. neighbouring networks on different channels could also cause packet losses, but would not result in more contention on the channel. The chosen disturbance does however, at least partially, emulate the contention which would arise on an WLAN network when a number of unicast streams are transmitted at the same time using transport level retransmission. Since every client will send back acknowledgement packets, the 802.11 MAC contention will become higher and the number of collisions will become higher.

It is believed that the assumption that the disturbance traffic emulates near real behaviour of the disturbance experienced when multiple unicast streams are transmitted at the same WLAN network holds.

8.2 Project Conclusion

It is concluded that 802.11 MAC alone is the best technique for doing the retransmissions, when dealing with delay sensitive unicast streams. The gain in performance from using higher level retransmissions are not significant enough to make a difference. When the disturbance on the wireless channel becomes so high that 802.11 MAC retransmission becomes exhausted, other techniques than retransmissions should be used e.g. using different WLAN channels for different traffic.

9 Project perspectives

This chapter describes the project perspectives. The first sections discuss the future work on the simple retransmission protocol, followed by a section that discuss the possibilities of expanding the project use-case. The last section discuss the possibilities in using the experience gained in this project to try to design an audio stream solution which incorporates state of the art techniques for avoiding packet loss as well as using multicast to make the solution scale better.

9.1 Future Work

In order to be able make a more final conclusion about whether it is possible to construct a simple retransmission protocol which achieves good performance in terms of the skew requirement more work is needed. First it is necessary to conduct experiments to reveal if the timer used to control the retransmission mechanism is able to achieve the required precision. If e.g. there is too much jitter on the detection of timeouts this can have a large impact on the performance of the simple retransmission protocol, and it is possible that this happens since the Python script is run through the Python interpreter as a single operating system process meaning e.g. scheduling events can generate significant jitter in the timer values.

The next possible enhancement concerns the use of acknowledgement packets. The number of duplicate packets received is high which means either acknowledgements are lost, they do not get read in time or the setting of the retransmission timeout value is wrong. Lost acknowledgements is difficult to handle without wasting even more bandwidth, while the other problems could be improved. Conducting experiments to tell where the problem lies specifically would make it possible to adjust the implementation and hopefully make it perform better.

The current simple retransmission protocol only retransmit packets based on timeouts. It is also possible to use the incoming acknowledgment packets do decide whether a retransmission should be performed. If an acknowledgement is received for a sequence number higher than expected, it either means that a packet was lost before reaching the receiver, or that an acknowledgement was lost. It could happen that this next acknowledgement could be received before the timeout value was triggered, making it possible to achieve better performance regarding the skew requirement. The simple retransmission protocol could also be changed to only use the acknowledgements for retransmissions to get and idea of how this strategy would perform.

Since the value of the retransmission timeout could be a reason for worse performance, different strategies for choosing this value could be investigated. Perhaps it would be beneficial if this value was not a fixed value, but could be adjusted according to the ongoing communication. Doing this would increase

the complexity of the protocol considerably though, but it could be worth the trade-off if it yields better performance.

The current implementation of the simple retransmission protocol has been made as an application level retransmission mechanism, which potentially yields poor results regarding the ability to respond quickly to timing incidents. If the implementation was instead done at the transport level, the comparison to TCP would be more fair. This would however loose the gain of having sequence numbers available at the application level, requiring the application to solve this instead.

The 802.11 MAC retransmission was investigated to isolate the performance gains of the different retransmission mechanisms, however more experiments regarding this mechanism would be interesting. Experiments could be made to answer; what happens to the performance of the simple retransmission protocol if 802.11 MAC retransmission is disabled? Is it possible to force the 802.11 MAC retransmission mechanism to do even more retransmission? How would more 802.11 MAC retransmissions affect the performance regarding the skew requirement?

9.2 Expanding the use-case

This project has worked primarily with a unicast stream solution to the specified project use-case, which can work as long as the number of stream clients is not too high. If the project use-case is expanded to cover other areas where streaming audio from a central focus point like e.g. a cinema or a university lecture, the number of clients would becomes too large to be able to support using a unicast solution, without using more WLAN networks. Instead of using unicast streams to the clients, multicast/broadcast streaming could be used as in the initial experiment. Doing this means loosing the 802.11 MAC retransmission which makes transport level and higher level retransmissions more interesting.

The use-case could also be expanded by utilizing more than one network technology for the streaming. In the project use-case only WLAN is used, but by using both WLAN, Bluetooth, LAN etc. a greater flexibilty could be offered to the users of the stream solution. Using e.g. LAN the audio stream could be of higher quality while using Bluetooth could perhaps result in longer battery life on a headset. Using more than one network technology at once sets up a whole new area of possible problems, since the performance requirements for the streaming device becomes higher, and the different technologies could interfere with each other.

Traditional audio in e.g. a cinema can brought to the audience by installing one set of speakers physically in the cinema which then distributes an approximately equal audio experience to the whole audience. New trends regarding cinema audio could make the streaming use-case more interesting, like the use of 3D audio. In order to enable a better 3D audio experience it could be better to have equipment for each individual in the cinema.

9.3 Alternative Approaches

As described in Chapter 2.5 on page 18, retransmissions is not the only method to prevent packet loss. It could be possible to combine different methods of packet loss recovery to perform better than just using retransmission alone.

If retransmission is used in a multicast/broadcast scenario different problems arises, one of them being that retransmitting a packet that perhaps was received correctly by some clients will waste network bandwidth. If different clients are missing different packets, it is possible to use a technique called network coding to construct special packets which can repair multiple packets at different clients. This would make the retransmission much more effective for a multicast/broadcast scenario but will also require more computational performance to construct and use these special network packets.

One interesting aspect that has not been analyzed in this project concerns the relatively new QoS mechanisms standardized for 802.11. These were originally added as the 802.11e amendment but are now a part of the official standard. Using 802.11e it is possible to prioritize different types of traffic in order to make QoS guarantees for traffic which needs this. This could be one way of making the audio streaming perform better when other types of traffic causes problems. 802.11e will not be able to solve the problems arising from multiple stream clients sending back transport level acknowledgements though, since this traffic should have the same priority.

Another approach to enhancing the packet loss problem in the project use-case could be to sample/record and stream both the audio and the video before it is transmitted to the TV. Doing this would make it possible to have a much larger buffer and still fulfill the streaming requirement. It could perhaps even be possible to adjust the speed of the video stream when the audio stream experiences problems. This would however mean that the synchronization between the video and audio stream would have to be controlled and adjusted to make sure the skew requirement is fulfilled.

Bibliography

[1] Gerold Blakowski and Ralf Steinmetz. A media synchronization survey: Reference model, specification, and case studies. *Selected Areas in Communications, IEEE Journal on*, 14:5–35, 1996.

[2] IEEE-SA Standards Board. Ieee standard for information technology-telecommunications and information exchange between systems-local and metropolitan area networks-specific requirements part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications.

[3] Orion Hodson Colin Perkins and Vicky Hardman. A survey of packet loss recovery techniques for streaming audio. *Network, IEEE Journal on*, 12:40 – 48, 1998.

[4] J.G. Gruber and L. Strawczynski. Subjective effects of variable delay and clipping in dynamically managed voice systems. *IEEE Trans. Commun.*, COM 33:801–808, 1985.

[5] N. Jayant and S. Christensen. Effects of packet losses in waveform coded speech and improvements due to an odd-even sample-interpolation procedure. *Communications, IEEE Transactions on*, 29:101– 109, 1981.

[6] H.F. Mattson and G. Solomon. A new treatment of bose-chaudhuri codes. *Journal of the Society for Industrial and Applied Mathematics*, 9:654 – 659, 1961.

[7] G.A. Miller and J.C.R. Licklider. The intelligibility of interrupted speech. *Acoust. Soc. Amer.*, 22:167–173, 1950.

[8] J. Nonnenmacher and L. Strawczynski. Parity-based loss recovery for reliable multicast transmission. *Proceedings of the ACM SIGCOMM*, pages 289 – 300, 1997.

[9] Jen Hay Paul Warren and Brynmor Thomas. *Auditory Perception*. Pergamon Press, 1982.

[10] HIDENETS project. Hidenets tutorial.

[11] I.S. Reed and G. Solomon. Polynomial codes over certain finite fields. *Journal of the Society for Industrial and Applied Mathematics*, 8:300–304, 1960.

[12] J. Rosenberg. Reliability enhancements to nevot. 1996.

[13] Ralf Steinmetz. Human perception of jitter and media synchronization. *Selected Areas in Communications, IEEE Journal on*, 14:61–72, 1996.

[14] Andrew S. Tanenbaum. *Computer Networks 4th Edition*. Pearson Education, 2003.

[15] R. Voelcker. Subjective effects of transmitting speech over packet networks. *Integrated Multiservice Communication Networks, IEE Colloquium on*, 21:3/1 – 3/4, 1985.

AAU Aalborg University

ACK Acknowledgement

AP Access Point

API Application Programming Interface

AR Active Retransmission

BER Bit Error Rate

CBR Constant Bit Rate

DCF Distributed Coordination Function

FCC Federal Communications Commission

FDM Frequency Division Multiplexing

FEC Forward Error Correction

FHSS Frequency Hopping Spread Spectrum

FIFO First In First Out

FLAC Free Lossless Audio Codec

FTP File Transfer Protocol

GFSK Gaussian shaped Frequency Shift Keying

GSM Global System for Mobile communications

HCF Hybrid Coordination Function

HR-DSSS High Rate Direct Sequence Spread Spectrum

IBSS Independent Basic Service Set

IEEE Institute of Electrical and Electronics Engineers

IPC Inter Process Communication

ISM Industrial, Scientific, Medical bands

IP Internet Protocol

LAN Local Area Network

LDU Logical Data Unit

LOS Line of Sight

MAC Medium Access Control

MD Mobile Device

OS Operating System

OSI Open Systems Interconnection

PCC Passive Channel Coding

PCF Point Coordination Function

PCM Pulse Code Modulation

PER Packet Error Rate

PHY Physical Layer

QoS Quality of Service

RSS Received Signal Strength

RTP Real-Time Protocol

RTT Round Trip Time

SCP Secure Copy

STA Station

TCP Transmission Control Protocol

TS Traffic Stream

TTL Time To Live

UDP User Datagram Protocol

UML Unified Modeling Language

USB Universal Serial Bus

VoIP Voice over IP

WAN Wide Area Network

WAV Waveform Audio Format

WLAN Wireless Local Area Network