Reverse Engineering the Positive Train Control (PTC) 220 MHz Wireless Protocol

Dave Twitchell

Shift5 Labs

September 28, 2022
1. Introduction

2. Hardware and Signal RE

3. FEC and Packet RE

4. Demonstration

5. Conclusion
1 Introduction

2 Hardware and Signal RE

3 FEC and Packet RE

4 Demonstration

5 Conclusion
Dave Twitchell (HerrAmeise)
https://github.com/HerrAmeise
https://twitter.com/HerrAmeise
background in mechanical engineering and software RE
~4 years of experience with DSP, SDRs, and GNU Radio
“the onboard data and cybersecurity platform for planes, trains, and tanks”

headquartered in Rosslyn, VA

https://www.shift5.io
Shift5 Labs

• “red team” element of Shift5
• CSRAs (Cyber Survivability Risk Assessments) on OT (Operational Technology) platforms
• independent research (such as this PTC project)
• https://www.shift5.io/labs
What is PTC?

• “Positive Train Control (PTC) systems are designed to prevent train-to-train collisions, over-speed derailments, incursions into established work zones, and movements of trains through switches left in the wrong position.” - FRA website

• Rail Safety Improvement Act of 2008 (RSIA) mandated PTC by 2015

• several extensions pushed compliance date to December 2020

• PTC radio in every locomotive, at certain points along the tracks (“wayside”), and in central offices (“base stations”)

• primary link: 220 MHz (1.25 meter) proprietary protocol

• secondary link: cellular network
Motivation for PTC Research

- PTC security has implications for critical national logistical platforms
- 28% of freight shipped by rail in the United States
- recent reaction to mere *threat* of railroad strikes
- drive cyber awareness in the rail industry
- currently no significant *open* train research in the security community
- it’s fun!
1 Introduction

2 Hardware and Signal RE

3 FEC and Packet RE

4 Demonstration

5 Conclusion
Hardware Acquisition

- monitor eBay for train components
- acquired a locomotive PTC radio and two wayside PTC radios
- had to purchase/fabricate power and data cables
- purchased benchtop power supplies, a networked power switch, and an Intel NUC
- set up a remotely accessible “Trains Lab” in my home office
Open-Source Research

sources:

- general FRA/DOT documents about PTC
- manufacturer FCC filings
- manufacturer patent filings, presentations, field manuals
- no significant community research (unlike EOT, ATCS, etc.)

findings:

- **TX power**: 25-75 Watts
- **modulation**: $\pi/4$ D-QPSK
- **data/symbol rate**: full (32 Kbps), half (16 Kbps)
- **FEC**: Reed-Solomon block codes (*)
- header structure, packet structure, CRC, etc. (*)
## Chart of PTC Radios

<table>
<thead>
<tr>
<th></th>
<th>Wayside Radio</th>
<th>Locomotive Radio</th>
<th>Base Station (24 VDC)</th>
<th>Base Station (48 VDC)</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Rated Power Output</strong></td>
<td>25 W PEP</td>
<td>50 W PEP</td>
<td>75 W PEP</td>
<td>75 W PEP</td>
</tr>
<tr>
<td><strong>Adjustment Range</strong></td>
<td>7.5 - 25 W PEP</td>
<td>15 - 50 W PEP</td>
<td>10 - 75 W PEP</td>
<td>10 - 75 W PEP</td>
</tr>
<tr>
<td><strong>Transmitter Class</strong></td>
<td>Quasi-Linear</td>
<td>Linear</td>
<td>Linear</td>
<td>Linear</td>
</tr>
<tr>
<td><strong>Transmitter Waveforms</strong></td>
<td>16 kbps pi/4 DQPSK</td>
<td>16 kbps, 32 kbps pi/4 DQPSK</td>
<td>16 kbps, 32 kbps pi/4 DQPSK</td>
<td>16 kbps, 32 kbps pi/4 DQPSK</td>
</tr>
<tr>
<td><strong>Transmitter Duty Cycle</strong></td>
<td>10%</td>
<td>30%</td>
<td>50%</td>
<td>50%</td>
</tr>
</tbody>
</table>
Hardware Teardown

220 MHz PTC Radio - Main Board

- D-Code (4-pin) Ethernet - "LAN"
- IEEE 802.3 Ethernet PHY's
- Signal TX/RX (ISO)
- 26 (1x2) 1.27mm header (ColdFire)
- NXP ColdFire MC/54450CR200 microcontroller
- Micron 512 MB LPDDR SDRAM (for ColdFire)
- Local oscillator (cable disconnected)
- Analog Devices AD9910 DDS
- 100 MHz clock
- Altera Cyclone III FPGA
- Altera Max II FPGA
- Spansion (Cypress) GL-128P11F8I2 128 MB FLASH (for ColdFire)
- "DSP JTAG" 14 (7x2) 1.27mm header for TMS320
- Texas Instruments DSP TMS3206741D2KBA3
- MaxLinear (Spex) RS-232 Transceiver
- 3-wire RS-232 header
- 20 (10x2) 2.54mm header labelled "JTAG HEADER"
Telnet Command Interface

- alluded to in patent filings (with some examples)
- not a full shell, but has many commands available
- help text for each command!
- radio also has configuration script stored on SD card
- config script is just a sequential list of these commands executed at boot
## Introduction

Hardware and Signal RE

- FEC and Packet RE
- Demonstration
- Conclusion

### Telnet “Help Menu”

| ADC | DUTYCYCLE | INISTOP | NOTIFICATION | SITE NAME |
| APPS | EASSIGN | INIWRITE | PARSELOG | SNP |
| ASSIGN | EVENT | IP | PING | SSH |
| BASE | FACTORY | IPCONFIG | POS | START |
| BOOT | FANS | KIT | POST | STAT |
| CANMSG | FILES | L1_TEST | RADIOSTATE | STOP |
| CANMSGOFF | FILTER | LED | RANK | SYSMGT |
| CHANNEL | FRAMEDEF | LIMITS | RELAY | T |
| CHECKIN | GUARD | LINKSTAT | REMCMD | TDMA |
| CLASSC | HELP | LOCATION | REMUP | TIME |
| CLASD | HOLDOFF | LOCK | REV | TIMEPROBE |
| CLEARLINKSTAT | HRX | LOGGING | RF | TIMESYNC |
| CLS | HNCONFIG | LOGOFF | RFSURVEY | TRACE |
| COMMISSION | ID | LOGON | ROLE | TRACERT |
| CONFIG | INICHECK | MEM | RXSTAT | TWD |
| CONNECT | INDELETE | MESSAGE | SAVE | TXSTAT |
| CUSTID | INILOCK | MODE | SCHED | TXTEST |
| DATE | INIPRINT | MODFEC | SERIAL | VSWR |
| DD | INIPURGE | MSTSEL | SHC | |
| DEVICE | INIRUN | MSTUP | SHOWDSBR | |
| DHCP | INISELECT | NETWORK | SHOWREMTES | |
Firmware Extraction the “Easy Way”

- firmware blobs stored unencrypted on internal flash memory
- able to copy firmware blobs to SD card via command interface
- some RE required to extract individual firmware images
- able to recover firmware for:
  - TI TMS320 - layer 1 (baseband) processing
  - NXP ColdFire - layer 2/3 processing
- if not possible this way, JTAG headers accessible
- (used JTAG for other RE purposes ...
NXP ColdFire

- General Purpose Processor (GPP)
- instruction set related to Motorola 68000
- loadable in Ghidra with some heavy assistance
- checks payload CRC
- checks SD card config script CRC
- otherwise, not particularly helpful to us in reverse engineering the PHY layer protocol
• Digital Signal Processor (DSP)
• specialized instruction set for DSP
• not loadable in Ghidra/IDA Pro/etc.
• able to linear-sweep disassemble with TI Code Composer Studio tools
• debug symbols not stripped!
• developed some tools to clean up the code, resolve string and references and function calls, etc.
• conclusion: definitely the baseband processor
Strings in TMS320 Firmware

```c
__vtbl__Q2_9payload_n14decoder_conv_c
_decode_Q2_9payload_n14decoder_conv_cSFPSćiQ2_9payload_n10fec_type_ePUc
_prep_viterbi_input_Q2_9payload_n14decoder_conv_cSFPSćiQ2_9payload_n10fec_type_eT1
_process_Q2_9payload_n14decoder_conv_cFPCfIRQ2_9payload_n16decoder_output_c
_deinterleave_Q2_9payload_n14decoder_conv_cSFPSćiT1
_deuncture_threefourths_to_half_Q2_9payload_n14decoder_conv_cSFPSćiT1
_deuncture_seveneighths_to_half_Q2_9payload_n14decoder_conv_cSFPSćiT1
_determine_scale_factor__Q2_9payload_n17bit_buffer_conv_cFf
_process__Q2_9payload_n17bit_buffer_conv_cFPCfl
_alpha__Q3_9payload_n17bit_buffer_conv_cSiir_c
_scale_current_window__Q2_9payload_n17bit_buffer_conv_cFf
_encode__22conv_payload_encoder_cSFPUciQ2_9payload_n16fec_type_eT1
_puncture_and_interleave__22conv_payload_encoder_cSFPUciQ2_9payload_n10fec_type_eT1
_puncture_onethird_to_seveneighths__22conv_payload_encoder_cSFPUciT1
_interleave__22conv_payload_encoder_cSFPUciT1
_puncture_onethird_to_half__22conv_payload_encoder_cSFPUciT1
_puncture_onethird_to_threefourths__22conv_payload_encoder_cSFPUciT1
_convert_db_to_linear__Fs
_header_v2__conv_1_2_uncoded_bits__Q2_5ber_n12calculator_c
_header_v2__conv_3_4_uncoded_bits__Q2_5ber_n12calculator_c
_process_raw__conv_payload__Q2_5ber_n12calculator_cFPCScRQ2_5ber_n16payload_output_c
_header_v2__conv_7_8_uncoded_bits__Q2_5ber_n12calculator_c
```
PN9 Test Mode

- PN9 = pseudorandom sequence
- transmit stream of Linear Feedback Shift Register (LFSR) bits
- demodulated signal with Signal Hound (Spike software)
- used Berlekamp-Massey algorithm to confirm LFSR polynomial
- helped to determine the correct D-QPSK symbol mapping

```
$ ./berlekamp.py
The input sequence is (1, 0, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 0, 0, 0, 0, 0, 1, 0, 1, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 0, 1, 1).
Its characteristic polynomial is (x^9 + x^4 + 1),
and linear span is 9.
```
Demodulating with Signal Hound (Spike)
Random Packet Test Mode

- normal header and payload of random bytes
- half or full data rate
- five FEC modes: NOFEC, 16RS12, CONV12, CONV34, CONV78
- debug trace mode on radio prints packet payloads before FEC!
- no mention of convolutional coding in patent/FCC filings ... 
- strings in TMS320 firmware confirmed convolutional coding 
- captured some packets in NOFEC mode for analysis in Python
Phase Shift Keying (PSK)

- data modulated AM, FM, PM, or combination
- PSK - phase shift keying, a digital modulation
- BPSK (binary) - two symbols, one bit/symbol
- QPSK (quadrature) - four symbols, two bits/symbol
- 8-PSK - eight symbols, three bits/symbol
QPSK (as Real Signal)
QPSK and 8-PSK (as Complex Signal)
Differential QPSK (D-QPSK)

The symbol of S(0-1) is -3π/4
The bits of S(0-1) are "11"
Differential QPSK (D-QPSK)

The symbol of S(1-2) is π/4
The bits of S(1-2) are "00"
Python Analysis - Constellation

![Constellation Diagram]
Python Analysis - Phase Changes

![Phase Differences](chart.png)

Dave Twitchell
Shift5 Labs
Reverse Engineering the Positive Train Control (PTC) 220 MHz Wireless Protocol
Preamble

- preamble marks start of the packet
- assists with time-domain synchronization
- preamble shown in patent filings
- two preamble versions in TMS320 firmware
- modulate preamble to D-QPSK and correlate with received signal
Python Analysis - Preamble Alignment
Synchronization and Corrections

- Sample Timing Offset (STO)
  - solution: oversampling the signal (16x)
  - use known preamble to determine ideal sampling time
- Carrier Frequency Offset (CFO)
  - CFO causes phase rotation
  - use preamble to calculate channel estimate and correct CFO
  - convert phase differences (differential symbols) back to Cartesian format for better visualization
Python Analysis - Preamble With CFO
Python Analysis - Corrected Constellation

![Constellation (Original)](chart1.png)

![Constellation (Corrected)](chart2.png)
Introduction

Hardware and Signal RE

FEC and Packet RE

Demonstration

Conclusion

GNU Radio Demodulator Flowgraph Part 1

Dave Twitchell

Reverse Engineering the Positive Train Control (PTC) 220 MHz Wireless Protocol

Shift5 Labs

36 / 64
GNU Radio Demodulator

Reverse Engineering the Positive Train Control (PTC) 220 MHz Wireless Protocol
1. Introduction

2. Hardware and Signal RE

3. FEC and Packet RE

4. Demonstration

5. Conclusion
The Road to Payload Decoding

- in NOFEC mode, received (RX) bytes same as transmitted (TX) bytes
- RX header always fixed size of 6 bytes (48 bits)
- no discernible length field in RX header
- header fields printed on radio, but not serialized header
- using other FEC modes, received (encoded) data size was proportional to transmitted (un-encoded) data size, as expected
- in 16RS12 mode, can see interleaved TX bytes
- in CONVXX mode, RX bytes are completely different from TX bytes
TX/RX Block Diagrams

Data → Segmentation → FEC Encoding → Interleaving → Modulation → Filtering (Pulse Shaping)

Data ← Concatenation ← FEC Decoding ← Deinterleaving ← Demodulation ← Filtering (Pulse Shaping)
TX/RX Block Diagrams (NOFEC)
Reed-Solomon (RS) Interleaver

- segmentation and interleaving partially described in patent
- double-nested block interleaver ("blocks" and "super blocks")
- FEC retains original bytes
- original TX bytes visible in RX data *before deinterleaving and FEC decoding*
- observed patterns on 6-, 12-, and 72-byte boundaries
- some edge cases with shorter packets, zero padding, etc.
- mostly data analysis; some firmware assistance to confirm edge cases
- result: deinterleaved, FEC-encoded data
Reed-Solomon (RS) FEC

- after deinterleaving, left with 16-byte Reed-Solomon codewords
- RS(16,12) = 12 bytes data, 4 bytes parity
- found log/antilog tables in TMS320 firmware
- confirmed by just brute forcing polynomial space
- common primitive polynomial: 0x11D
Convolutional Coding (CC) Interleaver

- dynamic interleaver (changes based on size of input data)
- stream interleaver (vs. block interleaver with RS)
- operates at the bit level (vs. byte level with RS)
- painful hand analysis of TMS320 firmware ...
CC Interleaver in TMS320 Firmware

```
1180a174  MV.L2           B3,B1
1180a176   MV.L1X          B4,A7
1180a178   CALLP.S2  _local_call_stub,B3
1180a17a   MV.L2X          A4,B4
1180a17c   fhead
1180a180   MKV.D2          0,B2
1180a184   INTDP.L1         A7,A5:A4
1180a188   MKV.S1           0xffffffff,A17
1180a18c   MKV.S1           0xffffffff9c47,A16
1180a190   MKVH.S1          0x3fe000,A17
1180a194   MKVH.S1          0xfaa10000,A16
1180a198   ADDDP.L1         A17:A16,A5:A4,A5:A4
1180a19c   MV.S1X          B4,A9
1180a19e   MV.L1           A6,A8
1180a1a0   NOP              4
1180a1a2   NOP              3
1180a1a4   CMP.EQ           0,A2,A0
1180a1a6   CMP.EQ           0,A2,A0
1180a1a8   CMP.EQ           0,A2,A0
1180a1ac  [!A0]           BNOP.S1   DONE,5

1180a1b0   MPY32.M1        A2,A2,A0
1180a1b4   MV.L1X          B2,A4
1180a1b8   MV.L2X          A2,B0
1180a1bc   .fhead
1180a1c0   SUB.L1          A0,0x1,A16

1180a1c4  OUTER_START:
1180a1c4   MV.L1X          B2,A1
1180a1c6   MV.S1           A16,A3

1180a1c8  INNER_START:
1180a1c8   ADD.L1          A6,A1,A5
1180a1ca   CMP.L.M1        A5,A7,A0
```

Dave Twitchell
Shift5 Labs
Reverse Engineering the Positive Train Control (PTC) 220 MHz Wireless Protocol
Reverse Engineering the Positive Train Control (PTC) 220 MHz Wireless Protocol

Dave Twitchell
Shift5 Labs
CC Interleaver in C++

```c
unsigned int* interleaver = new unsigned int[size];

int A0, A1, A2, A3, A4, A5, A6, A7, A16;
int B0, B2;

B2 = 0;
A7 = size;
double A5A4 = sqrt((double)A7);
double A17A16;
uint64_t adder uint64 = 0X3FEFFFFFFFA19C47;
memcpy((void*)&A17A16, (void*)&adder_uint64, sizeof(uint64_t));
A5A4 = A17A16 + A5A4;

unsigned int* A9 = indices;
unsigned int* A8 = interleaver;

A2 = (int)floor(A5A4);
if (!(0 < A2) == false) goto DONE;
A0 = A2 * A2;
A4 = B2;
B0 = A2;
A6 = B2;
A16 = A0 - 1;

OUTER_START:
  A1 = B2;
  A3 = A16;

INNER_START:
  A5 = A6 + A1;
  if (!(A5 < A7) == false) goto CHECK;

//COPY:
  A8[A4] = A9[A5];
```

Dave Twitchell
Shift5 Labs
Reverse Engineering the Positive Train Control (PTC) 220 MHz Wireless Protocol
Overview of Convolutional Coding

- based on LFSR
- data bits advance through LFSR and generate parity bits
- non-recursive vs. recursive
- flush (non-tail-biting) vs. tail-biting
- puncturing
PTC Convolutional Coding

- heavily assisted by TMS320 firmware RE
- always generated at 1/3 base rate
- punctured to 1/2, 3/4, or 7/8 rate
- problem: don’t know LFSR polynomials (taps)
- base 1/3 rate encoder function in TMS320 very complicated (392 lines of disassembly!)
- hand analysis too error-prone
- not worth the investment to write a more complex analysis framework (recursive descent parser, CFG generator, Ghidra module, etc.)
PTC Convolutional Coding

SWITCH TO SPREADSHEET
The Plan

PHASE 1  PHASE 2  PHASE 3

Collect convolutional encoder taps  Profit

Dave Twitchell  
Reverse Engineering the Positive Train Control (PTC) 220 MHz Wireless Protocol
The Real Plan

- radio board has header labeled “DSP JTAG” near TMS320
- look up pinout and probe with oscilloscope
- purchase JTAG debugger and use with TI Code Composer Studio
- set breakpoint before base convolutional coder function
- zero out input bits except for first bit
- set another breakpoint after coder has run to completion
- dump memory of convolutionally encoded bits
JTAG Hookup
### Calculation of CC Taps

<table>
<thead>
<tr>
<th>HEX</th>
<th>DEC</th>
<th>OCT</th>
</tr>
</thead>
<tbody>
<tr>
<td>0x5B</td>
<td>91</td>
<td>0133</td>
</tr>
<tr>
<td>0x79</td>
<td>121</td>
<td>0171</td>
</tr>
<tr>
<td>0x65</td>
<td>101</td>
<td>065</td>
</tr>
</tbody>
</table>
CC in C++

```c
struct lte_conv_code code;
memset((void*)&code, 0x00, sizeof(struct lte_conv_code));

code.n = 3;
code.k = 7;
code.len = data_size_bits;
code.gen[0] = 0x5B;
code.gen[1] = 0x79;
code.gen[2] = 0x65;
code.term = CONV_TERM_FLUSH;

unsigned int full_rate_size = data_size_term * 3;
```
Packet Header and CRCs

- header is convolutionally encoded at 1/2 rate, tail-biting
- header has three fields: FEC type enum, payload length, and packet type
- payload has a CRC-16 (not obvious in NOFEC random packets)
- used TMS320 firmware to RE a 3-bit header checksum
- used ColdFire firmware to RE the payload CRC
1 Introduction

2 Hardware and Signal RE

3 FEC and Packet RE

4 Demonstration

5 Conclusion
Demo Setup

13.6V Power Supply

PTC Radio

10 dB attenu.

RF (coax)

Network Switch

Ethernet

Ethernet

USRP B200

USB

Laptop (GNU Radio)

Demo Setup

Dave Twitchell

Reverse Engineering the Positive Train Control (PTC) 220 MHz Wireless Protocol
Code Release!

- releasing full PTC receive chain after talk
- https://github.com/shift5research/ptc-toolkit
- signal processing library: ptcrx
- GNU Radio 3.9 OOT module: gr-ptc
- library of “random packet” IQ files with all rates/FEC
- RX flowgraph that accepts both hardware (USRP B2XX) and file input
Software Used

- https://github.com/gnuradio/gnuradio
- https://github.com/gnuradio/volk
- https://github.com/EttusResearch/uhd
- https://github.com/quiet/libcorrect
- https://github.com/ttsou/turbofec
- https://github.com/sandialabs/gr-pdu_utils
- https://github.com/skysafe/gr-sigmf
- https://signalhound.com/spike/
- https://www.ti.com/tool/CCSTUDIO
Further Work

- improve burst/energy detection - not based on absolute threshold
- improve signal corrections
- soft demodulation
- pregenerate matrices/interleavers
- wideband channelizing demod
- investigate layer 2/3 of PTC protocol (packet types, TDD/FDD schedules, etc.)
Questions?

- Dave Twitchell
- dave.twitchell@shift5.io
- https://github.com/shift5research
- https://shift5.io/labs