Analyzing Amazon Sidewalk FSK Protocol

Victor Cai[1,2], Amit Krishnaiyer[1,3], Stefan Gvozdenovic[1], Dr. David Starobinski[1]

Boston University Laboratory of Networking & Information Systems (NISLAB), Electrical and Computer Engineering Department, 8 Saint Mary's Street, Boston, MA 02215[(1)]; Parkland High School, 2700 N Cedar Crest Blvd, Allentown, PA 18104[(2)]; Thomas Jefferson High School for Science and Technology, 6560 Braddock Rd, Alexandria, VA 22312[(3)]

Introduction

Background
- Amazon Sidewalk:
  - Low-bandwidth mesh Internet of Things (IoT) network
  - home automation and security
  - New proprietary FSK (Frequency Shift Keying) communication protocol

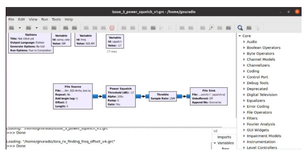
Research Goal
- Dissect and analyze the Amazon Sidewalk FSK Protocol
- Create a framework for analyzing IoT signals in general
- Important for:
  - Analyzing IoT traffic patterns
  - Evaluating IoT security/privacy

Challenges
- IoT protocols proprietary with little documentation publicly available
- Signals are:
  - Sparse/short in time domain
  - Separated on different frequency channels
  - Encoded with variable encoding schemes

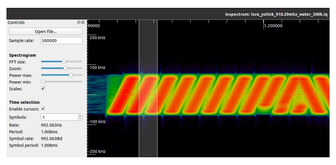
Methodology

This project focuses on building a platform to analyze Software-Defined Radio (SDR)-captured waveforms using the GNU Radio Companion.

Software Setup
- OS: Ubuntu (Linux)
  - Both native Ubuntu and a pre-built Docker file were tested
  - Pre-built Docker image
- Collaboration: GitHub Repository
- Software Tools:
  - GNU Radio: Flowgraph-based programmable radio design

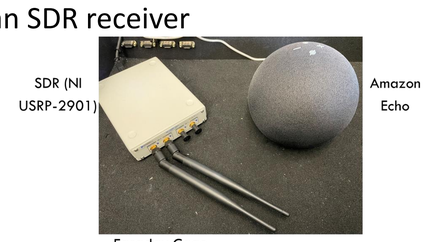


  - Inspectrum: Spectrogram (frequency versus time) for recorded radio signals; analyzes modulation of waveforms; generates decoded bitstream



  - Sublime: Python/C++ editor
  - Universal Radio Hacker (URH): automatic radio signal analysis
  - CRC RevEng: Python-evoked tool for reverse engineering Cyclic Redundancy Check (CRC) error correction codes

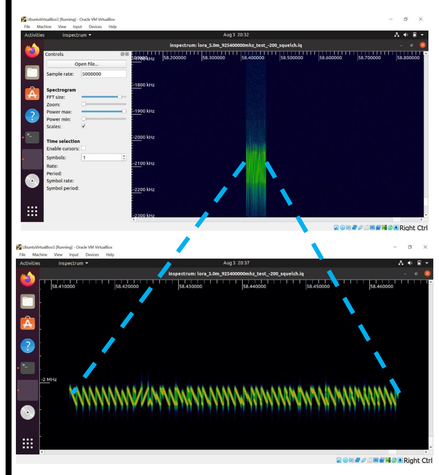
Experimental Setup
- Amazon Echo isolated in a Faraday cage next to an SDR receiver

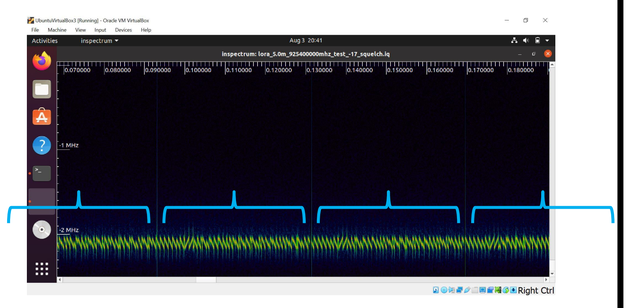


Faraday Cage

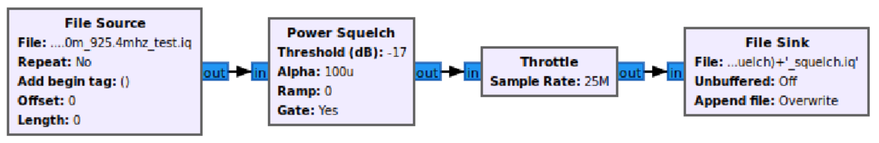
Results

Step 1 Finding Data Packets in Time





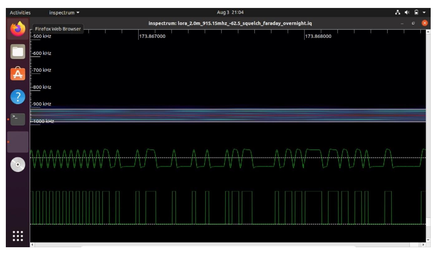
- Received data packets are sparse
- Want packets to be back-to-back in the time dimension
- Input: raw signal capture
- Power Squelch block: threshold on signal energy to detect sections with valid data
- Output: file of back-to-back packets



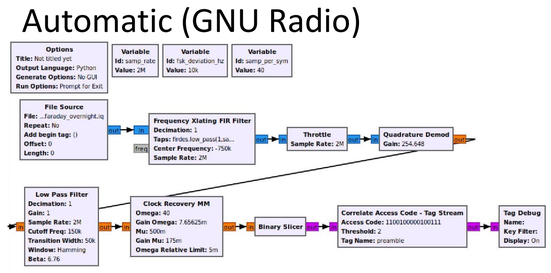
Step 2 Detect packet frequency channel and convert to digital

Part 1: Recovering data bits on known fixed channel
- Shift to known channel and filter
- Latch onto preamble alternating bits as a clock
- Use clock to find bit boundaries and determine if each is a 1 or 0

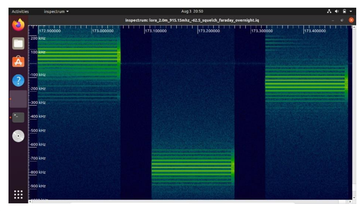
Manual (Inspectrum)

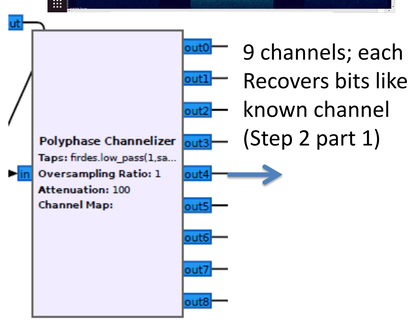


Automatic (GNU Radio)



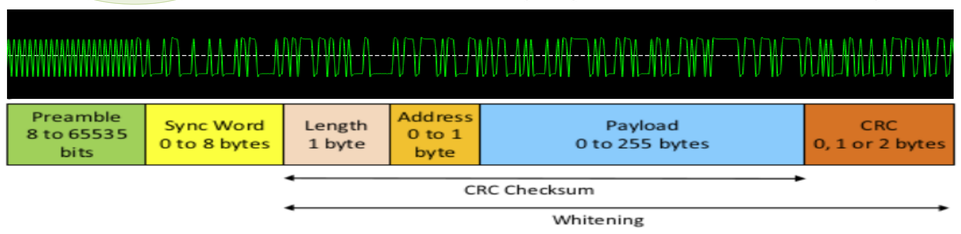
Part 2: Finding the channel
- Input: analog packets from Step 1
- Polyphase Channelizer block:
  - Divides bandwidth into multiple channels
  - Routes inputs to proper channels
  - Replicates bit recovery (part 1) on each channel
- Output: digital binary data





9 channels; each Recovers bits like known channel (Step 2 part 1)

Step 3 Find and extract payload from binary



CRC (Cyclic Redundancy Check) detection:

Python / CRC RevEng:
- Python code loops through possible payload lengths and payload/CRC positions for 4+ different data+CRC strings
- Feeds to CRC RevEng to find CRC that matches
- Also try:
  - Different CRC lengths
  - Invert CRC bits (bitwise XOR)

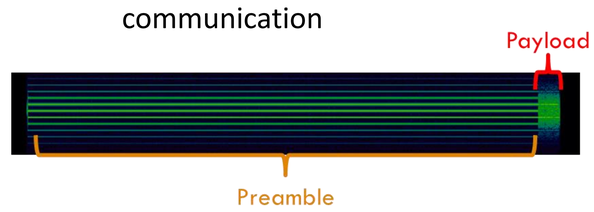
Universal Radio Hacker (URH):
- URH can generate and check CRCs by testing different CRC parameters and other checksums
- Representations such as bit, hex, and ASCII viewable in URH can also be used to help identify CRCs

Discussion/ Conclusions

Accomplishments:
- Captured wireless signals of the Amazon's FSK radio and YoLink's LoRa protocols using SDR and analyzed them in Inspectrum
- Created programmable method of capturing, channelizing, and decoding FSK packets of Amazon Sidewalk
- Demonstrated both manual (Inspectrum + Python) and automatic (GNU Radio / Universal Radio Hacker) methods of processing packets
  - Manual: easier to grasp at first
  - Automatic: better for large-scale operations
- Extracted packet features like center frequency, bandwidth, bitrate, etc.
- Attempted to reverse engineer CRC (cyclic redundancy check) using RevEng tool

Future Research:
- Finding exact CRC algorithm and automatically validating CRC in GNU Radio
- Implementing real-time GNU Radio receiver automating packet processing from input signal to outputted payload data

Discussion:
- Long preamble with respect to packet size
  - Conjecture: reliable long range communication



- Fixed portions across several packets from same device can help with detecting encoding schemes (especially the sync word) and identifying possible sources
- Power-based packet detection was only tested for single channel case without multiple packets coming in simultaneously
- Same setup applicable to other IoT protocols
  - Steps customizable for different signal strengths, channels, and encoding schemes

Acknowledgements

We would like to thank Dr. David Starobinski and Stefan Gvozdenovic for their expertise and guidance. We also thank the support of Boston University and the Laboratory of Networking & Information Systems (NISLAB) for making this collaboration possible.