Introduction to Internet & Security 2021, Project AudioLink

K.Hunziker

15.07.2021

1 Introduction

As part of the lecture Introduction to Internet & Security students are working a regional network called BACNet. The goal is to create a decentralised network similar to Secure Scuttlebutt which is able to operate without the internet. As in the classic OSI model, projects within BACNet can be separated to several abstraction layers. Seeing that users of BACNet should be able to transfer data from one device to another without connecting to the internet, the AudioLink project aims at providing a transport layer implementation to do so. The idea behind this project was to create an python interface which can be used to send and receive data as bytes using sound as the transmission medium. In this report we will first outline the goals and obstacles, then the solutions found to program a working prototype and finally the results and suggestions for further work.

2 Goals & Background

As described above, the goal of this project was to provide a way for BACNet users resp. BACNet applications to transfer arbitrary data between devices using audio. We are aware that in 2020 group 2 (qrLink) started a similar project and they pointed to an already existing python library called quiet (4) which can be used to transmit data as sound. We did, however, neither use nor analyse this library but opted to develop our own system starting from scratch.

The audioLink interface provides methods to calibrate the transmission between two devices and to send and receive bytes of data using a chosen encoding.

3 Implementation

3.1 Prerequisites

In order to develop and use audioLink one needs two devices which can run python scripts. The sender device needs to include or be connected to a speaker which is able to emit the transmission frequencies at a reasonable volume. The receiving device must be able to reliably record sound. The best results where obtained using an external microphone (in our setup we used the AKG C214).

For playing and recording sound from and to a python environment the libraries simpleaudio (5) and sounddevice (6) were used. The sound generation and internal calculations were done using both numpy (num) and scipy (sci), for plotting we included matplotlib and finally for hashing we used hashlib (3) which is already included in python versions 2.6 and above.

3.2 Modulation & Demodulation

This section supplies an overview of the general principles for modulating data which is to be transmitted and demodulating recorded audio signals.

In a first step the data which is to be transmitted is converted from bytes to bits. We then represent every one as a high frequency and every zero as a lower frequency. To do so we must chose τ_s which specifies the number of samples per bit, τ_0 and τ_1 which represent the number of samples per period of the high and the low frequency respectively. For the demodulation to work τ_s must be a multiple of both τ_0 and τ_1.

Demodulation is done by splitting the received signal into segments of τ_s samples and subsequently multiplying each segment with the modulation frequencies and performing discrete time integrals over τ_s. We use the fact that with the chosen relations between τ_s, τ_0 and τ_1 these integrals yield either $\frac{\tau_s}{2}$ or 0.

$$\int_0^{\tau_s} sin^2\left(\frac{2\pi}{\tau_0}t\right)\, dt \; = \frac{\tau_s}{2}$$

$$\int_0^{\tau_s} sin\left(\frac{2\pi}{\tau_0}t\right) sin\left(\frac{2\pi}{\tau_1}t\right)\, dt \; = 0$$

By performing this integration on each bit sized segment (τ_s samples) and using both modulation frequencies we are able determine which frequency was used to modulate a segment. This way we can conclude whether a one or a zero was received.

Evidently, with $r(t)$ being the recorded signal, the demodulation function is given as

$$\left\lceil \frac{\int_0^{\tau_s} sin\left(\frac{2\pi}{\tau_0}t\right) r\left(t\right)\, dt\ - \int_0^{\tau_s} sin\left(\frac{2\pi}{\tau_1}t\right) r\left(t\right)\, dt}{\tau_s} \right\rceil$$

3.3 Real World Problems

The above approach to modulation and demodulation of bit sequences works well in a vacuum resp. if we modulate a sequence, save it to a file and then decode this file. As soon as the modulated signal is actually transmitted through a channel - in this case through a speaker, the air and a microphone - one is faced with a number of physical challenges. The main obstacles will be presented here and the corresponding solutions in the next subsection.

Synchronisation
The recording can start at any point and we must determine where the transmission begins and ends. Also, we must accurately determine where each bit starts in order to slice the recorded signal into vectors of length τ_s.

Channel Imperfections
The transmission channel consists of three physical parts (speaker, room and microphone), all with their own non-linear frequency response. As a result the recorded signal has altered amplitudes, comb-filtering effects, a noise floor and underlying delayed copies of itself caused by reverberation and echo. The further away the microphone is positioned from the speaker, the more these effects come into play as the signal to noise ratio decreases and the reflections from the room become more apparent.

Error Detection
Without additional information in the transmitted data it is not possible for a receiver to establish if an error occurred or whether the data was fully and correctly reconstructed.

Phase Shift
The perhaps best hidden but most disruptive effect was an apparent time dependent phase shift of modulation frequencies. Over the course of a transmission the start of new bits, or at least the phase offset of the frequencies modulating the bits, shifts ever so slightly. This causes issues with the integration as the results approach zero and then turn negative. In 1 the effect of the phase shift on the results of the segment wise integration of a received signal multiplied with the high modulation frequency is clearly visible. In **??** we can observe similar behaviour for the lower modulation frequency. Note, that the zero crossings of the envelope do not occur at the same position. If one considers A to be the amplitude of the received signal and $w(t)$ the room noise, then

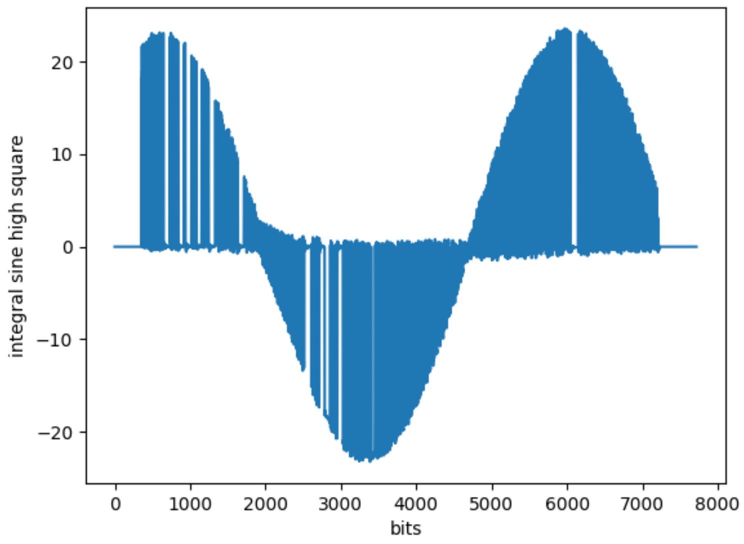

Figure 1: Result of segment wise integration, high modulation frequency

the result of the segment wise integration might be roughly approximated by

$$\int_0^{\tau_s} A\, sin\left(\frac{2\pi}{\tau_0}t\right) sin\left(\frac{2\pi}{\tau_0}t + B\,t\right) + w\,(t)\ dt$$

where B seems to be an unknown constant. Looking at the demodulation function from above, this leads to classification problems whenever one of the integrals approaches zero. This effect was not observable with the initial short test sequences and only comes into play once the transmission exceeds a certain length. Also the causes are not obvious at this point. It could either be a well hidden error in the splitting or demodulation process or the recording could be dropping a sample every now and then.

..

3.4 Real World Solutions

Synchronisation

The main problem of recognising the start and end of a transmission was solved by adding pilot sequences before and after the payload. These bit sequences will not be encoded any further and are known to both sender and receiver. When receiving a signal one can then perform cross-correlation between the signal and a pilot sequence

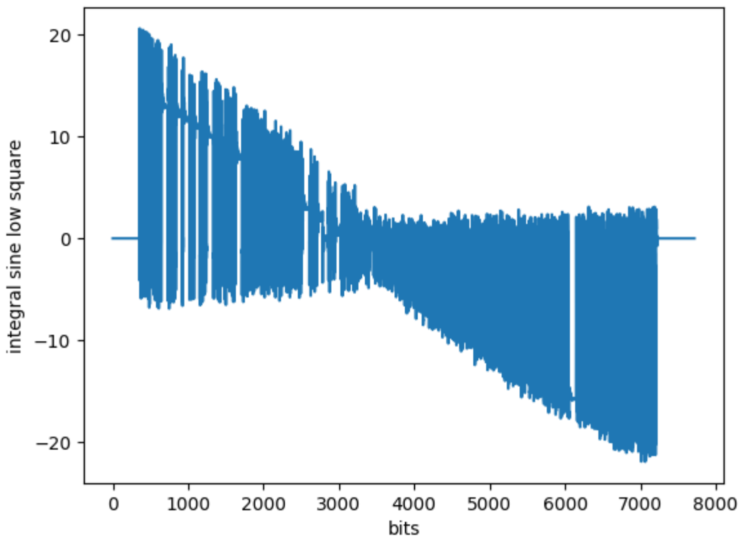

Figure 2: Result of segment wise integration, low modulation frequency

to find the position of the pilot within the recording. Similarly one can can determine where the first one and zero are encoded by cross-correlating the recording with a a modulated one-zero sequence. This position is then used to truncate the recording to a length divisible by τ_s.

Channel Imperfections

The physical properties of the channel can not be circumvented but can be addressed on several layers. By using a proper setup meaning speakers and microphones with approximately linear behaviour and by placing them in close proximity one can already drastically improve the reliability of data transfers. On a software side we added a calibration feature where the sender plays both frequencies used for modulation and the receiver analyses the amplitudes. The amplitude values on the receiving end can then (manually) be fed back to the sender where they are used to adjust the transmission volume. Figures 3 and 4 show the calibration signal recorded by the receiver before and after adjusting the transmission amplitudes. By doing so many side effects such as non-linear speakers and comb-filtering in the room can be mitigated. Further we used encoding on the data before they are modulated. The options are to either encode each bit by repeating it n times or to apply a Hamming (7,4) linear error correction code (7). Both encodings can also be applied in sequence.

Error Detection

Enabling the receiver to detect faulty transmissions was achieved by adding a 32 byte hash value of the entire payload to the end of the data. The used hash function is

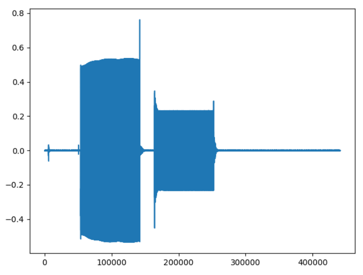
Figure 3: Calibration signal before adjustment

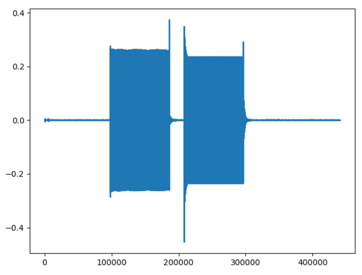
Figure 4: Calibration signal after amplitude adjustment

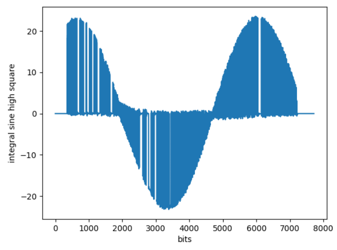

Figure 5: Demodulation of high frequency, normal start

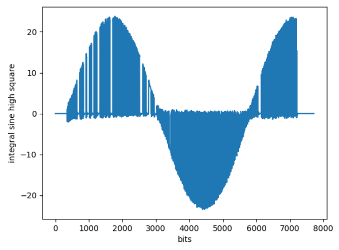

Figure 6: Demodulation of high frequency, offset by 16 samples

SHA-256 implemented in hashlib.

Phase Shift

Whether or not the observed phase shift is a result of an error in the code, we had to find a way to reconstruct the correct data. This was achieved by performing multiple demodulations with offsets varying between 0 and $\frac{T_s}{2}$. Figure 5 shows the results of the integration where the recorded signal starts at the usual point determined by the first detected frequency change. On the other side in 6 the recorded signal was shifted by 16 samples. By adding offsets, the misclassifications are shifted and we can average the classification over several demodulation attempts.

After aggregating demodulations over five iterations, each iteration offsetting the recording by another 16 samples, we obtain results as shown in figure 7. Values above two will be interpreted as ones and values below two as zeros. In practice this approach seems to produce the right results and could be fine tuned to fit other phase shifts, should they occur.

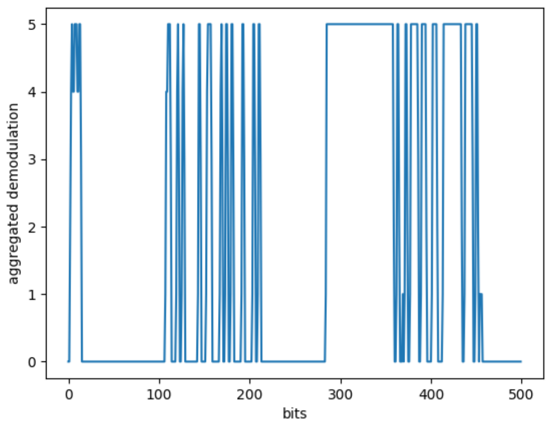

Figure 7: Aggregated demodulation of 500 bits

4 Possible Improvements

There is a number of possible improvements to either increase the efficiency and reliably of the current implementation or even change the modulation and demodulation process entirely to boost the transmission rate. Instead of demodulating the entire signal several times one could split the recording and for every part calculate a phase correction. Some attempts were made to increase the throughput by modulating the first half of the data with two frequencies and the second half with two different frequencies. The two modulated signals would then be superimposed and transmitted. Such a signal should be able to be decoded using the same integration procedure but with four frequencies. Due to time constraints this approach was not pursued further. Finally there are countless options to encode the bits before modulation not limited to repetition and Hamming (7,4).

5 Process

Due to the fact that two out of three group members left the lecture the planning process was difficult and the scope of the project was reduced. As of now the project is not directly integrated into BACNet but it could easily be included in applications to transmit BACNet feeds. The development of a working prototype went rather smoothly and early test results were very promising. Unfortunately testing at an early stage was only done using short bit sequences and the problems arising from the observed phase shift did not manifest. When the phase shift came into effect, a lot of time and energy (approximately three full days) were spent trying to uncover the cause of faulty bits and trying to find ways to enable longer transmissions.

6 Conclusion

Assuming a proper setup and initial calibration, the AudioLink data transmission works well for small amounts of data. We also managed to transfer pictures with sizes up to 2KB without any bit errors. As was to be expected, the throughput is rather low. Using the default settings it takes roughly three minutes to send a 2KB file. Also, if an error was to occur during a transfer, the whole file needs to be retransmitted.

However, a working data transfer method was developed from scratch which is able to synchronise, send and receive arbitrary data and check whether the decoded byte sequence is correct. The development process led to many insights regarding room acoustics, signal processing and analysis as well as a deeper understanding of transmission protocols on the link resp. even the physical layer.

References

[num] numpy. Accessed July 15, 2021, `https://numpy.org/`.

[sci] scipy. Accessed July 15, 2021, `https://www.scipy.org/`.

[3] (2009). hashlib. Accessed July 15, 2021, `https://docs.python.org/3/library/hashlib.html`.

[4] Brian Armstron, Yihui Xiong (2019). quiet.py. Accessed July 15, 2021, `https://pypi.org/project/quiet.py/`.

[5] Joe Hamilton (2019). simpleaudio. Accessed July 15, 2021, `https://pypi.org/project/simpleaudio/`.

[6] Matthias Geier, Spatialaudio (2020). sounddevice. Accessed July 15, 2021, `https://python-sounddevice.readthedocs.io/en/0.4.1/`.

[7] Wikipedia (2020). Hamming (7,4). Accessed July 8, 2021, `https://en.wikipedia.org/wiki/Hamming(7,4)`.