Design, Implementation and Evaluation of a VoIP Communication System

School of Computer Science, University of East Anglia

CMP-5037B 2016/17

Thomas Pachico (100058436)
Samuel Prebble (100046085)

March 5, 2017

Contents

1 Introduction

The aim of this project was to design and implement a full-duplex Voice over Internet Protocol (VoIP) system that would communicate between two hosts over a single network while minimising delay, providing effective error concealment and a high Quality of Service (QoS). The system follows a layered approach to design and implement a VoIP layer given an audio layer and a transport layer. The audio layer is provided by the *AudioRecorder* and *AudioPlayer* classes and the transport layer uses the *DatagramSocket* class in Java as the User Datagram Protocol (UDP) access point. Additional *DatagramSocket* classes were provided in order to simulate common non-ideal network conditions. The issues, such as corruption, out of order packets, packet loss and standardisation of VoIP packets all had to be handled by the VoIP layer designed and implemented in this project. All of these issues had to be addressed to ensure that the QoS of the designed VoIP system was as high as possible. Certain strategies were used to counter these problems, such as, repetition, filling, block interleaving, sorting and packet checksums.

2 Network Analysis

2.1 DatagramSocket

This represents a socket for sending and receiving datagram packets under conditions where there is no simulated packet loss or any other simulated errors. Every packet being sent and received on this socket is individually addressed and routed. Multiple packets sent from one host to another are not guaranteed to arrive reliably, in order or at all. Even though no simulated problems are presented in this socket, real-time network problems will still affect this socket.

2.2 DatagramSocket2

DatagramSocket2 simulates packet loss across the network by dropping packets in short or long bursts. When this occurs, the receiving side would hear nothing due to the packets containing the voice data getting lost in transit. Packet loss causes some words become broken up and during longer bursts of packet loss some sentences can become illegible thereby significantly impacting the QoS. Short bursts of packet loss can cause noticeable but often intelligible drops in the transmission of voice. Longer bursts of packet loss have both a noticeable and illegible impact on voice transmission, notably worsening the QoS.

2.3 DatagramSocket3

DatagramSocket3 simulates packets being sent and received out of order. For example, packets one through five are received in the order one, two, five, three and four. When this occurs, the voice transmission is illegible due to the fact the voice data is received out of order. If no measures are put in place to combat this then the received voice packets would sound scrambled with words being almost unintelligible. The QoS for *DatagramSocket3* much like *DatagramSocket2* is poor and needs a method of addressing the order of voice data transmission to increase overall QoS.

2.4 DatagramSocket4

DatagramSocket4 causes corruption randomly anywhere in the packet, this could be the payload, header or both. When corruption occurs in the payload the voice data is altered such that the sound emitted is a 'click'. When there is corruption within the header there is a number of problems that could occur upon receiving the packet. Some of these for example could be changing the sequence, transmission type or checksum.

3 Design

3.1 Block Interleaver

In order to combat bursts of packet loss during this project a block interleaver was implemented. The purpose of the block interleaver is to transmit packets in a scrambled order, allowing the packets to be reordered upon reception such that the packets lost in transit are in different positions across the line. The loss of a single packet from an interleaved stream results in multiple small gaps in the reconstructed stream, as opposed to the single large gap which would occur in a non-interleaved stream. This technique reduces the burst length of the packet loss. According to Perkins et. al. (1998) since the loss is spread over smaller parts it becomes easier for listeners to mentally patch over this small loss resulting in a perceived improvement to QoS. To enable the scrambling and reordering of the packets by the block interleaver, a numeric identifier was used for the sequential ordering which the packets had to follow when being reconstructed. More on the implementation of packet ordering is discussed in the Sorting & Sequencing section (3.2) of this document.

The block interleaver implemented works by rotating a NxN matrix as shown in Figure (6.4). Say packet positions 3,4,5 were lost the received packets would be 1,2,3,7,8,9. From our interleaved packets we would receive 1,3,4,6,7,9. This is the first step to resolving the packet loss, from our original packets we would have a long burst of loss, from our interleaved packets we would only have a few short losses. The size of the block interleaver designed and implemented by this project was of size 4 (2x2 matrix). Any perfect square could be used, however, the larger the size of the interleaver, the larger the delay in voice transmission would be.

3.2 Sorting & Sequencing

Since the order between packets in a UDP stream may be unpredictable depending on network paths, buffering and other issues, a sequence identified is generated in the sender thread and attached to the header of the packet as a 4 byte integer. The packet is then constructed to include its header and payload, at which point it is transmitted to the receiver. The receiver then unpacks the received packet and obtains the sequence identifier from the header metadata. The sequence identifier is used to check against other sequence identifiers of received packets and sorted using Java's Collections.Sort method and a custom implementation of a comparator. Having implemented a method of ordering packets enables the system to ensure the audio data is in its correct order. The system can deal with unordered packets either due to being interleaved, or due to network issues.

3.3 Checksum

A checksum is a number representing the sum of all byte values in a piece of stored or transmitted digital data, in this case the packet, against which later comparisons can be made to detect errors in the data. Pre-existing methods of generating checksums were explored, however it was decided that a simpler approach could be taken and as such a custom checksum was built for the purpose of this project. The implemented checksum returns the remainder of the sum of all the bytes in a packet and divided by a set divisor. This remainder would then be added to the packet header and transmitted along with the payload. When the packet is received the checksum method is performed again and compared to the checksum that was sent in the packet header. If the two values are not the same, the packet would would be discarded and considered corrupted. The discarding of the corrupted packet, would then be handled by the strategy currently being used to combat packet loss.

3.4 Strategies

3.4.1 Generic

The Generic strategy designed for this project uses silence substitution to fill the gaps left by a lost packet with silence in order to maintain the correct audio length between the packets sent and received. According to Perkins et. al. (1998) this strategy is only effective when packet audio lengths are shorter

than 4ms and there is a low loss rate (less than two percent). When packet audio lengths increases over 4ms and packet loss is greater than two percent, the QoS will rapidly decrease and is considered by Perkins et. al. (1998) to be unacceptably bad for use in common network audio conferencing tools. Due to its simplicity and ease of implementation it was included in the implementation of this project as a reference point for QoS in comparison to other strategies.

3.4.2 Filling

The Filling strategy in this project takes the previously implemented Generic strategy one step further. Rather than filling packet loss with silence, a pre-recorded sound is used to fill the gaps left by lost packets. In comparison to the Generic strategy Filling is an improvement in terms of QoS. Perkins et. al. (1998) mentions the use of white noise has been shown to give both subjectively better quality and improved intelligibility in comparison to silence. However, words and sentences can still be illegible in places.

3.4.3 Repetition

Repetition is a method used to combat packet loss. Unlike filling it handles packet loss by creating a copy of the last received packet and using that to replace the lost packet. Repetition is best used where there are small bursts of packet loss. Over larger bursts of packet loss, repetition can cause the QoS to deteriorate. According to Perkins et. al. (1998), despite deterioration over long bursts of packet loss using repetition, generally a significant improvement is achieved by using a repetition due to its low computational complexity and its ease of implementation.

4 Evaluation

4.1 PESQ Scores

In order to check if the QoS is improved by the strategies an audio file was recorded of one group member saying the numbers one through ten, three times, totalling a time of 30 seconds. This was then played for each datagram socket on each strategy with interleaving both enabled and disabled. These input and output files were then added into Matlab to determine a PESQ score by running the provided script. The input.wav file was compared to the output.wav file.

Figure (6.5) displays the results from the PESQ scores.

When carrying out the tests an error frequently occurred, shown in Figure (6.6). Despite this error the scores that were generated showed promising results. For example with *DatagramSocket3*, when using the generic strategy with the interleaver on the PESQ score is 0.8656. This low PESQ score is attributed to due to packet loss. Once the repeat strategy was added to combat the packet loss a far greater score of 1.9677 is achieved. The same improvement is shown with *DatagramSocket4,* a score of 1.3059 is given when using the generic strategy, but when using a filling strategy the resulting score improves to 2.0588 and finally when using a repetition strategy the PESQ score improved significantly to 2.8052. Based on these results it is clear that repetition is the best strategy to improve a PESQ score. Unfortunately due to the PESQ error that was encountered it is impossible to fully justify all improvements in all areas of testing.

4.2 Bit Rate

The bit rate is the number of bits transmitted per second.

The first step in calculating the bit rate is to determine how many packets are sent in each second:

Number of packets per second = 1 / duration of packet = 1 / 0.032 = 31.25 packets per second

Next the size of each packet is calculated in bits:

Packet size = payload size + header size = 512 bytes + 6 bytes = 518 bytes = 4,144 bits

Now the bit rate can be calculated:

Bit rate = packets per second x packet size = 31.25 x 4,144 bits = 129,500 bits/second = 129.5kbps

For the system to work, the throughput of the channel must be able to support a bit rate of at least 129.5kbps.

4.3 Delay

Davidson et. al. (2006) defines VoIP delay or latency as the amount of time it takes for speech to exit the speaker's mouth and reach the listener's ear. The main cause of delay in this project is due to the fact that the receiver contains a buffer which waits for four packets to be received, before the voice data is played to the listener. This means there is a minimum delay of 128ms; the duration of four 32ms payloads filling the buffer, which can be considered a type of handling delay. Davidson et. al. (2006) describes handling delay as many different causes of delay (actual packetization, compression, and packet switching) and is caused by devices that forward the frame through the network. Furthermore, we can assume there will be increases in delay caused by computational tasks, propagation delay and serialization delay. Davidson et. al. (2006) notes propagation delays are caused by the length a signal must travel via light in fiber or electrical impulse in copper-based networks. Davidson et. al. (2006) also explains serialization delay is the amount of time it takes to actually place a bit or byte onto an interface and notes its influence on delay is relatively minimal.

5 Conclusion

This project provides a VoIP system which identifies common non-ideal network conditions, and provides the design and implementation for a system which is able to communicate between two hosts over a single network while minimising delay, providing effective error concealment and a high QoS. A minimal delay was demonstrated due to a relatively small buffer size and a 2x2 interleaver. The interleaver performed well in eliminating long bursts of packet loss though improvements could have been made by extending the interleaver size to a 4x4 or even 9x9 matrix size. The increased sizes of the interleaver would have had negative impacts on delay, but would improve the system's ability to handle larger packet loss bursts. Throughout the design it was vital that the system remain as generic as possible. This need for genericism enabled the implementation of a sorting method which was capable of handling not only unordered packets by cause of network issues, but also the reordering of packets which had been interleaved. The best strategy in terms of improving QoS throughout was repetition, this worked well at making a copy of the last received packet and replaying it when a packet was lost.

One of the best elements of the project was the checksum implemented to combat the corruption from *DatagramSocket4*. As shown in the demonstration there was roughly a 1 in 30 chance that a corrupt packet passed through the checksum. While using a checksum such as MD5 may have further reduced the number of corrupt packets which eluded detection, having built a custom checksum provided a much more in depth understanding of the mechanics behind a checksum. There is clearly room for improvement in relation to the checksum, ideally no corrupt packets would be allowed through. Checksums by nature do allow some very specific errors to happen and perhaps the implementation of another error checking layer would have been necessary. The whole system performed well throughout taking into account high error detection and also low delay while still maintaining a good QoS.

References

[1] Unknown
`https://learn.uea.ac.uk/bbcswebdav/pid-1871099-dt-content-rid-2748240_1/courses/CMP-5037B-16-SEM2-B/Lecture%203a%20Calculating%20bit%20rates.pdf`

[2] Colin Perkins, Orion Hodson and Vicky Hardman (1998)
`https://learn.uea.ac.uk/bbcswebdav/pid-1861876-dt-content-rid-2715387_1/courses/CMP-5037B-16-SEM2-B/PerkinsVoIP1998.pdf`

[3] Jonathan Davidson, James Peters, Manoj Bhatia, Satish Kalidindi and Sudipto Mukherjee (2006)
`https://learn.uea.ac.uk/bbcswebdav/pid-1861877-dt-content-rid-2715388_1/courses/CMP-5037B-16-SEM2-B/VoIPinDepth.pdf`

6 Appendix

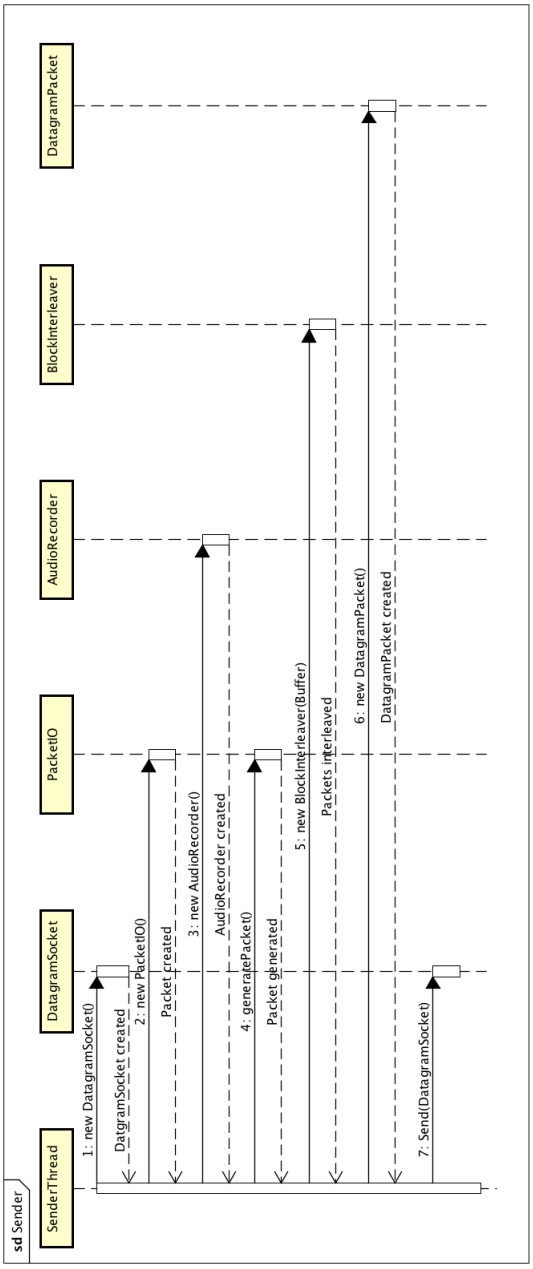

Figure 6.1: Sender Sequence Diagram

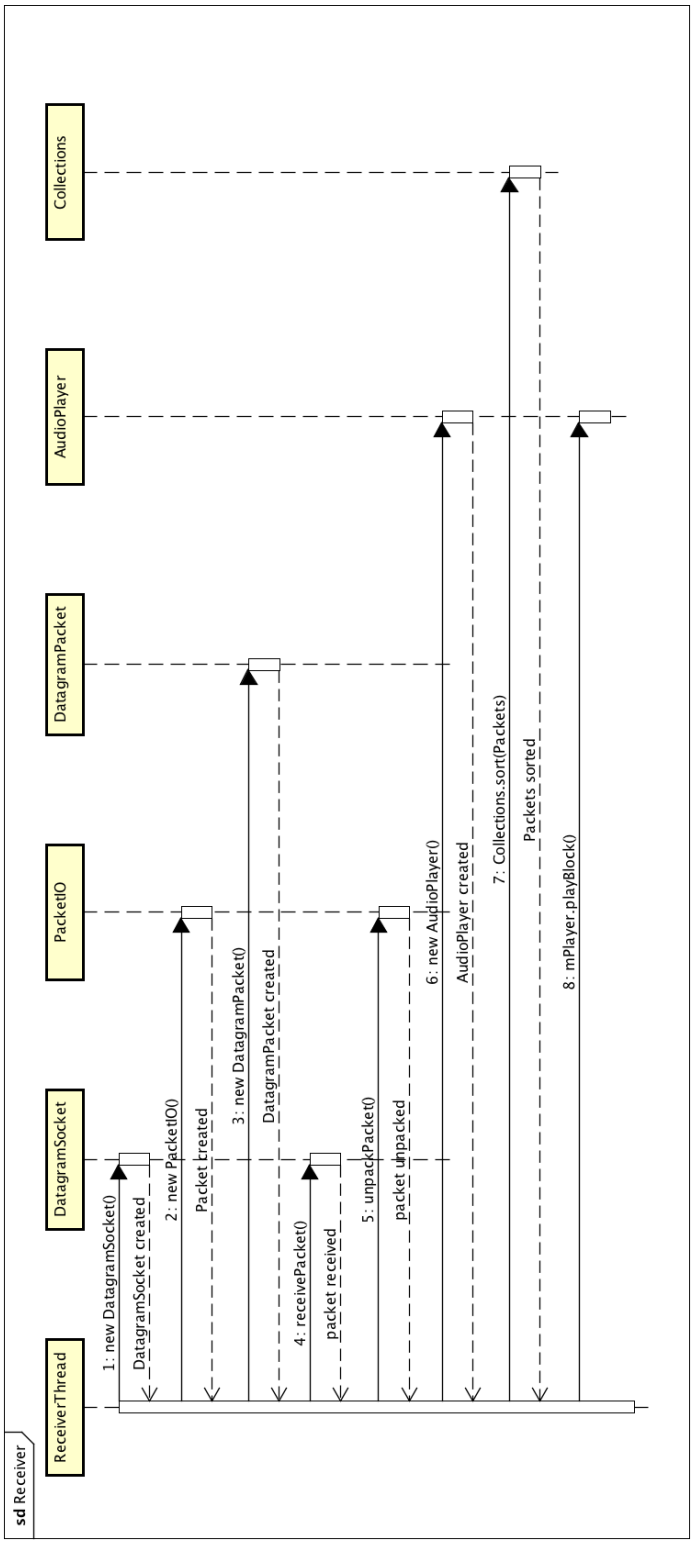

Figure 6.2: Receiver Sequence Diagram

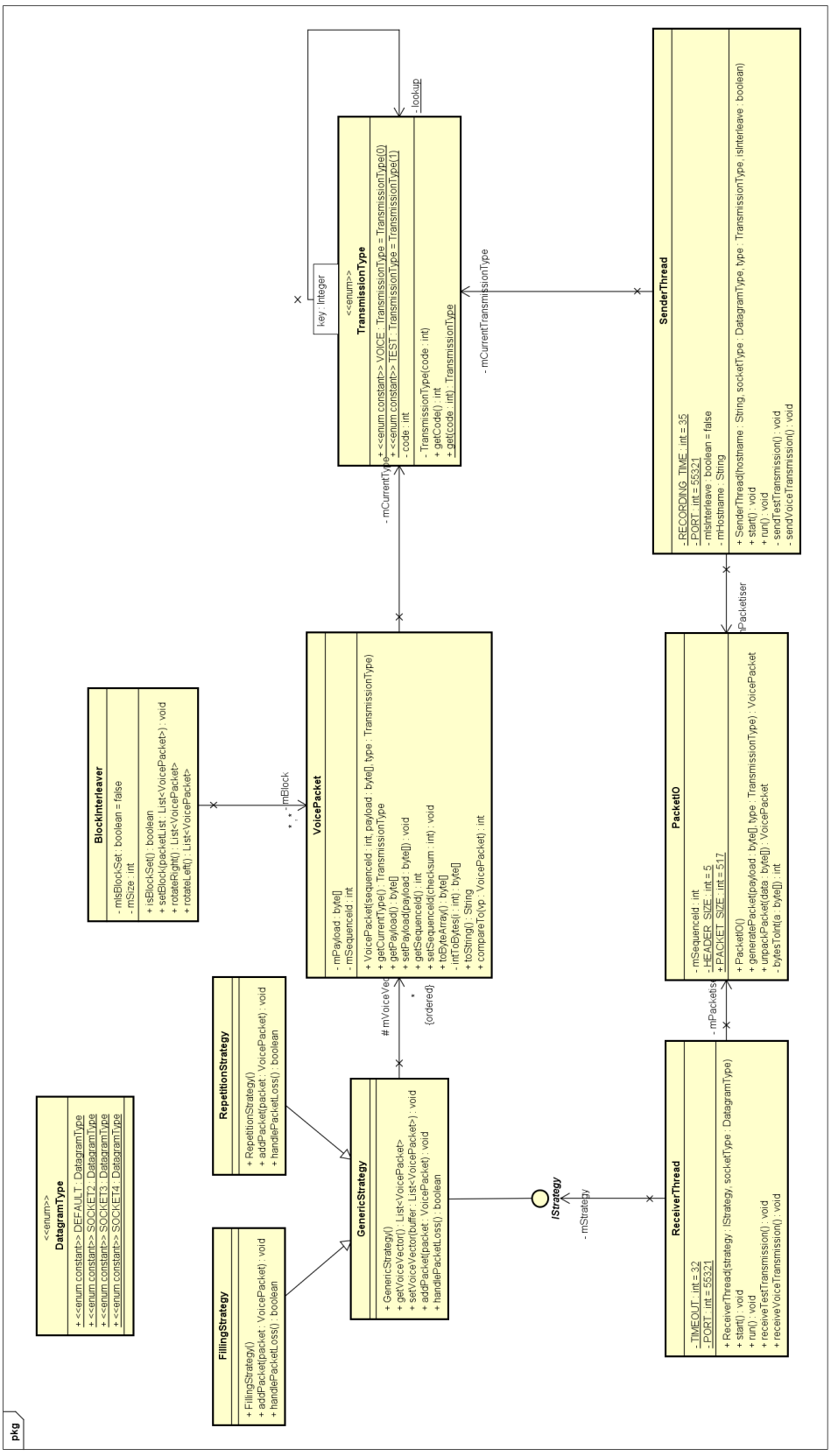

Figure 6.3: Class Diagram

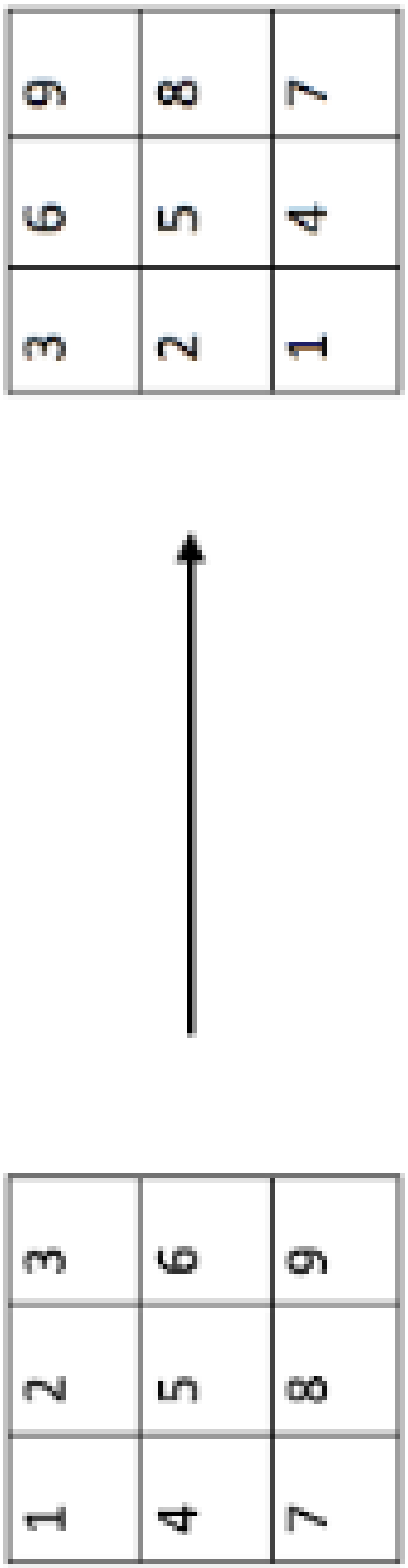

Figure 6.4: Block Interleaver Diagram

Gen = Generic Strategy
Fill = Fill Strategy
Rep = Repetition Strategy
_f = Interleaver Off
_t = Interleaver On

PESQ Scores	gen_f	gen_t	fill_f	fill_t	rep_f	rep_t
DatagramSocket	3.6143	3.6143	Error	3.6532	3.4022	Error
DatagramSocket2	Error	1.0184	Error	Error	Error	Error
DatagramSocket3	Error	0.8656	Error	Error	1.9677	Error
DatagramSocket4	1.3059	Error	2.0588	1.7711	2.8052	Error

Figure 6.5: PESQ Scores

```
Subscript indices must either be real positive integers or logicals.
Error in FFTNXCorr (line 6)
x1= ref_VAD( startr: startr+ nr- 1);
Error in crude_align (line 51)
Y= FFTNXCorr( ref_logVAD, startr, nr, deg_logVAD, startd, nd);
Error in utterance_locate (line 10)
crude_align( ref_logVAD, ref_Nsamples, deg_logVAD, deg_Nsamples, Utt_id);
Error in pesq (line 106)
utterance_locate (ref_data, ref_Nsamples, ref_VAD, ref_logVAD,...
```

Figure 6.6: PESQ Error Message