amodem Documentation

Release latest

August 16, 2015

Contents

CHAPTER 1

Description

This program can transmit a file between 2 computers, using a simple headset, allowing true air-gapped communication (via a speaker and a microphone), or an audio cable (for higher transmission speed).

The sender modulates the input data into an audio signal, which is played to the sound card.

The receiver records the audio, and demodulates it back to the original data.

The process requires a single manual calibration step: the transmitter has to find the optimal output volume for its sound card, which will not saturate the receiving microphone and provide good enough Signal-to-Noise ratio for the demodulation to succeed.

CHAPTER 2

Technical Details

The modem is using OFDM over an audio cable with the following parameters:

- Sampling rate: 8/16/32 kHz
- Baud rate: 1 kHz
- Symbol modulation: BPSK, 4-PSK, 16-QAM, 64-QAM, 256-QAM
- Carriers: 2-11 kHz (up to ten carriers)

This way, modem may achieve 80kbps bitrate = 10 kB/s (for best SNR).

A simple CRC-32 checksum is used for data integrity verification on each 250 byte data frame.

CHAPTER 3

Installation

Make sure that `numpy` and `PortAudio v19` packages are installed (on Debian):

```
$ sudo apt-get install python-numpy portaudio19-dev
```

Get the latest released version from PyPI:

```
$ pip install --user amodem
```

Or, try the latest (unstable) development version from GitHub:

```
$ git clone https://github.com/romanz/amodem.git
$ cd amodem
$ pip install --user -e .
```

For graphs and visualization (optional), install *matplotlib* Python package.

For validation, run:

```
$ export BITRATE=48  # explicitly select high MODEM bit rate (assuming good SNR).
$ amodem -h
usage: amodem [-h] {send,recv} ...

Audio OFDM MODEM: 48.0 kb/s (64-QAM x 8 carriers) Fs=32.0 kHz

positional arguments:
  {send,recv}
    send         modulate binary data into audio signal.
    recv         demodulate audio signal into binary data.

optional arguments:
  -h, --help     show this help message and exit
```

CHAPTER 4

Calibration

Connect the audio cable between the sender and the receiver, and run the following scripts:

On the sender's side:

```
~/sender $ export BITRATE=48  # explicitly select high MODEM bit rate (assuming good SNR).
~/sender $ amodem send --calibrate
```

On the receiver's side:

```
~/receiver $ export BITRATE=48  # explicitly select high MODEM bit rate (assuming good SNR).
~/receiver $ amodem recv --calibrate
```

If BITRATE is not set, the MODEM will use 1 kbps settings (single frequency with BPSK modulation).

Change the sender computer's output audio level, until all frequencies are received well:

```
 3000 Hz: good signal
 4000 Hz: good signal
 5000 Hz: good signal
 6000 Hz: good signal
 7000 Hz: good signal
 8000 Hz: good signal
 9000 Hz: good signal
10000 Hz: good signal
```

If the signal is "too weak", increase the sender's output audio level.

If the signal is "too strong", decrease the sender's output audio level.

If the signal is "too noisy", it may be that the noise level is too high or that the analog signal is being distorted. Please run the following command during the calibration session, and send me the resulting `audio.raw` file for debugging:

```
~/receiver $ arecord --format=S16_LE --channels=1 --rate=32000 audio.raw
```

You can see a screencast of the calibration process.

CHAPTER 5

Usage

Prepare the sender (generate a random binary data file to be sent):

```
~/sender $ dd if=/dev/urandom of=data.tx bs=60KB count=1 status=none
~/sender $ sha256sum data.tx
008df57d4f3ed6e7a25d25afd57d04fc73140e8df604685bd34fcab58f5ddc01  data.tx
```

Start the receiver (will wait for the sender to start):

```
~/receiver $ amodem recv -vv -o data.rx
```

Start the sender (will modulate the data and start the transmission):

```
~/sender $ amodem send -vv -i data.tx
```

A similar log should be emitted by the sender:

```
2015-02-06 18:12:46,222 DEBUG      Audio OFDM MODEM: 48.0 kb/s (64-QAM x 8 carriers) Fs=32.0 kHz
2015-02-06 18:12:46,222 INFO       PortAudio V19-devel (built Feb 25 2014 21:09:53) loaded
2015-02-06 18:12:48,297 INFO       Sending 2.150 seconds of training audio
2015-02-06 18:12:48,297 INFO       Starting modulation
2015-02-06 18:12:49,303 DEBUG      Sent      6.000 kB
2015-02-06 18:12:50,296 DEBUG      Sent     12.000 kB
2015-02-06 18:12:51,312 DEBUG      Sent     18.000 kB
2015-02-06 18:12:52,290 DEBUG      Sent     24.000 kB
2015-02-06 18:12:53,299 DEBUG      Sent     30.000 kB
2015-02-06 18:12:54,299 DEBUG      Sent     36.000 kB
2015-02-06 18:12:55,306 DEBUG      Sent     42.000 kB
2015-02-06 18:12:56,296 DEBUG      Sent     48.000 kB
2015-02-06 18:12:57,311 DEBUG      Sent     54.000 kB
2015-02-06 18:12:58,293 DEBUG      Sent     60.000 kB
2015-02-06 18:12:58,514 INFO       Sent 60.000 kB @ 10.201 seconds
2015-02-06 18:12:59,506 DEBUG      Closing input and output
```

A similar log should be emitted by the receiver:

```
2015-02-06 18:12:44,848 DEBUG      Audio OFDM MODEM: 48.0 kb/s (64-QAM x 8 carriers) Fs=32.0 kHz
2015-02-06 18:12:44,849 INFO       PortAudio V19-devel (built Feb 25 2014 21:09:53) loaded
2015-02-06 18:12:44,929 DEBUG      AsyncReader thread started
2015-02-06 18:12:44,930 DEBUG      Skipping 0.100 seconds
2015-02-06 18:12:45,141 INFO       Waiting for carrier tone: 3.0 kHz
2015-02-06 18:12:47,846 INFO       Carrier detected at ~2265.0 ms @ 3.0 kHz
2015-02-06 18:12:47,846 DEBUG      Buffered 1000 ms of audio
2015-02-06 18:12:48,025 DEBUG      Carrier starts at 2264.000 ms
2015-02-06 18:12:48,029 DEBUG      Carrier symbols amplitude : 0.573
```

```
2015-02-06 18:12:48,030 DEBUG      Current phase on carrier: 0.061
2015-02-06 18:12:48,030 DEBUG      Frequency error: -0.009 ppm
2015-02-06 18:12:48,030 DEBUG      Frequency correction: 0.009 ppm
2015-02-06 18:12:48,030 DEBUG      Gain correction: 1.746
2015-02-06 18:12:48,198 DEBUG      Prefix OK
2015-02-06 18:12:48,866 DEBUG        3.0 kHz: SNR = 34.82 dB
2015-02-06 18:12:48,866 DEBUG        4.0 kHz: SNR = 36.39 dB
2015-02-06 18:12:48,867 DEBUG        5.0 kHz: SNR = 37.88 dB
2015-02-06 18:12:48,867 DEBUG        6.0 kHz: SNR = 38.58 dB
2015-02-06 18:12:48,867 DEBUG        7.0 kHz: SNR = 38.86 dB
2015-02-06 18:12:48,867 DEBUG        8.0 kHz: SNR = 38.63 dB
2015-02-06 18:12:48,867 DEBUG        9.0 kHz: SNR = 38.07 dB
2015-02-06 18:12:48,868 DEBUG       10.0 kHz: SNR = 37.22 dB
2015-02-06 18:12:48,869 INFO       Starting demodulation
2015-02-06 18:12:49,689 DEBUG      Got       6.000 kB, SNR: 41.19 dB, drift: -0.01 ppm
2015-02-06 18:12:50,659 DEBUG      Got      12.000 kB, SNR: 41.05 dB, drift: -0.00 ppm
2015-02-06 18:12:51,639 DEBUG      Got      18.000 kB, SNR: 40.96 dB, drift: -0.00 ppm
2015-02-06 18:12:52,610 DEBUG      Got      24.000 kB, SNR: 41.47 dB, drift: -0.01 ppm
2015-02-06 18:12:53,610 DEBUG      Got      30.000 kB, SNR: 41.06 dB, drift: -0.00 ppm
2015-02-06 18:12:54,589 DEBUG      Got      36.000 kB, SNR: 41.37 dB, drift: -0.00 ppm
2015-02-06 18:12:55,679 DEBUG      Got      42.000 kB, SNR: 41.13 dB, drift: -0.00 ppm
2015-02-06 18:12:56,650 DEBUG      Got      48.000 kB, SNR: 41.31 dB, drift: -0.00 ppm
2015-02-06 18:12:57,631 DEBUG      Got      54.000 kB, SNR: 41.23 dB, drift: +0.00 ppm
2015-02-06 18:12:58,605 DEBUG      Got      60.000 kB, SNR: 41.31 dB, drift: +0.00 ppm
2015-02-06 18:12:58,857 DEBUG      EOF frame detected
2015-02-06 18:12:58,857 DEBUG      Demodulated 61.205 kB @ 9.988 seconds (97.9% realtime)
2015-02-06 18:12:58,858 INFO       Received 60.000 kB @ 9.988 seconds = 6.007 kB/s
2015-02-06 18:12:58,876 DEBUG      Closing input and output
2015-02-06 18:12:58,951 DEBUG      AsyncReader thread stopped (read 896000 bytes)
```

After the receiver has finished, verify the received file's hash:

```
~/receiver $ sha256sum data.rx
008df57d4f3ed6e7a25d25afd57d04fc73140e8df604685bd34fcab58f5ddc01  data.rx
```

You can see a screencast of the data transfer process.

CHAPTER 6

Visualization

Make sure that `matplotlib` package is installed, and run (at the receiver side):

```
~/receiver $ amodem recv --plot -o data.rx
```