Detecting ADS-B Signals with the RTL-SDR

Dayton Software Defined Radio Meetup
May 2, 2016

Meetup Sponsorship

PreTalen
innovative efficiency

This Meetup is sponsored by PreTalen

Company is growing quickly. Check out all the new office space!

We need good engineers to fill these desks.

If you know anyone who might be interested, contact me at bhart@pretalen.com or visit http://www.pretalen.com/careers

Meetup Updates

Next meetup tentatively scheduled for Monday June 6th

Topic: TBD -- Any suggestions?

Visit: http://www.meetup.com/Dayton-Area-Software-Defined-Radio-Meetup for the latest updates

Want to speak or know someone who does? Get in touch: bhart@pretalen.com

Sister Meetup

Dayton Cyber Security Meetup: http://www.meetup.com/Dayton-Cyber-Security/

Last week's meetup: Identifying Hardware Vulnerabilities

Next meetup will be scheduled soon. Check the Meetup site for updates.

What is ADS-B?

ADS-B stands for Automatic Dependent Surveillance – Broadcast

Worldwide standard for aircraft to broadcast identification and position

Currently required in Australia and other countries

Required in Europe by 2017

Required in the US by 2020

Many US aircraft already broadcast ADS-B signals

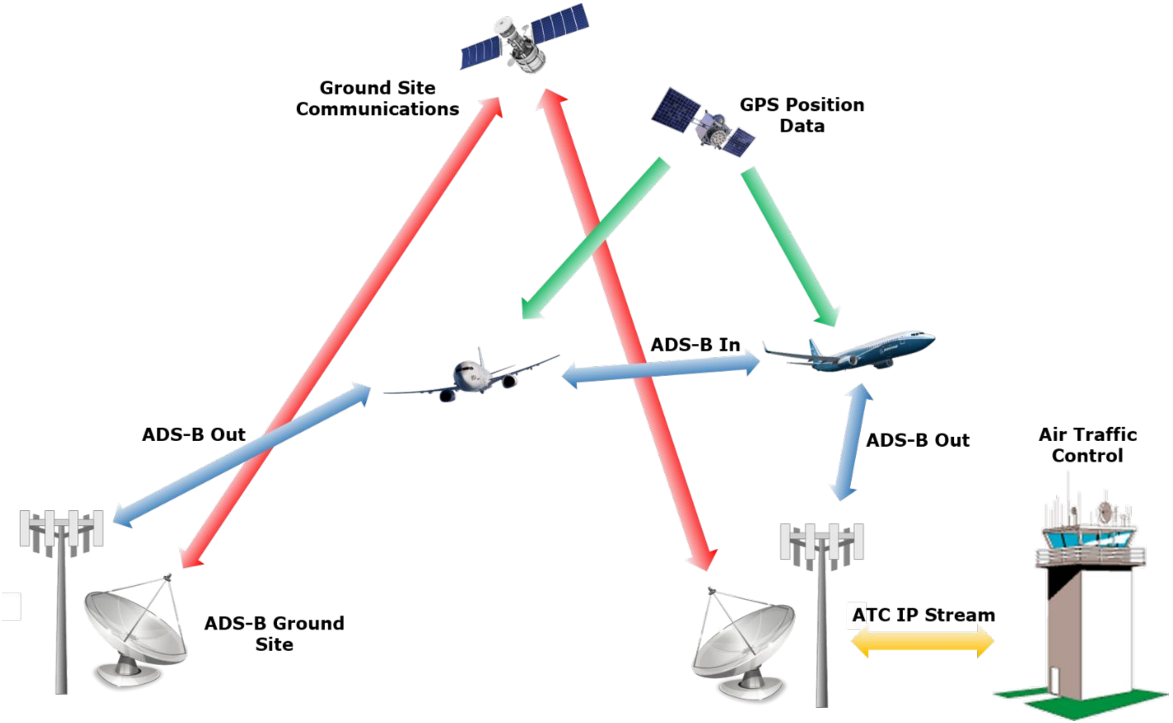
RF Specifications

- Internationally, ADS-B is broadcast on 1090 MHz
  - Also called 1090ES (Extended Squitter)
- In the US, ADS-B supports dual frequencies: 978 MHz and 1090 MHz
  - 978 MHz service is called UAT
  - UAT only used below 18,000 feet in US airspace
- Ground stations in the US broadcast on both frequencies:
  - TIS-B (Traffic Information Service Broadcast) on 978 MHz and 1090 MHz
  - FIS-B (Flight Information Service Broadcast) includes weather and is only on 978 MHz

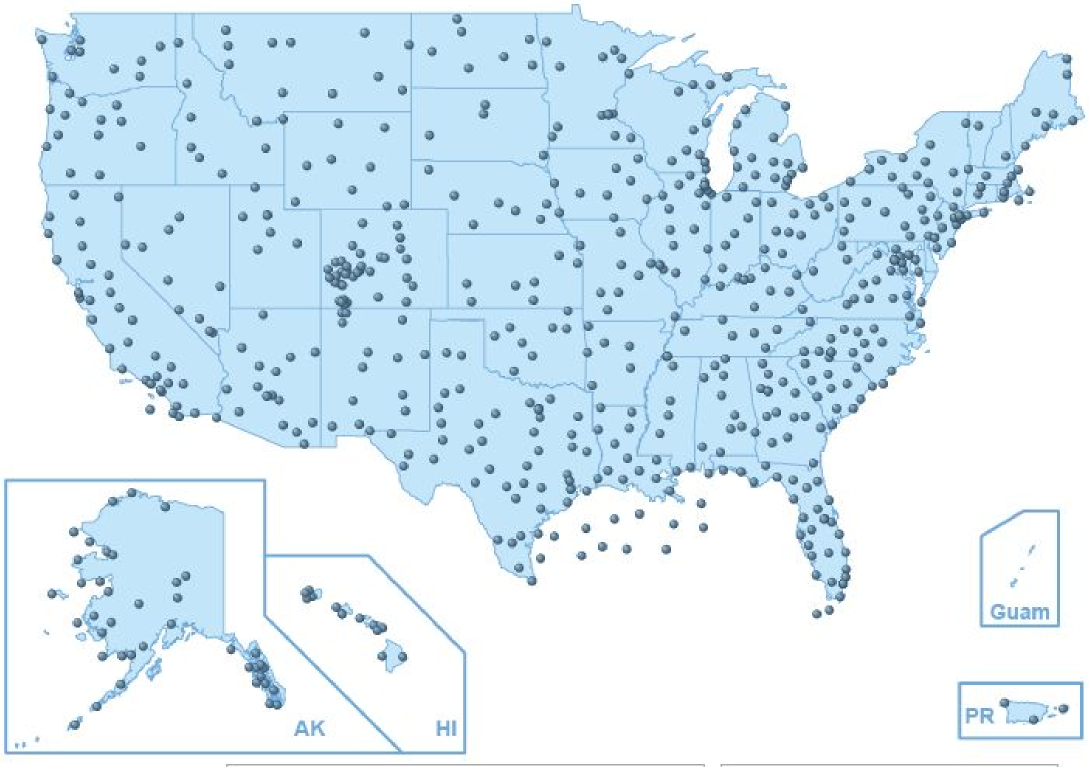
ADS-B In and ADS-B Out

- ADS-B In -- Incoming data to an aircraft about surrounding aircraft
- ADS-B Out -- Data broadcast out from an aircraft to the surrounding area
    - Can Be Received By:
        - Ground stations/Air Traffic Control
        - Other aircraft
        - Flight tracking services
        - Hobbyists with a basic SDR setup

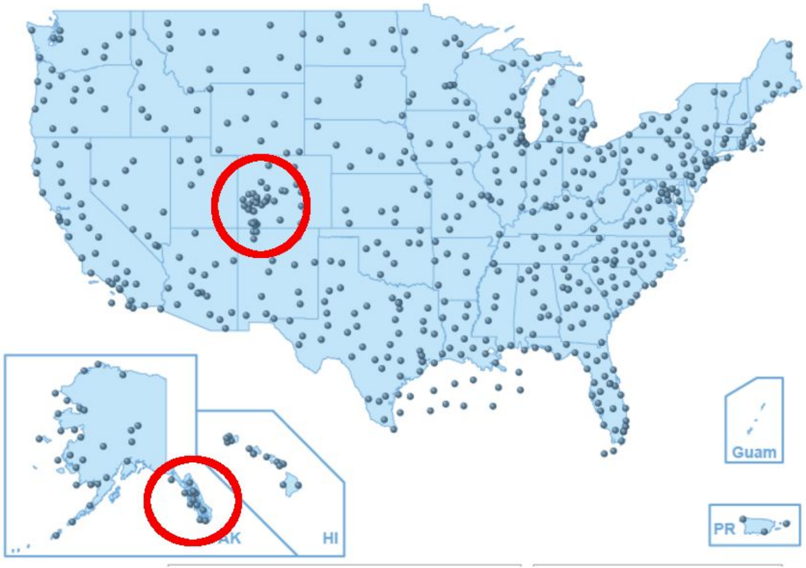
ADS-B System Overview



ADS-B Ground Stations

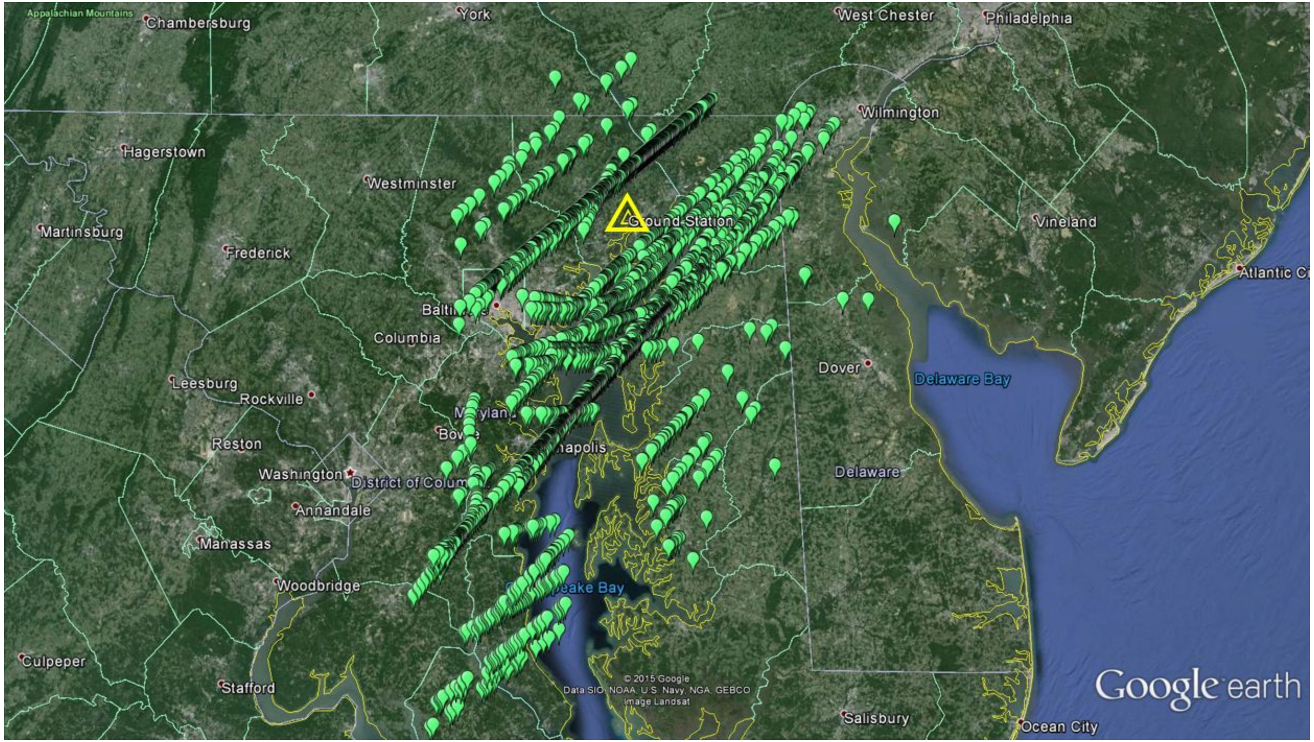


Clusters of Ground Stations?



- Multilateration projects by the FAA
  - Juneau, AK
  - Western Colorado
- Uses multiple ground stations to coordinate aircraft positions in areas where terrain obscure radar usage
- Techniques such as time-difference of arrival can be used to supplement ADS-B measurements

ADS-B Captures over Maryland



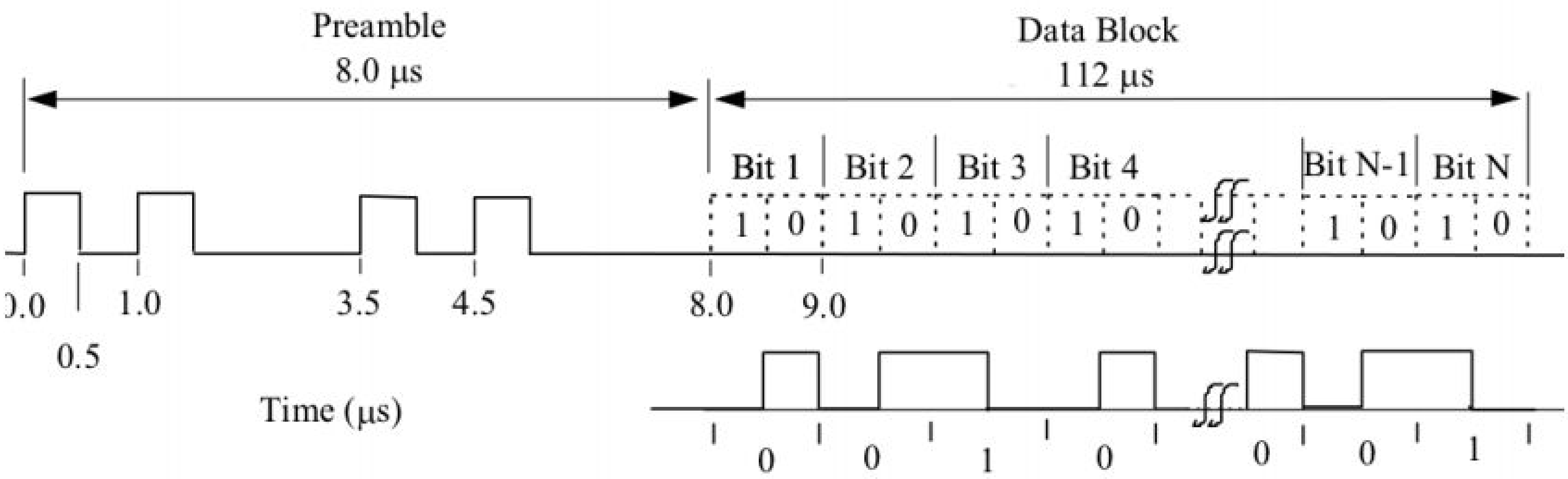
What’s In an ADS-B Transmission?

- Aircraft identification -- ICAO aircraft address
  - Some aircraft hide their identification and broadcast things like “XXXXXXX”
  - Sites like FlightAware hide some tail numbers. You can see everything looking at the raw ADS-B transmissions.
- Altitude
- Position (Latitude/Longitude)
- Velocity
- Other less common transmissions
  - Aircraft operation status
  - Test messages
  - Surface system status
  - Target state and status

ADS-B Message Types

DF	TC	Content
17	1 to 4	Aircraft identification
17	5 to 8	Surface position
17	9 to 18	Airborne position (Baro Alt)
17	19	Airborne velocities
17	20 to 22	Airborne position (GNSS Height)
17	23	Test message
17	24	Surface system status
17	25 to 27	Reserved
17	28	Extended squitter AC status
17	29	Target state and status (V.2)
17	30	Reserved
17	31	Aircraft Operation status

ADS-B Pack Transmission Structure



ADS-B Message Structure

Bit from	Bit to	Abbr.	Name
1	5	DF	Downlink Format
6	8	CA	Message Subtype
9	32	ICAO24	ICAO aircraft address
33	88	DATA	Data frame
89	112	PC	Parity check

ADS-B Message Structure

Bit from	Bit to	Abbr.	Name
1	5	DF	Downlink Format
6	8	CA	Message Subtype
9	32	ICAO24	ICAO aircraft address
33	88	DATA	Data frame
89	112	PC	Parity check

Decoding Aircraft Identification in Python

```
from math import log

def hex2bin(hexstr):
  scale = 16
  num_of_bits = len(hexstr)*log(scale, 2)
  binstr = bin(int(hexstr, scale))[2:].zfill(int(num_of_bits))
  return binstr

def bin2int(binstr):
  return int(binstr, 2)

charset = '#ABCDEFGHIJKLMNOPQRSTUVWXYZ#####_###############0123456789######'

msg = "8D4840D6202CC371C32CE0576098"
msgbin = hex2bin(msg)
databin = msgbin[32:88]

csbin = databin[8:]
callsign = ''

for i in range(0,8):
    callsign += charset[ bin2int(csbin[6*i:6*i+6]) ]

chars_to_remove = ['_', '#']
cs = callsign.translate(None, ''.join(chars_to_remove))

print cs
```

Aircraft Identification Message

`'#ABCDEFGHIJKLMNOPQRSTUVWXYZ#####_###############0123456789######'`

```
HEX: 202CC371C32CE0
BIN: 00100 000 | 001011 001100 001101 110001 110000 110010 110011 100000
DEC:           |   11     12     13     49     48     50     51     32
LTR:           |   K      L      M      1      0      2      3      _
```

Airborne Position Message

MSG bits	# bits	Abbr	Content
1-5	5	DF	Downlink format
33-37	5	TC	Type code
38-39	2	SS	Surveillance status
40	1	NICsb	NIC supplement-B
41-52	12	ALT	Altitude
53	1	T	Time
54	1	F	CPR odd/even frame flag
55-71	17	LAT-CPR	Latitude in CPR format
72-88	17	LON-CPR	Longitude in CPR format

Decoding CPR Format

- Step 1: Collect Odd and Even Frames
  - Frames are sent in pairs: “odd frames” and “even frames”
  - Bit 54 in the airborne position message specifies odd or even frame:
    - 0 - odd frame / 1 - even frame
- Step 2: Convert Binary to Decimal Value
  - Values are 17 bits
  - Convert to decimal by dividing by 2^{17} (= 131072)
- Step 3: Calculate the Latitude Index j

$$j = floor\left(59 * Lat_{CPR-E} - 60 * Lat_{CPR-O} + 0.5\right)$$

Decoding CPR Format - Latitude

- Step 4: Calculate Relative Latitudes

  $DLat_{Even}$ = 360.0 / 60.0 = 6.000

  $DLat_{Odd}$ = 360.0 / 59.0 = 6.102

$$Lat_E = DLat_E * (mod(j, 60) + Lat_{CPR-E})$$

$$Lat_E = Lat_E - 360 \quad \text{if } (Lat_E \geq 270)$$

$$Lat_O = DLat_O * (mod(j, 59) + Lat_{CPR-O})$$

$$Lat_O = Lat_O - 360 \quad \text{if } (Lat_O \geq 270)$$

Decoding CPR Format - Longitude

- Step 5: Compute Longitude
  - Step 5a : Define two functions:

```
def cprN(lat, is_odd):
    nl = cprNL(lat) - is_odd
    return nl if nl > 1 else 1


def cprNL(lat):
    try:
        nz = 60
        a = 1 - math.cos(math.pi * 2 / nz)
        b = math.cos(math.pi / 180.0 * abs(lat)) ** 2
        nl = 2 * math.pi / (math.acos(1 - a/b))
        return int(nl)
    except:
        # happens when latitude is +/-90 degree
        return 1
```

Decoding CPR Format - Longitude

- Step 5b : Compute two interim variables m and ni

$$ni = \begin{cases} N(Lat_E, 0) & \text{if } (T_0 \geq T_1) \\ N(Lat_O, 1) & \text{else} \end{cases}$$

$$m = \begin{cases} floor\left[Lon_{CPR-E} * (NL(Lat_E) - 1) - Lon_{CPR-O} * NL(Lat_E) + 0.5\right] & \text{if } (T_0 \geq T_1) \\ floor\left[Lon_{CPR-E} * (NL(Lat_O) - 1) - Lon_{CPR-O} * NL(Lat_O) + 0.5\right] & \text{else} \end{cases}$$

- Step 5c: Compute longitude

$$Lon = \begin{cases} \frac{360.0}{ni} * (Mod(m, ni) + Lon_{CPR-E}) & \text{if } (T_0 \geq T_1) \\ \frac{360.0}{ni} * (Mod(m, ni) + Lon_{CPR-O}) & \text{else} \end{cases}$$

$$Lon = Lon - 360 \quad \text{if } (Lon \geq 180)$$

Putting it All Together In Python

```python
def cpr2position(cprlat0, cprlat1, cprlon0, cprlon1, t0, t1):
  cprlat_even = cprlat0 / 131072.0
  cprlat_odd  = cprlat1 / 131072.0
  cprlon_even = cprlon0 / 131072.0
  cprlon_odd  = cprlon0 / 131072.0

  air_d_lat_even = 360.0 / 60
  air_d_lat_odd = 360.0 / 59

  # compute latitude index 'j'
  j = int(59 * cprlat_even - 60 * cprlat_odd + 0.5)

  lat_even = float(air_d_lat_even * (j % 60 + cprlat_even))
  lat_odd  = float(air_d_lat_odd  * (j % 59 + cprlat_odd))

  if lat_even >= 270:
    lat_even = lat_even - 360

  if lat_odd >= 270:
    lat_odd = lat_odd - 360

  # check if both are in the same latitude zone, exit if not
  if cprNL(lat_even) != cprNL(lat_odd):
    return None
```

```python
# compute ni, longitude index m, and longitude
if (t0 > t1):
  ni = cprN(lat_even, 0)
  m = math.floor( cprlon_even * (cprNL(lat_even)-1) \
      - cprlon_odd * cprNL(lat_even) + 0.5 )
  lon = (360.0 / ni) * (m % ni + cprlon_even)
  lat = lat_even
else:
  ni = cprN(lat_odd, 1)
  m = math.floor( cprlon_even * (cprNL(lat_odd)-1) \
     - cprlon_odd * cprNL(lat_odd) + 0.5 )
  lon = (360.0 / ni) * (m % ni + cprlon_odd)
  lat = lat_odd

if lon > 180:
  lon = lon - 360

return [lat, lon]
```

Computing Altitude

- Much easier than lat/long computation!
- Step 1: Look at the Q-bit (bit 48)
  - If value is 0 then altitude is encoded in multiples of 100 ft.
  - If value is 1 then altitude is encoded in multiple of 25 ft.
- Step 2: Remove the Q bit

```
1100001 1 1000
        ^
      Q-bit
```

```
N = 1100001 1000 => 1560 (in decimal)
```

Computing Altitude

- Step 3: Compute the altitude

```
Altitude = N*100 - 1000 (feet) if Q = 0
Altitude = N*25  - 1000 (feet) if Q = 1
```

- Example from previous slide:

```
N = 1560 , Q = 1
Altitude  = 1560*25 - 1000  = 38,000 feet
```

Altitude Accuracy and Range

For Q = 0:

- Accuracy is +/- 100 feet
- Range is -1000 to 203,700 feet

For Q = 1:

- Accuracy is +/- 25 feet
- Range is -1000 to 50,175 feet

Airborne Velocity Messages

MSG Bits	N bits	Abbr	Content
33-37	5	TC	Type code
38-40	3	ST	Subtype
41	1	IC	Intent change flag
42	1	RESV_A	Reserved-A
43-45	3	NAC	Velocity uncertainty (NAC)
46	1	S-WE	East-West velocity sign
47-56	10	V-WE	East-West velocity
57	1	S-NS	North-South velocity sign
58-67	10	V-NS	North-South velocity
68	1	VrSrc	Vertical rate source
69	1	S-Vr	Vertical rate sign
70-78	9	Vr	Vertical rate
79-80	2	RESV_B	Reserved-B
81	1	S-Dif	Diff from baro alt, sign
82-88	7	Dif	Diff from baro alt

Subtype 1 = Ground speed (subsonic)
Subtype 2 = Ground speed (supersonic)
Subtype 3/4 = Airspeed (rarely used)

0 = Flying West to East
1 = Flying East to West

Velocity component in knots

0 = Flying South to North
1 = Flying North to South

Velocity component in knots

0 = Down / Descending
1 = Up / Ascending

Rate in ft/min

Computing Speed and Heading

Speed is computed using the square root of the square of the components:

$$v = \sqrt{V_{we}^2 + V_{sn}^2}$$

Heading is the inverse tangent of the components

$$h = arctan(\frac{V_{we}}{V_{sn}}) * \frac{360}{2\pi} \quad \text{(deg)}$$

Dump1090 to the Rescue!

Dump 1090 is a Mode S decoder specifically designed for RTLSDR devices.

Features include:

- Robust decoding of messages from weak signals
- Network support: TCP stream, HTTP
- Embedded HTTP server with Google Map overlay
- Bit error correction using the 24 bit CRC
- Command line interface
- Ability to decode a wide variety of message types
- Can decode raw IQ samples from a file

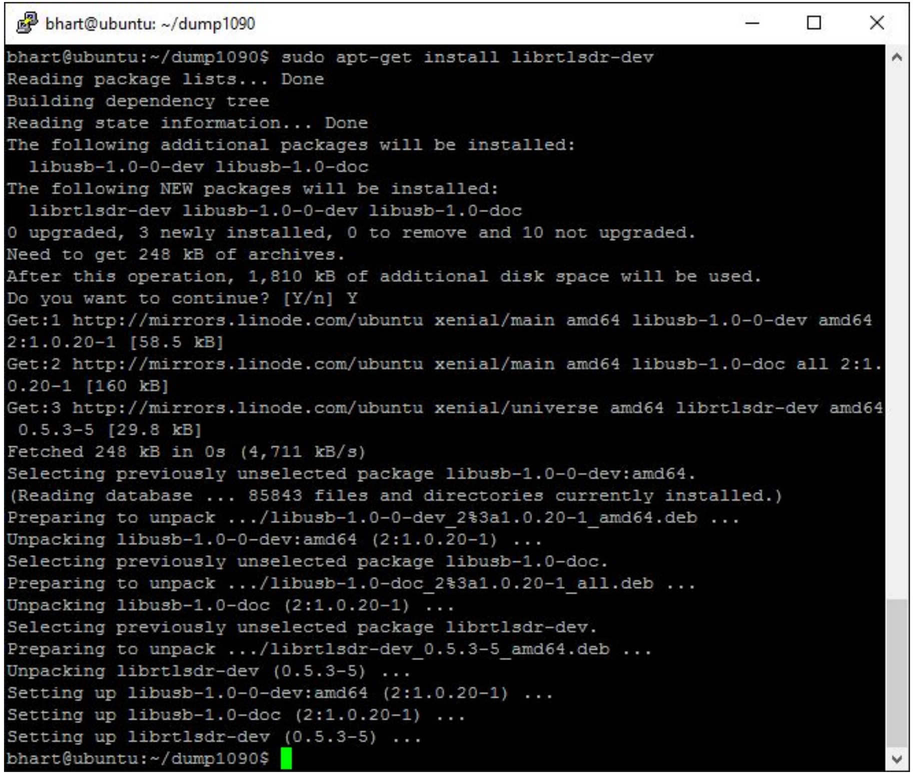
Installation of Dump1090

First install librtlsdr:

```
sudo apt-get install \
   librtlsdr-dev
```

Also install git (if you haven't already):

```
sudo apt-get install git
```



Installation of Dump1090

Download the dump1090 repo from GitHub:

```
git clone https://github.com/antirez/dump1090.git
```

Or

```
git clone https://github.com/MalcolmRobb/dump1090.git
```

Open the directory and run make:

```
cd dump1090 && make
```

Run the program with interactive mode and web browser support:

```
./dump1090 --interactive --net --aggressive
```

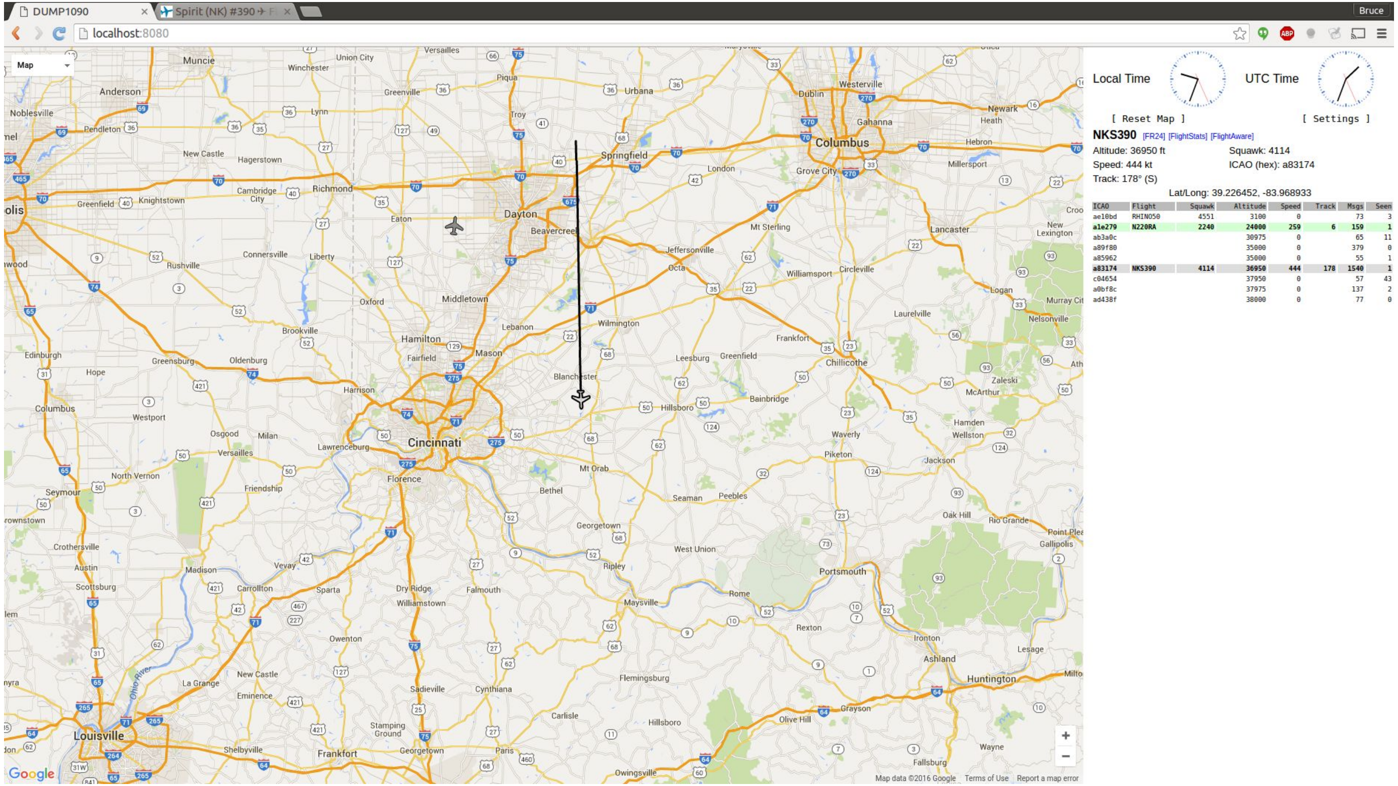
Dump1090 Interactive CLI

bhart@bhart-asus-laptop: ~/Projects/dayton-sdr/2016-05/dump1090

bhart@bhart-asus-laptop: ~/Projects/dayton-sdr/2016-05/dump1090 80x24

Hex	Mode	Sqwk	Flight	Alt	Spd	Hdg	Lat	Long	Sig	Msgs	Ti
A0BF8C	S			38000					5	163	1
AE10BD	S	4551	RHINO50	3100					17	89	1
A85962	S			35000					15	89	1
AB3A0C	S			30975					5	65	30
A89F80	S			35000					13	442	1
AD438F	S			38000					10	101	2
A1E279	S	2240	N220RA	24000	258	006	39.746	-84.436	7	181	1
A83174	S	4114	NKS390	36975	444	178	39.198	-83.968	6	1570	6

Dump1090 Web Interface

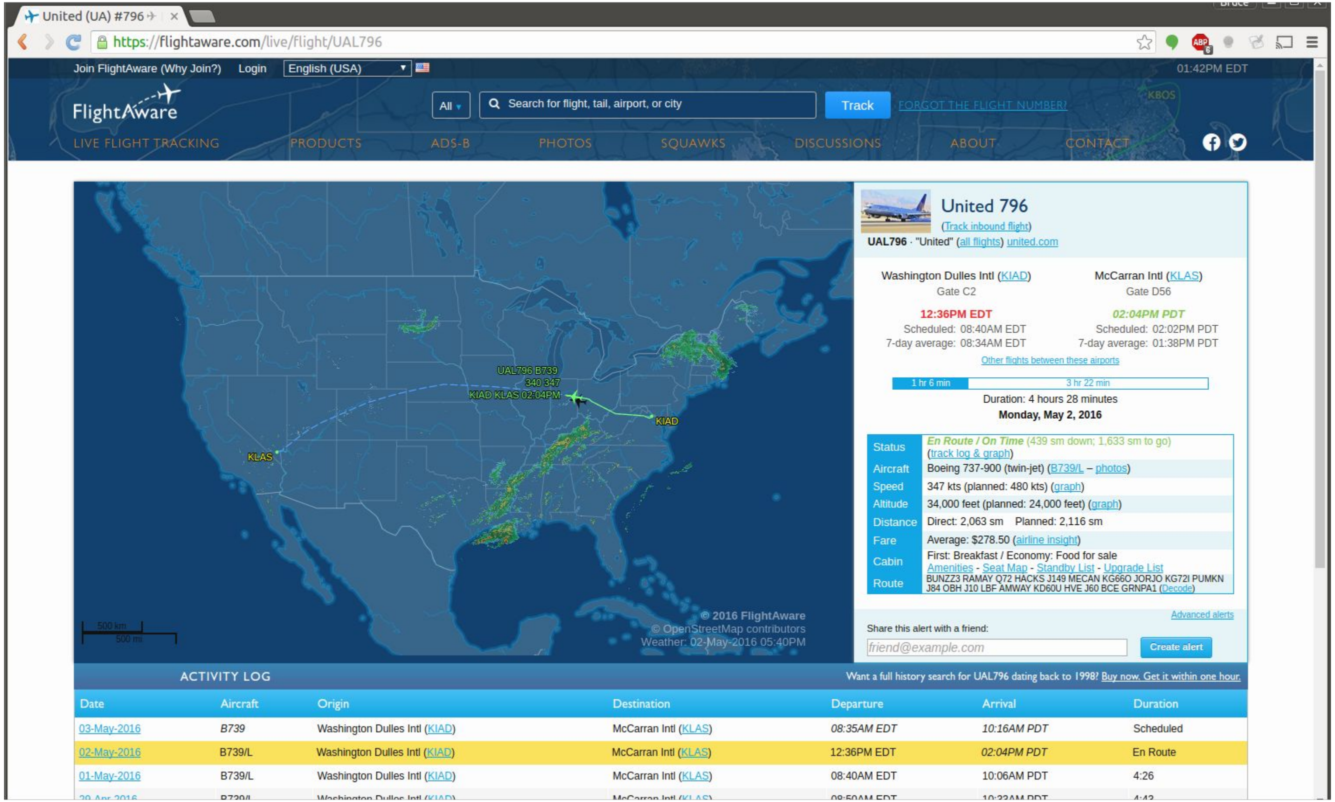


Dump1090 in Action

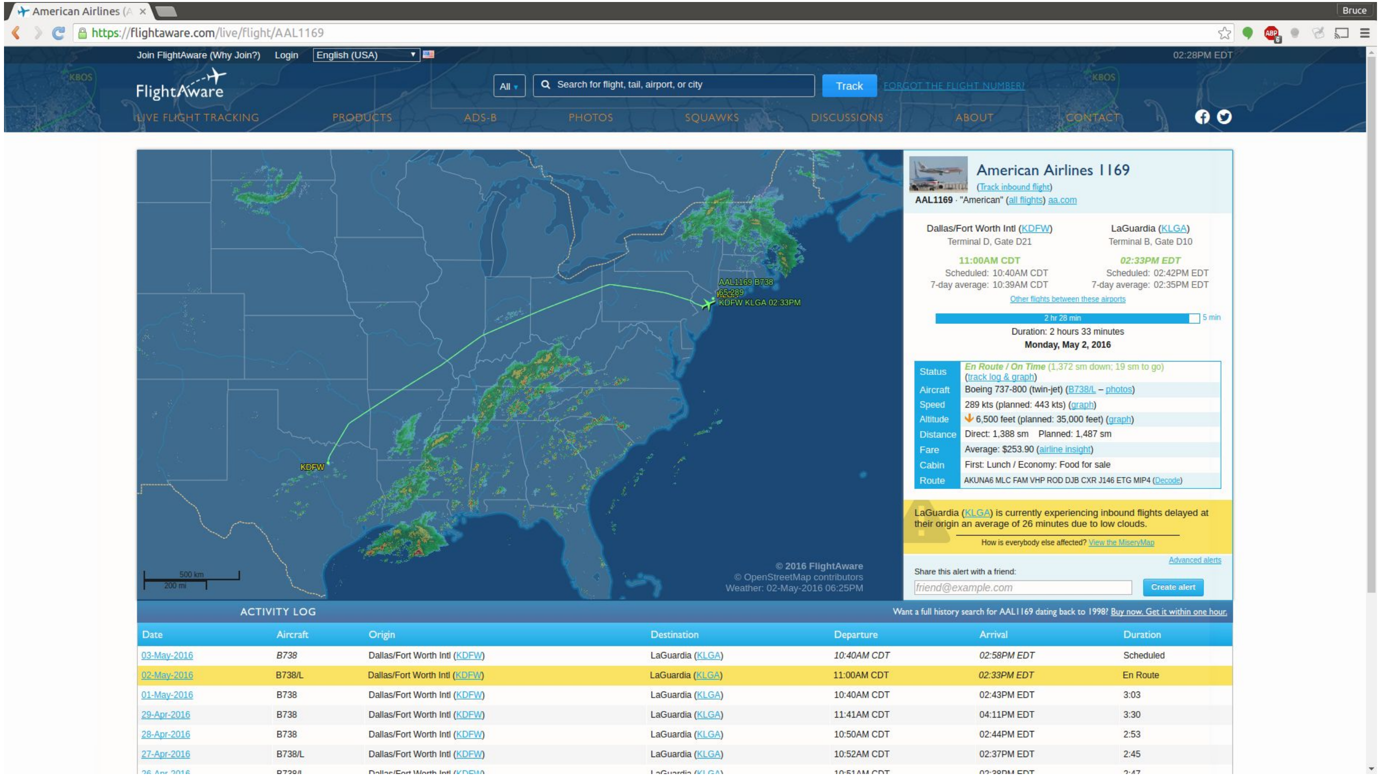
Command line interface

Web interface

United Airlines Flight 796



American Airlines Flight 1169 (AAL1169)



Other Equipment to Improve Reception

1090 MHz RF Filter ($20 on Amazon)

1090 MHz antenna ($9 on Amazon)

Thank You!

Any questions?

Next Meetups for Dayton SDR and Dayton Cyber Security will be posted soon

Slides will be posted on the Meetup site tonight