Reverse Engineering Outernet

Daniel Estévez

27 December 2016
33rd Chaos Communication Congress, Hamburg

Brief info about speaker

- Currently finishing a PhD in pure Mathematics
- Also have a background in Computer Science
- Amateur Radio operator since 2.5 years. Callsign: EA4GPZ / M0HXM
- Started decoding Amateur satellites a year ago.

Outline

Outline

What is Outernet?

- Startup company with goal of easing worldwide Internet access by broadcasting content from satellites
- Started broadcasting on Ku-band DTH satellites using DVB-S
- Ku-band no longer in use
- Currently broadcasts on L-band (around 1.5GHz) through 3 Inmarsat satellites (Americas, Europe/Africa, Asia/Pacific)
- Almost worldwide coverage

Receiving equipment

- **Hardware:**
  - L-band antenna. Tipically a patch antenna, can also use a dish
  - LNA (preamplifier)
  - RTL-SDR dongle
  - Single board ARM computer: C.H.I.P. or Raspberry Pi 3
  - Outernet sells a kit with these items
- **Software:**
  - rxOS: A Linux image for ARM that does everything for you
  - Most of the software is open-source
  - Key parts of the receiver are closed-source binary only
  - GPL libraries (`librtlsdr` and `libmirisdr`) are used in the closed-source receiver. Possible GPL violation
  - The protocols, modulation and format of the signal are secret

Receiving equipment

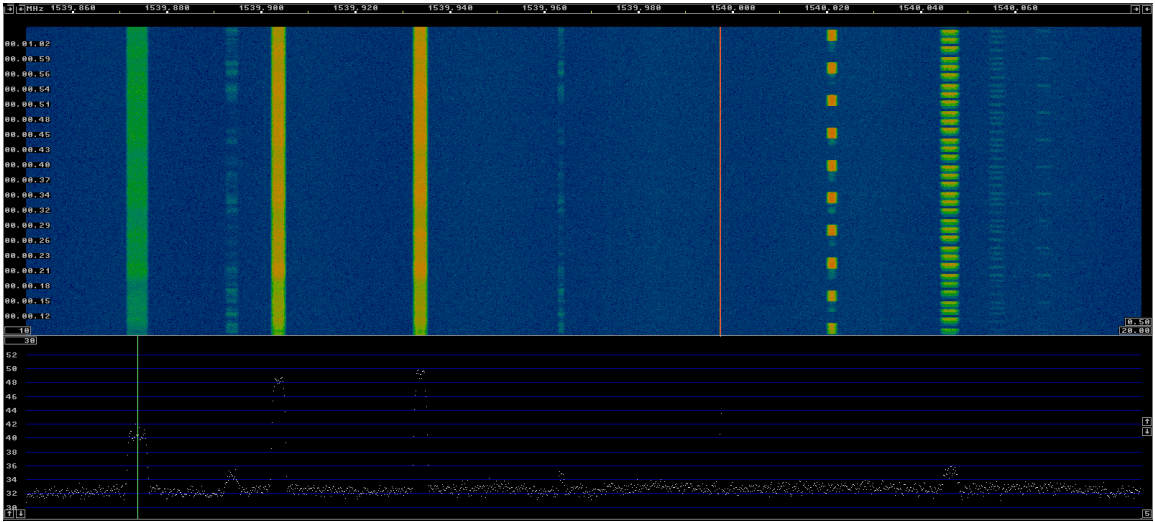
- **Hardware:**
  - L-band antenna. Tipically a patch antenna, can also use a dish
  - LNA (preamplifier)
  - RTL-SDR dongle
  - Single board ARM computer: C.H.I.P. or Raspberry Pi 3
  - Outernet sells a kit with these items
- **Software:**
  - rxOS: A Linux image for ARM that does everything for you
  - Most of the software is open-source
  - Key parts of the receiver are closed-source binary only
  - GPL libraries (`librtlsdr` and `libmirisdr`) are used in the closed-source receiver. Possible GPL violation
  - The protocols, modulation and format of the signal are secret

Why reverse engineer Outernet?

- A secret protocol and closed-source software don't serve well the goal of easing worldwide Internet access
- Amateur Radio operators started playing with Outernet. Closed-source and secret protocols detrimental for Amateur Radio
- Things I knew before starting:
  - RF goes in, files come out. About 2kbps bitrate or 20MB of content per day
  - `outernet-linux-lband` closed-source software (Older version for Linux x86_64. Now everything is for ARM): `sdr100-1.0.4`, SDR receiver for RTL-SDR; `ondd-2.2.0`, does everything else
  - IQ recordings by Scott Chapman K4KDR

Outline

Waterfall in Linrad



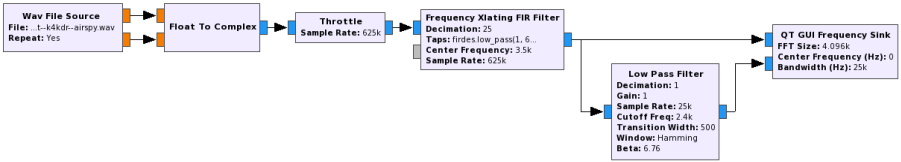
Modulation

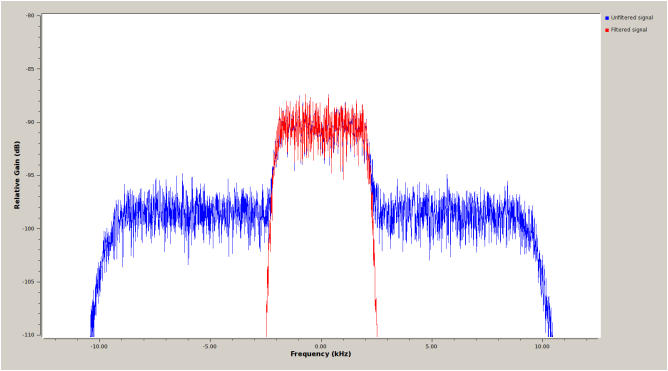
- 4.8kHz wide
- Looks like a hump in the noise floor
- “Any sufficiently advanced communication scheme is indistinguishable from noise” — Phil Karn KA9Q.
- We suspect PSK modulation. BPSK and QPSK are good candidates
- We use GNU Radio for signal processing. First step: find out PSK order and baudrate

Modulation

- 4.8kHz wide
- Looks like a hump in the noise floor
- “Any sufficiently advanced communication scheme is indistinguishable from noise” — Phil Karn KA9Q.
- We suspect PSK modulation. BPSK and QPSK are good candidates
- We use GNU Radio for signal processing. First step: find out PSK order and baudrate

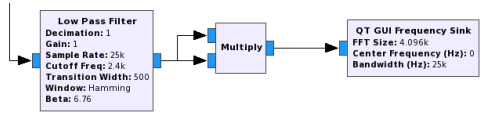
Reading from IQ wav file in GNU Radio

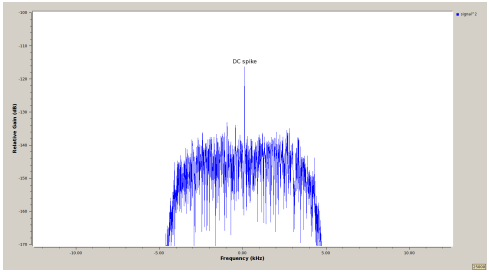




PSK order

Raise the signal to integer powers

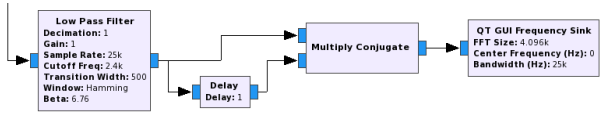


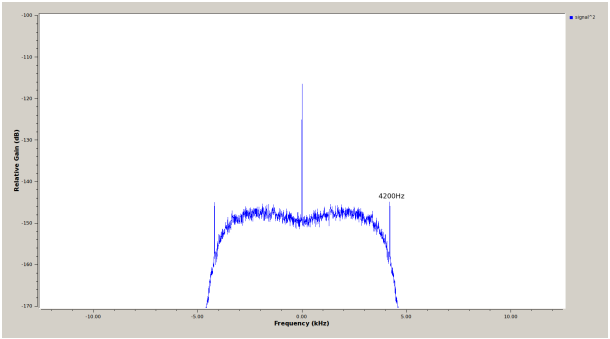


Power 2 of the signal has DC spike $\Rightarrow$ BPSK
For QPSK, we would need to go to 4th power

Baudrate

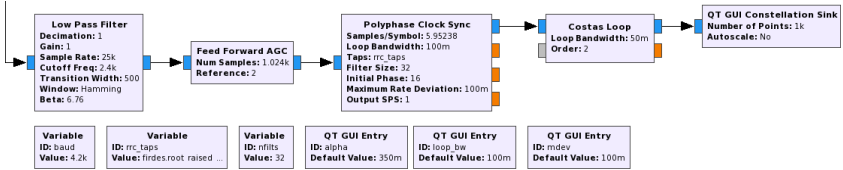
Cyclostationary analysis

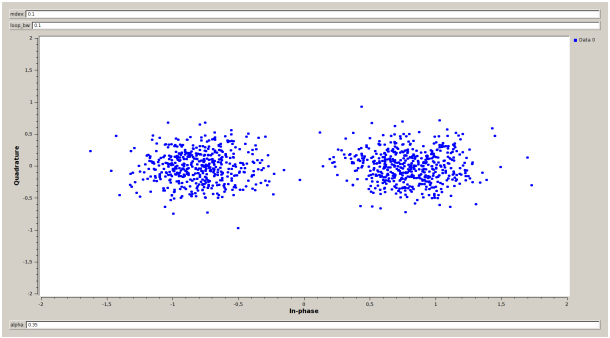




Baudrate is 4200baud

BPSK demodulation





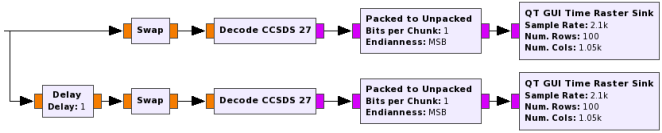
Coding

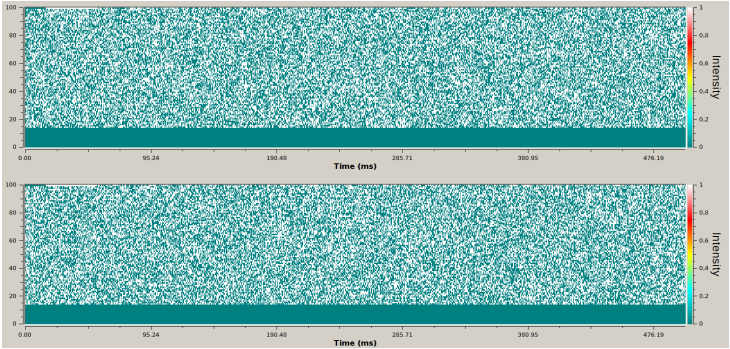
- Baudrate is 4200baud but bitrate is only about 2kbps
- We suspect $r = 1/2$ FEC in use
- Most popular choice: $r = 1/2, k = 7$ convolutional code with CCSDS polynomials
- We use Balint Seeber's AutoFEC to find FEC parameters
- Standard CCSDS convolutional code, but with the two polynomials swapped
- We use GNU Radio Viterbi decoder to decode FEC

Coding

- Baudrate is 4200baud but bitrate is only about 2kbps
- We suspect $r = 1/2$ FEC in use
- Most popular choice: $r = 1/2, k = 7$ convolutional code with CCSDS polynomials
- We use Balint Seeber's AutoFEC to find FEC parameters
- Standard CCSDS convolutional code, but with the two polynomials swapped
- We use GNU Radio Viterbi decoder to decode FEC

Viterbi decoding

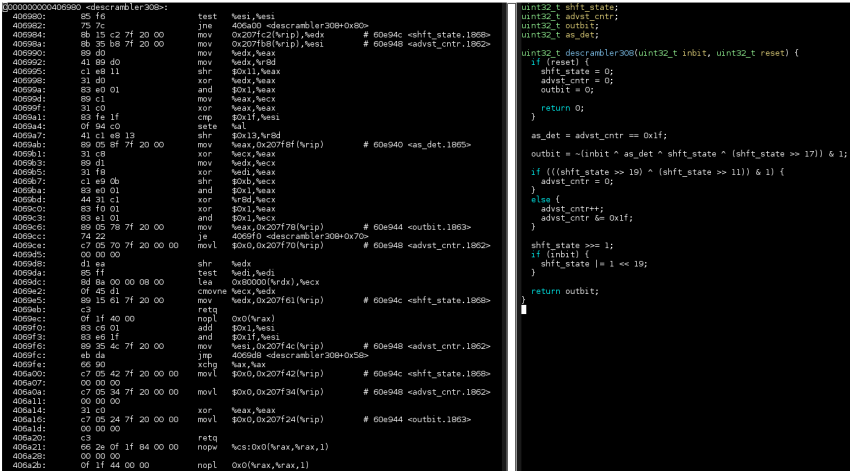




Output looks random $\Rightarrow$ we need a descrambler

Descrambler

- The most popular descramblers I knew of didn't work
- Reverse engineer the assembler code for the descrambler in `sdr100`

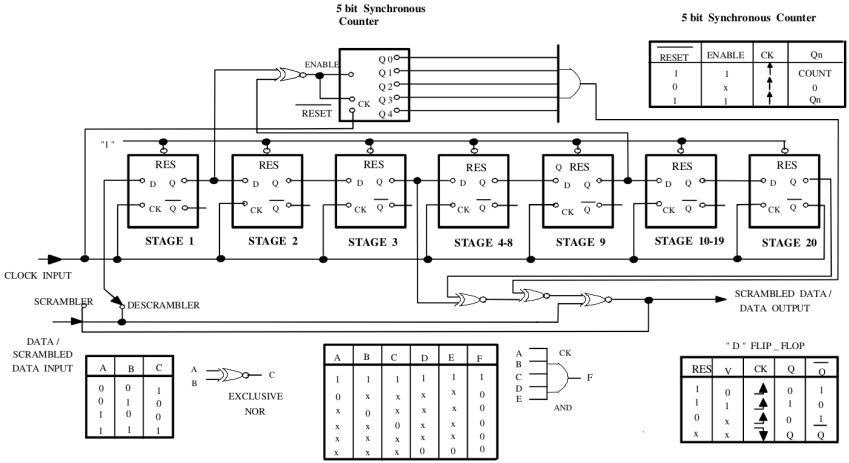


```
0000000000406980 <descrambler308>:
 406980:       85 f6                   test   %esi,%esi
 406982:       75 7c                   jne    406a00 <descrambler308+0x80>
 406984:       8b 15 c2 7f 20 00       mov    0x207fc2(%rip),%edx        # 60e94c <shift_state.1868>
 40698a:       8b 35 b8 7f 20 00       mov    0x207fb8(%rip),%esi        # 60e948 <advst_cntr.1862>
 406990:       89 d0                   mov    %edx,%eax
 406992:       41 89 d0                mov    %edx,%r8d
 406995:       c1 e8 11                shr    $0x11,%eax
 406998:       31 d0                   xor    %edx,%eax
 40699a:       83 e0 01                and    $0x1,%eax
 40699d:       89 c1                   mov    %eax,%ecx
 40699f:       31 c0                   xor    %eax,%eax
 4069a1:       83 fe 1f                cmp    $0x1f,%esi
 4069a4:       0f 94 c0                sete   %al
 4069a7:       41 c1 e8 13             shr    $0x13,%r8d
 4069ab:       89 05 8f 7f 20 00       mov    %eax,0x207f8f(%rip)        # 60e940 <as_det.1865>
 4069b1:       31 c8                   xor    %ecx,%eax
 4069b3:       89 d1                   mov    %edx,%ecx
 4069b5:       31 f0                   xor    %edi,%eax
 4069b7:       c1 e9 0b                shr    $0xb,%ecx
 4069ba:       83 e0 01                and    $0x1,%eax
 4069bd:       44 31 c1                xor    %r8d,%ecx
 4069c0:       83 f0 01                xor    $0x1,%eax
 4069c3:       83 e1 01                and    $0x1,%ecx
 4069c6:       89 05 78 7f 20 00       mov    %eax,0x207f78(%rip)        # 60e944 <outbit.1863>
 4069cc:       74 22                   je     4069f0 <descrambler308+0x70>
 4069ce:       c7 05 70 7f 20 00 00    movl   $0x0,0x207f70(%rip)        # 60e948 <advst_cntr.1862>
 4069d5:       00 00 00
 4069d8:       d1 ea                   shr    %edx
 4069da:       85 ff                   test   %edi,%edi
 4069dc:       8d 8a 00 00 08 00       lea    0x80000(%rdx),%ecx
 4069e2:       0f 45 d1                cmovne %ecx,%edx
 4069e5:       89 15 61 7f 20 00       mov    %edx,0x207f61(%rip)        # 60e94c <shift_state.1868>
 4069eb:       c3                      retq
 4069ec:       0f 1f 40 00             nopl   0x0(%rax)
 4069f0:       83 c6 01                add    $0x1,%esi
 4069f3:       83 e6 1f                and    $0x1f,%esi
 4069f6:       89 35 4c 7f 20 00       mov    %esi,0x207f4c(%rip)        # 60e948 <advst_cntr.1862>
 4069fc:       eb da                   jmp    4069d8 <descrambler308+0x58>
 4069fe:       66 90                   xchg   %ax,%ax
 406a00:       c7 05 42 7f 20 00 00    movl   $0x0,0x207f42(%rip)        # 60e94c <shift_state.1868>
 406a07:       00 00 00
 406a0a:       c7 05 34 7f 20 00 00    movl   $0x0,0x207f34(%rip)        # 60e948 <advst_cntr.1862>
 406a11:       00 00 00
 406a14:       31 c0                   xor    %eax,%eax
 406a16:       c7 05 24 7f 20 00 00    movl   $0x0,0x207f24(%rip)        # 60e944 <outbit.1863>
 406a1d:       00 00 00
 406a20:       c3                      retq
 406a21:       66 2e 0f 1f 84 00 00    nopw   %cs:0x0(%rax,%rax,1)
 406a28:       00 00 00
 406a2b:       0f 1f 44 00 00          nopl   0x0(%rax,%rax,1)
```

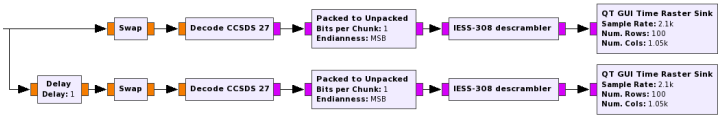
```
uint32_t shft_state;
uint32_t advst_cntr;
uint32_t outbit;
uint32_t as_det;

uint32_t descrambler308(uint32_t inbit, uint32_t reset) {
  if (reset) {
    shft_state = 0;
    advst_cntr = 0;
    outbit = 0;

    return 0;
  }

  as_det = advst_cntr == 0x1f;

  outbit = ~(inbit ^ as_det ^ shft_state ^ (shft_state >> 17)) & 1;

  if (((shft_state >> 19) ^ (shft_state >> 11)) & 1) {
    advst_cntr = 0;
  }
  else {
    advst_cntr++;
    advst_cntr &= 0x1f;
  }

  shft_state >>= 1;
  if (inbit) {
    shft_state |= 1 << 19;
  }

  return outbit;
}
```

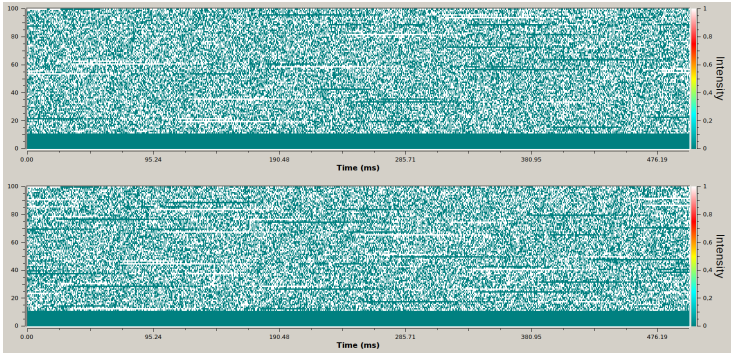
IESS-308 scrambler

It turns out the scrambler is the IESS-308, very popular in GEO satellite comms, but unheard of in Amateur LEO satellites



Descrambling



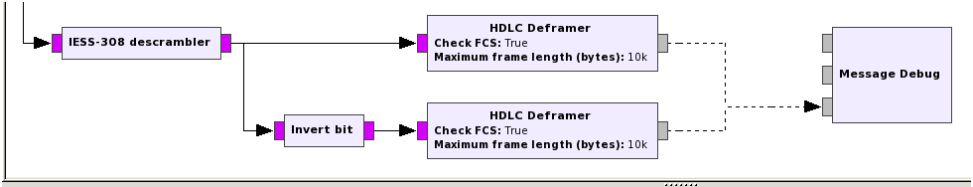


Now we can see some structure in the output

Framing

- Several functions in the `sdr100` binary have “HDLC” in them
- We suspect HDLC framing
- We use the HDLC deframer from `gr-kiss` (there’s also a stock deframer in GNU Radio)

HDLC deframing




```
****************************************
* MESSAGE DEBUG PRINT PDU VERBOSE *
()
pdu_length = 276
contents =
0000: ff ff ff ff ff ff 00 30 18 c1 dc a8 8f ff 01 04
0010: 3c 02 00 00 18 00 01 00 00 00 08 11 10 ba de e0
0020: bc 38 b4 34 e1 f9 74 73 92 f9 b8 41 52 db 20 ce
0030: a0 65 f5 c6 9b 66 0c c5 36 42 3c 66 fb 69 0e d8
0040: ca 2d fa 44 5a 57 74 8e 91 6b 98 34 45 51 3f e7
0050: c8 a6 08 69 f7 c5 67 71 cd b7 26 60 0a 03 cd 20
0060: 5d 49 45 88 bd a6 e9 89 87 86 25 3d 9e 83 9a e7
0070: fd 35 73 aa 4e 96 12 8d 1c 16 8f 0f 25 74 a2 12
0080: de bc 03 c9 47 57 5a 26 85 b2 a4 a8 be 4b 22 ce
0090: bd f7 e3 8a 9d 96 42 4a 25 7e c9 c3 be 64 ab 9d
00a0: b4 14 34 3a 24 4d 8a 40 1a 7e ad e8 0b d9 0e 0b
00b0: 8a a9 10 c2 c8 49 7c 69 4c a9 4e 65 53 e6 89 a4
00c0: aa 6b e8 7e ae 78 95 4b f8 96 68 05 17 15 8f 15
00d0: a2 79 0a 3d dd 52 37 86 fa 31 97 b9 d0 2b 1b 1e
00e0: 79 da 93 0c 02 81 77 3a 2e 35 80 10 74 0f 54 e3
00f0: 86 af cb c5 8b 38 64 78 de 09 37 9f 3d 3a 64 4e
0100: fe 86 21 7b 8c b1 55 05 5d fd 2a 4a 17 c1 37 69
0110: 5c d1 7b 1c
****************************************
```

Outline

Reverse engineering frames

- Techniques used:
  - Look at hex dumps of the frames
  - `ondd` usually gets frames from `sdr100` via Unix socket. Inject frames into `ondd` and see what happens
- Outernet uses custom network protocols ⇒ I get to name them as I like!

A typical frame

```
0000: ff ff ff ff ff ff 00 30 18 c1 dc a8 8f ff 01 04
0010: 3c 02 00 00 18 00 01 00 00 00 08 11 10 e5 21 4b
0020: 48 2c e0 77 00 86 4d 14 06 3c 24 f7 30 e7 19 4c
0030: ed 60 d4 44 94 6a 4a 18 34 ad b2 b5 92 01 b7 87
0040: 06 ba 80 61 a5 87 06 80 f6 04 12 f6 d9 12 13 02
0050: 64 0b 68 94 21 36 01 ab af 01 50 d0 13 4b dc b6
0060: 92 90 6b f4 76 27 73 3d 91 f5 84 3d 75 d9 77 90
0070: d2 74 15 49 66 e5 9a 57 df df 72 28 32 48 97 ed
0080: 9a 46 6e 68 8e 72 b3 54 5f 52 ce f6 f5 de c1 fd
0090: e4 e6 f8 a2 bd bb bb 65 cf 9e d0 ed 80 1e ad 8c
00a0: 0c b8 59 28 41 cf 27 d3 cf a9 9e 28 06 8e c0 c8
00b0: 42 7a bd ea da ae 7e 41 ee 24 c2 f9 28 b7 35 f6
00c0: 8b 12 13 23 1f fb 0d 3e 32 49 b9 75 4b 31 d3 29
00d0: 11 c1 48 a2 3b d4 8b 40 e6 2c 69 02 59 f2 f8 c8
00e0: d2 ea aa ce 63 57 ed f7 25 42 8e 9b 21 d4 64 07
00f0: 89 59 d0 47 d6 7b c7 3c c7 11 2c 91 d3 ca b1 52
0100: ea ba be e3 00 39 fb be 6a 02 52 e3 8f ac ba 30
0110: b7 d1 c2 3f
```

A typical frame

```
0000: ff ff ff ff ff ff 00 30 18 c1 dc a8 8f ff 01 04
0010: 3c 02 00 00 18 00 01 00 00 00 08 11 10 e5 21 4b
0020: 48 2c e0 77 00 86 4d 14 06 3c 24 f7 30 e7 19 4c
0030: ed 60 d4 44 94 6a 4a 18 34 ad b2 b5 92 01 b7 87
0040: 06 ba 80 61 a5 87 06 80 f6 04 12 f6 d9 12 13 02
0050: 64 0b 68 94 21 36 01 ab af 01 50 d0 13 4b dc b6
0060: 92 90 6b f4 76 27 73 3d 91 f5 84 3d 75 d9 77 90
0070: d2 74 15 49 66 e5 9a 57 df df 72 28 32 48 97 ed
0080: 9a 46 6e 68 8e 72 b3 54 5f 52 ce f6 f5 de c1 fd
0090: e4 e6 f8 a2 bd bb bb 65 cf 9e d0 ed 80 1e ad 8c
00a0: 0c b8 59 28 41 cf 27 d3 cf a9 9e 28 06 8e c0 c8
00b0: 42 7a bd ea da ae 7e 41 ee 24 c2 f9 28 b7 35 f6
00c0: 8b 12 13 23 1f fb 0d 3e 32 49 b9 75 4b 31 d3 29
00d0: 11 c1 48 a2 3b d4 8b 40 e6 2c 69 02 59 f2 f8 c8
00e0: d2 ea aa ce 63 57 ed f7 25 42 8e 9b 21 d4 64 07
00f0: 89 59 d0 47 d6 7b c7 3c c7 11 2c 91 d3 ca b1 52
0100: ea ba be e3 00 39 fb be 6a 02 52 e3 8f ac ba 30
0110: b7 d1 c2 3f
```

- Ethernet frame:
  - Broadcast destination
  - Source MAC
  - Custom ethertype
- Length: 276 bytes ⇒ aprox. 1 second over the air (this is Outernet's MTU)

L3 protocol: OP

- OP = “Outernet Protocol” (pun on IP)
- Handles fragmentation
- Packet order is preserved ⇒ fragmentation is very simple

```
0000: ff ff ff ff ff ff 00 30 18 c1 dc a8 8f ff 01 04
0010: 3c 02 00 00 18 00 01 00 00 00 08 11 10 e5 21 4b
.....................................................
```

- OP packet size
- Fragmentation `3c` = last fragment, `c3` = fragments remain
- ?
- Fragment number of last fragment
- Fragment number of this fragment

L4 protocol: LDP

- LDP = “Lightweight Datagram Protocol” (pun on UDP)
- Datagram protocol. Has some concept of ports to identify services

```
0000: ff ff ff ff ff ff 00 30 18 c1 dc a8 8f ff 01 04
0010: 3c 02 00 00 18 00 01 00 00 00 08 11 10 e5 21 4b
.....................................................
0110: b7 d1 c2 3f
```

- A field, B field ⇒ port (A = `0x1800`, B = `0x0000` marks a file block)
- LDP packet size
- Checksum

Time service packets

- Time packet broadcast every minute
- Used to set the receiver clock
- NTP not an option for receiver without internet access

```
0000: ff ff ff ff ff ff 00 30 18 c1 dc a8 8f ff 00 1c
0010: 3c 00 00 00 81 00 00 18 01 04 6f 64 63 32 02 08
0020: 00 00 00 00 57 f6 94 20 48 3a ca 8d 00 00 00 00
0030: 00 00 00 00 00 00 00 00 00 00 00 00
```

- Ethernet + OP + LDP header
- ASCII for `odc2` (Outernet DataCasting 2) ⇒ Groundstation for Americas satellite
- ?
- Unix timestamp 06 Oct 2016 18:12:48
- LDP checksum
- Padding (not included in OP or LDP packet) ⇒ mTU (minimum transfer unit) = 46 bytes

File service overview

- Broadcasts one file at a time (could broadcast several simultaneosly)
- Splits each file into 242 byte blocks
- Uses LDPC codes to recover the file even if some blocks are not received. FEC rate around 0.83
- Types of packets:
  - File announcement. Sent first. Basic info about file
  - File block (242 bytes of the file)
  - FEC block (242 bytes of parity check symbols from LDPC code)
- File blocks and FEC blocks are sent interleaved and in order (not necessary)

File announcement packets

- Large LDP packet (uses fragmentation)
- File info in ASCII XML
- Signed with X.509 certificate (to prevent spoofing?)

```
<?xml version="1.0" encoding="UTF-8"?>
<file>
  <id>2380</id>
  <path>opaks/dad7-Alt-right.html.tbz2</path>
  <hash>aed3e3b58193bdda9af9adb700972cb
        426ca26b336e36c2dfa0175b6e1deb4c8</hash>
  <size>109186</size>
  <block_size>242</block_size>
  <fec>ldpc:k=452,n=543,N1=2,seed=1000</fec>
</file>
```

- Hash is SHA256

File block packets

```
0000: ff ff ff ff ff ff 00 30 18 c1 dc a8 8f ff 01 04
0010: 3c 02 00 00 18 00 01 00 00 00 08 11 10 e5 21 4b
0020: 48 2c e0 77 00 86 4d 14 06 3c 24 f7 30 e7 19 4c
0030: ed 60 d4 44 94 6a 4a 18 34 ad b2 b5 92 01 b7 87
0040: 06 ba 80 61 a5 87 06 80 f6 04 12 f6 d9 12 13 02
0050: 64 0b 68 94 21 36 01 ab af 01 50 d0 13 4b dc b6
0060: 92 90 6b f4 76 27 73 3d 91 f5 84 3d 75 d9 77 90
0070: d2 74 15 49 66 e5 9a 57 df df 72 28 32 48 97 ed
0080: 9a 46 6e 68 8e 72 b3 54 5f 52 ce f6 f5 de c1 fd
0090: e4 e6 f8 a2 bd bb bb 65 cf 9e d0 ed 80 1e ad 8c
00a0: 0c b8 59 28 41 cf 27 d3 cf a9 9e 28 06 8e c0 c8
00b0: 42 7a bd ea da ae 7e 41 ee 24 c2 f9 28 b7 35 f6
00c0: 8b 12 13 23 1f fb 0d 3e 32 49 b9 75 4b 31 d3 29
00d0: 11 c1 48 a2 3b d4 8b 40 e6 2c 69 02 59 f2 f8 c8
00e0: d2 ea aa ce 63 57 ed f7 25 42 8e 9b 21 d4 64 07
00f0: 89 59 d0 47 d6 7b c7 3c c7 11 2c 91 d3 ca b1 52
0100: ea ba be e3 00 39 fb be 6a 02 52 e3 8f ac ba 30
0110: b7 d1 c2 3f
```

- We return to our typical frame
- Ethernet + OP + LDP header
- File ID
- Block number
- Block contents (242 bytes)
- LDP checksum
- FEC blocks have the same structure (and different A, B fields)

Putting all this together

- We have enough information to recover a file *provided all file blocks are received*
- LDPC codes used for FEC have not been reverse engineered yet
- Python implementation: `https://github.com/daniestevez/free-outernet`

```
$ ./free-outernet.py -k outernet.kiss
Receiving Ethernet frames from groundstation with MAC 00:30:18:c1:dc:a8
[Time service] Received time packet from odc2: 2016-10-15 18:01:01 UTC
Malformed LDP packet: length field mismatch
[Time service] Received time packet from odc2: 2016-10-15 18:02:01 UTC
[File service] New file announced: opaks/e89f-messages-0.html.tbz2 size 2435 bytes
Malformed LDP packet: length field mismatch
[File service] File reconstructed: opaks/e89f-messages-0.html.tbz2
[File service] New file announced: opaks/ed57-Amazon.com.html.tbz2 size 206080 bytes
Malformed LDP packet: length field mismatch
[Time service] Received time packet from odc2: 2016-10-15 18:03:01 UTC
......
[Time service] Received time packet from odc2: 2016-10-15 18:22:03 UTC
--------------------------------------------------------------------------
FEC debug info for file opaks/ed57-Amazon.com.html.tbz2 FEC decoding not implemented yet
ldpc:k=852,n=1023,N1=2,seed=1000
Length of FEC data: 41140 bytes; File size: 206080 bytes
--------------------------------------------------------------------------
[File service] File reconstructed: opaks/ed57-Amazon.com.html.tbz2
[File service] New file announced: opaks/efa3-Amber_Heard.html.tbz2 size 173736 bytes
Malformed LDP packet: length field mismatch
[Time service] Received time packet from odc2: 2016-10-15 18:23:04 UTC
[Time service] Received time packet from odc2: 2016-10-15 18:24:04 UTC
```

What do we have now?

- Lots of documentation about Outernet protocols:
  `http://destevez.net/tag/outernet/`
- GNU Radio receiver. Uses an SDR to get Outernet frames. Realtime-output by UDP socket and KISS file recording:
  `https://github.com/daniestevez/gr-outernet`
- Python implementation of the file transfer protocol. Can get frames in realtime by UDP socket or from KISS file recording:
  `https://github.com/daniestevez/free-outernet`

What about LDPC decoding?

- It would allow to recover files even if a few blocks are missing
- We have a few ideas of how it's supposed to work, but we are missing something
- Currently not much motivation to reverse engineer this
- Any help is welcomed
- Current progress:
  `https://github.com/daniestevez/free-outernet/issues/1`

Outline

Outernet groundstation satellite modem

- X.509 certificates for file announcements use as CN `odc2.outernet.is`, `odc3.outernet.is`, etc.
- Let's go to `http://odc2.outernet.is/`!
- The HTTP port is blocked now, but previously it led to the login page of the satellite modem (huge security flaw)
- It's the M7 modem from Datum Systems
- Lots of documentation available for you modem fans!

Specifications	
Operating Modes	TX and RX Continuous (SCPC) *Flex*LDPC, Flexible Block and Code Rates, Low Latency Advanced TPC and Industry Compatible Std and Custom Async Low Overhead Channels, AUPC Remote Modem Control Channel IP, Ethernet, Dual G.703/E1 (D&I), Serial, HSSI Opt Plug-in I/O Selections (Up to 2 per M7 Unit)
Data Rate Range	1.2 kbps to 59.04 Mbps, (1 bps steps)
Symbol Rate Range	2400 sps to 14.76 Msps (1 sps steps)
Frequency Tuning Range	M7 50-180 MHz, M7L 950-2150 MHz (1 Hz steps)
Modulation Types	BPSK,QPSK,OQPSK,8PSK,QAM,16QAM
FEC Options	None, Viterbi, TCM, Reed-Solomon, *Flex*LDPC TPC 4k and TPC 16k (Opt Plug-in HW)
Advanced *Flex*LDPC	Block Sizes 256,512,1k,2k,4k,8k,16k Rates 1/2,2/3,3/4,14/17,7/8,10/11,16/17
Turbo Product Code	TPC-4k 21/44, 1/2, 3/4, 7/8, 0.950 TPC-16k 1/2, 3/4, 7/8, 0.453, 0.922
Viterbi	1/2, 3/4, 7/8 (k=7), Trellis 2/3
Reed Solomon	Selectable N & K, IESS 308/309/310
Scrambler/Descrambler	IBS, V.35, IESS, TPC, RS, LDPC, EFD

Demodulator	
Input Acquisition Range	±100 Hz to ±3 MHz, 1 Hz Steps
Minimum Input Level	10 × Log(Symbol Rate) - 125 = Lvl (dBm)
Maximum Input Level	10 × Log(Symbol Rate) - 80 = Lvl (dBm)
Maximum IF Input Power Density	-20 dBc/Hz
Maximum Total Power	-10 dBm
Receive Acquisition Time	Typical 71 ms at 64 kbps, QPSK
Input Impedance	IF 50 or 75 Ohms BNC (User Selectable) L-Band 50 Ohms SMA
Input Return Loss	IF > 20 dB, L-Band > 16dB
Input Phase Noise	> Intelsat by 6 dB typical, 4 dB min
Demod Roll-Off Factor %	5, 8, 10, 15, 20, 25, 30, 35, 40 (%)

Smart Carrier Cancelling	
Delay Range	0 to 320 msec
Acquisition Time	< 30 Sec for Full Delay Sweep
Power Spectral Density	Ratio: +/- 10 dB: Symbol Rate Ratio: +/- 30% of Symbol Rate Frequency Offset: +/- 12.5% of Symbol Rate
Eb/No Degradation	PSD Ratio 0 dB BPSK/QPSK/OQPSK: 0.2 dB 8PSK/8QAM: 0.3 dB 16QAM: 0.5 dB

*Flex*LDPC™	Typical Eb/No for 1E-8 BER				Delay @64kbps
	QPSK	8PSK	8QAM	16QAM	
LDPC-1/2 - 2k	2.04 dB	n/a	3.80 dB	4.48 dB	49.6 ms
LDPC-1/2-4k	1.73 dB	n/a	3.44 dB	4.16 dB	98.0 ms
LDPC-1/2-8k	1.52 dB	n/a	3.19 dB	3.92 dB	195.0 ms
LDPC-1/2-16k	1.38 dB	n/a	3.04 dB	3.76 dB	388.6 ms
LDPC-2/3-2k	2.77 dB	4.88 dB	4.68 dB	5.85 dB	44.4 ms
LDPC-2/3-4k	2.46 dB	4.53 dB	4.36 dB	5.46 dB	87.5 ms
LDPC-2/3-8k	2.23 dB	4.28 dB	4.09 dB	5.19 dB	173.7 ms
LDPC-2/3-16k	2.09 dB	4.14 dB	3.91 dB	5.01 dB	346.1 ms
LDPC-3/4-2k	3.52 dB	5.97 dB	5.51 dB	6.78 dB	41.9 ms
LDPC-3/4-4k	3.14 dB	5.56 dB	5.11 dB	6.37 dB	82.4 ms
LDPC-3/4-8k	2.89 dB	5.27 dB	4.83 dB	6.07 dB	163.1 ms
LDPC-3/4-16k	2.72 dB	5.07 dB	4.63 dB	5.87 dB	325.0 ms
LDPC-7/8-2k	4.96 dB	7.89 dB	6.98 dB	8.48 dB	38.1 ms
LDPC-7/8-4k	4.32 dB	7.21 dB	6.40 dB	7.84 dB	74.6 ms
LDPC-7/8-8k	4.00 dB	6.86 dB	6.05 dB	7.51 dB	147.3 ms
LDPC-7/8-16k	3.90 dB	6.66 dB	5.87 dB	7.32 dB	293.6 ms
LDPC-10/11-2k	5.63 dB	8.73 dB	7.68 dB	9.37 dB	37.0 ms
LDPC-10/11-4k	5.00 dB	7.99 dB	7.02 dB	8.63 dB	72.3 ms
LDPC-10/11-8k	4.58 dB	7.51 dB	6.60 dB	8.18 dB	143.0 ms
LDPC-10/11-16k	4.40 dB	7.33 dB	6.35 dB	7.95 dB	284.5 ms

Guaranteed Eb/No is 0.2 dB > Typical

Interface Options: (Choose Up to Two Per Modem)

Serial Data Interface (S7)	
Main Interface Modes	Sync RS-232,449,V.35,EIA-530 (DB-25)
Internal Clock (ST) Accuracy	±1E-12, (±1 part per Trillion)
Doppler Buffer Depth	4 Bits to 524,284 Bits, 1 Bit Steps
ESC Overhead I/O Modes	Async RS-232,RS-485 (DB-25)
Adv Mux ESC OH Data Rate	Disabled, 300 bps to 3.5 Mbps, 1 bps Steps
Adv Mux (MCC) OH Data Rate	Disabled, 300 to 29.52 Mbps, 1 bps Steps
ESC Remote Signaling I/O's	Form C (Qty 2)

Advanced IP Interface (I7)	
Adv Ethernet IP Interface	10/100 BaseT, Gigabit Ethernet (RJ-45)
Operating System	Debian Linux Operating System
Operating Modes	Bridge and Vyatta Router
Packets Per Second	70,000 PPS
Network Protocols:	See Specification

Express Ethernet Interface (E7)	
Express Ethernet Ports	4Ports (RJ-45), 1 Port SFP
4 Port Interface	10/100 BaseT, Gigabit Ethernet (RJ-45)
SFP Port	Optional Gigabit or Optiuc Fiber
Ethernet Protocol	Layer 2 Switched Bridge Only
Features	QoS and VLAN Selectable

Dual G.703/E1 Interface (G7)	

Groundstation geolocation

- Geolocate the `odc?.outernet.is` IPs
- `odc2.outernet.is` Americas 216.129.171.61 ⇒ Toronto
- `odc3.outernet.is` Europe/Africa 212.165.126.66 ⇒ Amsterdam
- `odc4.outernet.is` Asia/Pacific 123.100.88.137 ⇒ Ketu Bay, New Zealand

Actual data throughput

- Outernet stated about 20MB of content per day
- Is this true?
- 242 byte blocks sent inside 272 byte Ethernet frames $\Rightarrow$ 12% overhead for headers
- All but the smallest files use LDPC codes with a rate of about 0.83 $\Rightarrow$ 20% overhead for FEC
- Total overhead of 30%
- Bitrate is 2.1kbps (At most. Should account for HDLC bit-stuffing)
- This only gives 15.14MB of content per day

Actual data throughput

- Outernet stated about 20MB of content per day
- Is this true?
- 242 byte blocks sent inside 272 byte Ethernet frames $\Rightarrow$ 12% overhead for headers
- All but the smallest files use LDPC codes with a rate of about 0.83 $\Rightarrow$ 20% overhead for FEC
- Total overhead of 30%
- Bitrate is 2.1kbps (At most. Should account for HDLC bit-stuffing)
- This only gives 15.14MB of content per day

Thanks for your attention!