logparser

March 26, 2021

By @VirusFriendly

```
[1]: import base64
     import json
     import time
     import pandas
```

```
[2]: def find_frame(data):
         begin = -1

         if data[:2] != b'\x10\x02':
             for i in range(1, len(data)-7):
                 if data[i:i+4] == b'\x10\x03\x10\x02':
                     begin = i+2
                     break
         else:
             begin = 0

         return(begin)
```

```
[3]: def parse_frame(data):
         padding = find_frame(data)

         if find_frame(data) == -1:
             display('Bad Frame')
             display(data)
             return None

         data = data[padding:]

         if data[5+data[3]:data[3]+7] != b'\x10\x03':
             if find_frame(data[5+data[3]:]) == -1:
                 display('Frame Length Fail')
                 return None

             # This data uses 0x1002 to symbolize start of frame and 0x1003 to␣
     ↪symbolize the end of frame
```

```
        # To protect against 0x1003 being included in the datastream and␣
↪misinterpreted as an end of frame
        # The device escapes 0x10 as 0x1010. Thus 0x1003 will become 0x101003.
        # This causes our length checks to fail, so we must detect them and␣
↪de-escape the 0x1010 sequence

        frags = list()
        frag_start = 0

        for x in range(2, 5+data[3]+find_frame(data[5+data[3]:])):
            if data[x:x+2] == b'\x10\x10' and x >= frag_start:
                padding=padding+1
                frags.append(data[frag_start:x+1])
                frag_start = x+2

        if 0 < frag_start < len(data):
            frags.append(data[frag_start:])

        y=b''.join(frags)

        if (len(y) < 3) or (len(y) < y[3]+5) or (len(y) < y[3]+7) or (y[y[3]+5:
↪y[3]+7] != b'\x10\x03'):
            display('bad frame not repaired')
            display(data[:5+data[3]+find_frame(data[5+data[3]:])])
            display(y[:data[3]+7])
            return None

        data=y

    n2kframe = dict()
    n2kframe['start'] = data[0:2]
    n2kframe['cmd'] = data[2]
    n2kframe['len'] = data[3]

    if n2kframe['cmd'] == 147:
        n2kframe['priority'] = data[4]
        n2kframe['pgn'] = int.from_bytes(data[5:8], 'little')
        n2kframe['dst'] = data[8]
        n2kframe['src'] = data[9]
        n2kframe['timestamp'] = int.from_bytes(data[10:14], 'little')
        n2kframe['datalen'] = data[14]

        if n2kframe['datalen'] > n2kframe['len']-11:
            display("Data Length Fail")
            return None

        pgndata = list()
        for byte in data[15:15+n2kframe['datalen']]:
            pgndata.append('{:02x}'.format(byte))

        n2kframe['data'] = ','.join(pgndata)
        n2kframe['crc'] = data[4+n2kframe['len']]
    else:
        unk_bytes = list()

        for byte in data[4:5+n2kframe['len']]:
            unk_bytes.append('{:02x}'.format(byte))

        n2kframe['data'] = ','.join(unk_bytes)

        for label in ['priority', 'pgn', 'dst', 'src', 'timestamp', 'datalen',␣
↪'crc']:
            n2kframe[label] = ''

    n2kframe['end'] = data[5+n2kframe['len']:5+n2kframe['len']+2]

    return n2kframe, n2kframe['len']+7+padding
```

[4]:
```
def parse_frames(data):
    begin = 0
    n2kframes = list()

    begin = find_frame(data)

    frame_no = 0

    while(True):
        if begin < 0:
            print("Could not find next frame")
            break
        elif begin > len(data)-7:
            break

        parsed_data = parse_frame(data[begin:])

        if parsed_data == None:
            print ("Bad frame at %s, finding next frame" % frame_no)
            begin=begin+7+find_frame(data[begin+7:])
            continue

        (frame, length) = parsed_data
        n2kframes.append(frame)
        begin=begin+length
        frame_no = frame_no+1

    return n2kframes
```

[5]:
```
def display_frame(n2kframe):
    return ','.join((str(n2kframe['priority']), str(n2kframe['pgn']),␣
 ↪str(n2kframe['src']), str(n2kframe['dst']), str(n2kframe['datalen']),␣
 ↪n2kframe['data']))
```

[6]:
```
def parse_timestamp(timestamp):
    epoch = timestamp/1000.0
    millsec = timestamp%1000
    epoch_str = time.strftime('%Y-%m-%d-%H:%M:%S', time.gmtime(epoch))
    return "%s.%d" % (epoch_str, millsec)
```

[7]:
```
def parse_log(logfile):
    lines = None
    capture = b''
    maxtime = 0

    with open(logfile) as n2klog:
        lines = n2klog.read().splitlines()

    for line in lines:
        log=json.loads(line)

        if int(log["milliunixtimestamp"]) >= maxtime:
            if 0 == maxtime:
                display(f"First timestamp: {int(log['milliunixtimestamp'])}")

            maxtime=int(log["milliunixtimestamp"])
        else:
            print("!!!!OUT OF SEQUENCE LOG!!!!!")
            print(maxtime, int(log["milliunixtimestamp"]))

        data = base64.b64decode(log['data'])
        capture = capture+data

    display(f"Last timestamp: {maxtime}")

    return pandas.DataFrame(parse_frames(capture))
```

[8]:
```
n2k_logs = list()
print("first log")
n2k_logs.append(parse_log('NMEA2000PacketCapture2021.log'))
#print("second log")
```

```
#n2k_logs.append(parse_log('NMEA2000PacketCapture2021b.log'))

# This second log seems to be a
```

```
first log
The history saving thread hit an unexpected error (OperationalError('disk I/O
error',)).History will not be written to the database.
```

```
'First timestamp: 1596916750497'
```

```
'Last timestamp: 1596917535978'
```

[9]:
```
for n2k_log in n2k_logs:
    display(n2k_log[n2k_log.src == 16])
```

```
              start  cmd  len priority    pgn dst src timestamp datalen  \
1       b'\x10\x02'  147   19        3  61184   0  16  23121358       8
12      b'\x10\x02'  147   19        3  61184   0  16  23121409       8
21      b'\x10\x02'  147   19        3  61184   0  16  23121459       8
32      b'\x10\x02'  147   19        3  61184   0  16  23121509       8
40      b'\x10\x02'  147   19        3  61184   0  16  23121559       8
...             ...  ...  ...      ...    ...  ..  ..       ...     ...
168722  b'\x10\x02'  147   19        3  61184   0  16  23906546       8
168731  b'\x10\x02'  147   19        3  61184   0  16  23906597       8
168740  b'\x10\x02'  147   19        3  61184   0  16  23906647       8
168752  b'\x10\x02'  147   19        3  61184   0  16  23906697       8
168764  b'\x10\x02'  147   19        3  61184   0  16  23906747       8

                           data  crc          end
1       01,7e,7e,00,00,00,09,00   78  b'\x10\x03'
12      01,7e,7e,00,00,00,09,00   26  b'\x10\x03'
21      01,7e,7e,00,00,00,09,00  232  b'\x10\x03'
32      01,7e,7e,00,00,00,09,00  182  b'\x10\x03'
40      01,7e,7e,00,00,00,09,00  132  b'\x10\x03'
...                         ...  ...          ...
168722  01,7e,7e,00,00,00,09,00   35  b'\x10\x03'
168731  01,7e,7e,00,00,00,09,00  239  b'\x10\x03'
168740  01,7e,7e,00,00,00,09,00  189  b'\x10\x03'
168752  01,7e,7e,00,00,00,09,00  139  b'\x10\x03'
168764  01,7e,7e,00,00,00,09,00   89  b'\x10\x03'

[15656 rows x 12 columns]
```

```
[10]: # This includes non-NMEA2000 canbus IDs as PGNs

      display("SoManyMessages")
      pgns_together = list()

      for n2k_log in n2k_logs:
          pgns_seperate = list()

          for pgn in n2k_log[n2k_log.pgn != ''].sort_values(by=['pgn']).pgn.unique():
              pgns_seperate.append(pgn)

              if pgn not in pgns_together:
                  pgns_together.append(pgn)

          display(pgns_seperate)

      display("pgns from both logs", pgns_together)
```

```
'SoManyMessages'


[59392,
 59904,
 60928,
 61184,
 126208,
 126992,
 127250,
 127251,
 127257,
 127258,
 128259,
 128267,
 128275,
 129025,
 129026,
 129029,
 129033,
 129044,
 129538,
 129539,
 129540,
 130306,
 130323,
 130821,
 130823,
 130827,
 130945]
```

```
'pgns from both logs'
```

```
[59392,
 59904,
 60928,
 61184,
 126208,
 126992,
 127250,
 127251,
 127257,
 127258,
 128259,
 128267,
 128275,
 129025,
 129026,
 129029,
 129033,
 129044,
 129538,
 129539,
 129540,
 130306,
 130323,
 130821,
 130823,
 130827,
 130945]
```

[11]:
```
pgn_map = {
    '59392': 'NMEA2000', # Acknowledge
    '59904': 'NMEA2000', # Request for Address Claimed
    '60928': 'NMEA2000', # Address Claimed
    '126208': 'Controller', # Request Group Function
    '126720': 'Proprietary', # Addressable Multi-Frame Proprietary
# For Future Research
# PGN 126720-32 Proprietary: Attitude Offsets
# PGN 126720-33 Proprietary: Calibrate Compass
# PGN 126720-34 Proprietary: True Wind Options
# PGN 126720-35 Proprietary: Simulate Mode
# PGN 126720-49 Set WAAS Satellite
# PGN 126720-50 Set Tzz Parameter
    '126992': 'NMEA2000', # System Time
    '126993': 'NMEA2000', # Heartbeat
    '127237': 'Autopilot',
    '127250': 'Weather', # Vessel Heading
    '127251': 'Weather', # Rate of Turn
    '127257': 'Weather', # Attitude
    '127258': 'Weather', # Magnetic Variation
    '128259': 'Speed', # Speed
    '128267': 'Speed', # Water Depth
    '128275': 'Speed',
    '129025': 'Weather', # Position, Rapid Update
    '129026': 'Weather', # COG & SOG, Rapid Update
    '129029': 'GPS', # GNSS Position Data
    '129033': 'GPS', # Time & Date
    '129044': 'GPS', # Datum
    '129538': 'GPS', # GNSS Control Status
    '129539': 'GPS', # GNSS DOPs
    '129540': 'GPS', # GNSS Sats in View
    '130306': 'Weather', # Wind Data
    '130311': 'Speed', # Environmental Parameters
    '130312': 'Weather', # Temperature
    '130314': 'Weather', # Actual Pressure
    '130316': 'Weather', # Temperature
    '130821': 'Auto Pilot', # NavSource Speed (FEC)
    '130323': 'Weather', # Meteorological Station Data
    '130823': 'Auto Pilot', #  Browser Control Status (FEC)
    '130827': 'Auto Pilot' #FURUNO Proprietary
}

display("WhatIsThis")

for n2k_log in n2k_logs:
    devices = list()

    for src in n2k_log["src"].unique():
        if '' == src:
            continue

        device = dict()
        device['src'] = src
        device['device'] = list()

        for pgn in n2k_log[n2k_log.src == src]["pgn"].unique():
            if str(pgn) in pgn_map.keys():
                if pgn_map[str(pgn)] not in device['device']:
                    device['device'].append(pgn_map[str(pgn)])
            else:
                device['device'].append("Unk_" + str(pgn))
        devices.append(device)

    pandas.set_option("max_colwidth", 200)
    display(pandas.DataFrame(devices).sort_values(by='src'))
```

```
'WhatIsThis'


    src                                   device
12    1                               [NMEA2000]
9     2                          [GPS, NMEA2000]
11    3                   [NMEA2000, Controller]
8     4       [Auto Pilot, NMEA2000, Controller]
10    5                   [Auto Pilot, NMEA2000]
1    16                              [Unk_61184]
6    32                              [Unk_61184]
7    35                        [Speed, NMEA2000]
0    36     [Weather, NMEA2000, GPS, Unk_130945]
2    48                              [Unk_61184]
3    50                              [Unk_61184]
4    64                              [Unk_61184]
5    65                              [Unk_61184]
```

[12]:
```
# This includes non-NMEA2000 canbus IDs as PGNs

display("TalkFast", "This turned out to overcomplicate the answer, as I␣
↪seperated the updates per device, whereas the admins just took a rough␣
↪average.")
pgn_refresh = dict()

for n2k_log in n2k_logs:
    for src in n2k_log["src"].unique():
        if '' == src:
            continue

        n2k_src = n2k_log[n2k_log.src == src]

        for pgn in n2k_src[n2k_src.pgn != ''].sort_values(by=['pgn']).pgn.
↪unique():
            first_ts = 0
            last_ts = 0
            smallest_delta = 0xffffffff
            entries = 0

            for ts in n2k_src[n2k_src.pgn == pgn]["timestamp"]:
                entries = entries + 1
                if 0 == last_ts:
                    last_ts = ts
                    first_ts = ts
                    continue

                delta = ts-last_ts
                last_ts = ts

                if delta < smallest_delta:
                    smallest_delta = delta

            if str(pgn) not in pgn_refresh.keys():
                pgn_refresh[str(pgn)] = list()

            pgn_refresh[str(pgn)].append({
                "Device": src,
                "Smallest Delta": smallest_delta,
                "Total Entries": entries,
                "First": first_ts,
                "Last": ts,
                "Average" : (ts-first_ts)/entries
            })

for pgn in pgn_refresh.keys():
    display(pgn)

    for _ in pgn_refresh[pgn]:
        display(_)
```

```
'TalkFast'

'This turned out to overcomplicate the answer, as I seperated the updates per device, whereas

'59392'

{'Device': 36,
 'Smallest Delta': 4,
 'Total Entries': 547,
 'First': 23122861,
 'Last': 23905669,
 'Average': 1431.0932358318098}

{'Device': 4,
 'Smallest Delta': 4860,
 'Total Entries': 155,
 'First': 23126323,
 'Last': 23903786,
 'Average': 5015.890322580645}

{'Device': 5,
 'Smallest Delta': 0,
 'Total Entries': 405,
 'First': 23122859,
 'Last': 23905672,
 'Average': 1932.8716049382715}

{'Device': 3,
 'Smallest Delta': 1,
 'Total Entries': 560,
 'First': 23122860,
 'Last': 23905669,
 'Average': 1397.8732142857143}

'60928'

{'Device': 36,
 'Smallest Delta': 20754,
 'Total Entries': 38,
 'First': 23123872,
 'Last': 23897362,
 'Average': 20355.0}

{'Device': 35,
 'Smallest Delta': 20754,
 'Total Entries': 38,
 'First': 23123872,
 'Last': 23897361,
 'Average': 20354.973684210527}

{'Device': 2,
 'Smallest Delta': 20754,
 'Total Entries': 38,
 'First': 23123870,
 'Last': 23897360,
 'Average': 20355.0}

{'Device': 5,
 'Smallest Delta': 20753,
 'Total Entries': 38,
 'First': 23123871,
 'Last': 23897361,
 'Average': 20355.0}

{'Device': 3,
 'Smallest Delta': 20754,
 'Total Entries': 38,
 'First': 23123871,
 'Last': 23897360,
 'Average': 20354.973684210527}

{'Device': 1,
 'Smallest Delta': 20754,
 'Total Entries': 38,
 'First': 23123867,
 'Last': 23897358,
 'Average': 20355.026315789473}

'126992'

{'Device': 36,
 'Smallest Delta': 992,
 'Total Entries': 785,
 'First': 23121876,
 'Last': 23905918,
 'Average': 998.7796178343949}

'127250'

{'Device': 36,
 'Smallest Delta': 3,
 'Total Entries': 31416,
 'First': 23121350,
 'Last': 23906768,
 'Average': 25.000572956455308}

'127251'

{'Device': 36,
 'Smallest Delta': 81,
 'Total Entries': 7854,
 'First': 23121376,
 'Last': 23906718,
 'Average': 99.99261522790934}
```

```
'127257'

{'Device': 36,
 'Smallest Delta': 988,
 'Total Entries': 785,
 'First': 23121878,
 'Last': 23905921,
 'Average': 998.7808917197452}

'127258'

{'Device': 36,
 'Smallest Delta': 990,
 'Total Entries': 785,
 'First': 23121877,
 'Last': 23905920,
 'Average': 998.7808917197452}

'129025'

{'Device': 36,
 'Smallest Delta': 182,
 'Total Entries': 3927,
 'First': 23121476,
 'Last': 23906718,
 'Average': 199.9597657244716}

'129026'

{'Device': 36,
 'Smallest Delta': 182,
 'Total Entries': 3927,
 'First': 23121477,
 'Last': 23906719,
 'Average': 199.9597657244716}

'129029'

{'Device': 36,
 'Smallest Delta': 975,
 'Total Entries': 785,
 'First': 23121894,
 'Last': 23905937,
 'Average': 998.7808917197452}

{'Device': 2,
 'Smallest Delta': 91,
 'Total Entries': 785,
 'First': 23122109,
 'Last': 23906585,
 'Average': 999.3324840764332}

'129033'

{'Device': 36,
 'Smallest Delta': 992,
 'Total Entries': 785,
 'First': 23121877,
 'Last': 23905919,
 'Average': 998.7796178343949}

'129044'

{'Device': 36,
 'Smallest Delta': 9984,
 'Total Entries': 78,
 'First': 23126899,
 'Last': 23896939,
 'Average': 9872.307692307691}

'129538'

{'Device': 36,
 'Smallest Delta': 60032,
 'Total Entries': 13,
 'First': 23136864,
 'Last': 23857621,
 'Average': 55442.846153846156}

'129539'

{'Device': 36,
 'Smallest Delta': 987,
 'Total Entries': 785,
 'First': 23121881,
 'Last': 23905924,
 'Average': 998.7808917197452}

'129540'

{'Device': 36,
 'Smallest Delta': 978,
 'Total Entries': 785,
 'First': 23121916,
 'Last': 23905960,
 'Average': 998.7821656050955}

'130306'

{'Device': 36,
 'Smallest Delta': 242,
 'Total Entries': 3141,
 'First': 23121375,
 'Last': 23906668,
 'Average': 250.01368990767273}

'130323'

{'Device': 36,
 'Smallest Delta': 989,
 'Total Entries': 785,
 'First': 23121878,
 'Last': 23905921,
 'Average': 998.7808917197452}

'130945'

{'Device': 36,
 'Smallest Delta': 4294967295,
 'Total Entries': 1,
 'First': 23606968,
 'Last': 23606968,
 'Average': 0.0}

'61184'

{'Device': 16,
 'Smallest Delta': 47,
 'Total Entries': 15656,
 'First': 23121358,
 'Last': 23906747,
 'Average': 50.16536791006643}

{'Device': 48,
 'Smallest Delta': 48,
 'Total Entries': 15656,
 'First': 23121360,
 'Last': 23906729,
 'Average': 50.164090444558}

{'Device': 50,
 'Smallest Delta': 45,
 'Total Entries': 15656,
 'First': 23121360,
 'Last': 23906730,
 'Average': 50.16415431783342}

{'Device': 64,
 'Smallest Delta': 47,
 'Total Entries': 15644,
 'First': 23121372,
 'Last': 23906734,
 'Average': 50.20212221938123}

{'Device': 65,
 'Smallest Delta': 47,
 'Total Entries': 15644,
 'First': 23121372,
 'Last': 23906734,
 'Average': 50.20212221938123}

{'Device': 32,
 'Smallest Delta': 48,
 'Total Entries': 15654,
 'First': 23121381,
 'Last': 23906736,
 'Average': 50.16960521272518}

'128259'

{'Device': 35,
 'Smallest Delta': 198,
 'Total Entries': 3925,
 'First': 23121389,
 'Last': 23906703,
 'Average': 200.08}

'128267'

{'Device': 35,
 'Smallest Delta': 998,
 'Total Entries': 785,
 'First': 23121916,
 'Last': 23905920,
 'Average': 998.7312101910828}

'128275'

{'Device': 35,
 'Smallest Delta': 995,
 'Total Entries': 785,
 'First': 23121918,
 'Last': 23905932,
 'Average': 998.743949044586}

'59904'

{'Device': 4,
 'Smallest Delta': 0,
 'Total Entries': 1983,
 'First': 23122854,
 'Last': 23906341,
 'Average': 395.10186585980836}

{'Device': 5,
 'Smallest Delta': 0,
 'Total Entries': 2459,
 'First': 23122904,
 'Last': 23906666,
 'Average': 318.73200488003255}

'126208'

{'Device': 4,
 'Smallest Delta': 1,
 'Total Entries': 1578,
 'First': 23123261,
 'Last': 23906338,
 'Average': 496.24651457541194}

{'Device': 3,
 'Smallest Delta': 20753,
 'Total Entries': 38,
 'First': 23123874,
 'Last': 23897362,
 'Average': 20354.947368421053}

'130821'

{'Device': 4,
 'Smallest Delta': 1,
 'Total Entries': 1442,
 'First': 23121738,
 'Last': 23905855,
 'Average': 543.7704576976422}

'130827'

{'Device': 4,
 'Smallest Delta': 1,
 'Total Entries': 939,
 'First': 23121451,
 'Last': 23905816,
 'Average': 835.3194888178914}

{'Device': 5,
 'Smallest Delta': 995,
 'Total Entries': 782,
 'First': 23122187,
 'Last': 23905924,
 'Average': 1002.2212276214834}

'130823'

{'Device': 5,
 'Smallest Delta': 20754,
 'Total Entries': 38,
 'First': 23123875,
 'Last': 23897363,
 'Average': 20354.947368421053}
```

[13]:
```
display('SpoofedMessage')

srcs = [4, 5]

for src in srcs:
    pgns = list()

    for pgn in n2k_log[n2k_log.src == src]["pgn"].unique():
        pgns.append(pgn)

    display(src, pgns)

# '59392': 'NMEA2000', # Acknowledge
# '59904': 'NMEA2000', # Request for Address Claimed
# '126208': 'NMEA2000', # Request Group Function
# '130821': 'Auto Pilot', # NavSource Speed (FEC)
# '130823': 'Auto Pilot', # Browser Control Status (FEC)
# '130827': 'Auto Pilot' # FURUNO Proprietary
```

```
'SpoofedMessage'


4


[130827, 130821, 59904, 126208, 59392]


5


[130827, 59392, 59904, 60928, 130823]
```

[14]:
```
def pinpoint(frames, timestamp):
    time_text = parse_timestamp(timestamp)
    frame = parse_frames(frames)
    display(time_text, frame)
```

[15]:
```
display("Pinpoint 1")

encoded_frame = "EAKTEwIS8QH/JOybDQAIIZ0DAAC6Av0bEAM="
timestamp = 1595528704747

pinpoint(base64.b64decode(encoded_frame), timestamp)
```

```
'Pinpoint 1'
```

```
'2020-07-23-18:25:04.747'
```

```
[{'start': b'\x10\x02',
  'cmd': 147,
  'len': 19,
  'priority': 2,
  'pgn': 127250,
  'dst': 255,
  'src': 36,
  'timestamp': 891884,
  'datalen': 8,
  'data': '21,9d,03,00,00,ba,02,fd',
  'crc': 27,
  'end': b'\x10\x03'}]
```

[16]:
```
display("Pinpoint 2")

frame =␣
 ↪b'\x10\x02\x93\x13\x02\x03\xf5\x01\xff\x23\x0c\x90\x08\x00\x08\xff\x5f\x01\xff\xff\x00\xff\
timestamp = 1596659812869

pinpoint(frame, timestamp)
```

```
'Pinpoint 2'
```

```
'2020-08-05-20:36:52.869'
```

```
[{'start': b'\x10\x02',
  'cmd': 147,
  'len': 19,
  'priority': 2,
  'pgn': 128259,
  'dst': 255,
  'src': 35,
  'timestamp': 561164,
  'datalen': 8,
  'data': 'ff,5f,01,ff,ff,00,ff,ff',
  'crc': 54,
  'end': b'\x10\x03'}]
```

[17]:
```
display("Pinpoint 3")
```

```
frame =␣
↪b'\x10\x02\x93\x13\x02\x01\xf8\x01\xff\x24\x1c\x23\x09\x00\x08\xff\xff\xff\x7f\xff\xff\xff\
timestamp = 1596659812869

pinpoint(frame, timestamp)
```

```
'Pinpoint 3'
```

```
'2020-08-05-20:36:52.869'
```

```
[{'start': b'\x10\x02',
  'cmd': 147,
  'len': 19,
  'priority': 2,
  'pgn': 129025,
  'dst': 255,
  'src': 36,
  'timestamp': 598812,
  'datalen': 8,
  'data': 'ff,ff,ff,7f,ff,ff,ff,7f',
  'crc': 243,
  'end': b'\x10\x03'}]
```