It WISN't me, attacking industrial wireless mesh networks

Introduction

- Erwin Paternotte
- Lead security consultant
- @stokedsecurity

- Mattijs van Ommeren
- Principal security consultant
- @alcyonsecurity

nixu

Industrial (r)evolution

A brief history of control systems:

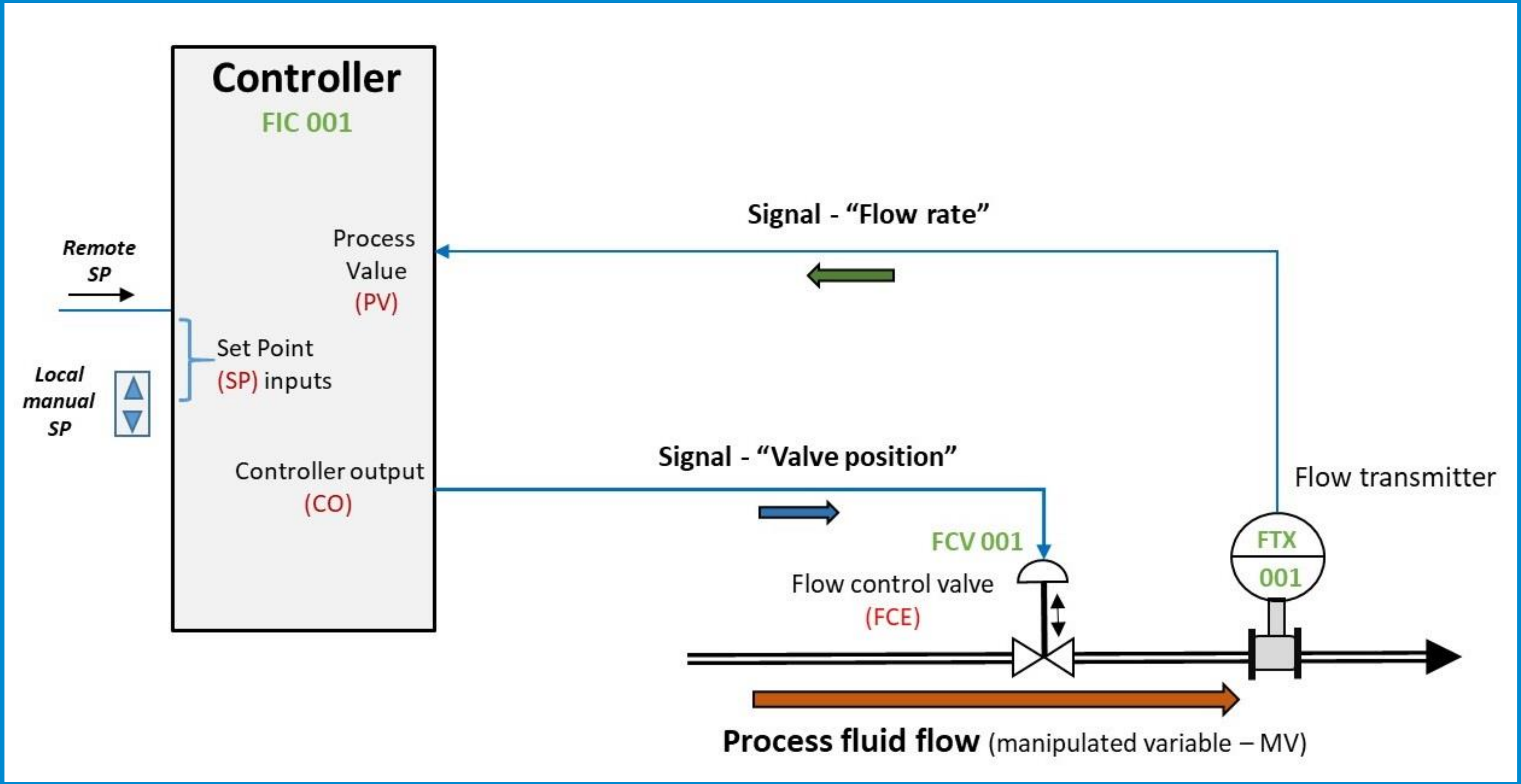
- ~1940: Air: Pneumatic logic systems: 3 - 15 psi
- Mid 1950: Analog: Current loop: 4 - 20 mA
- Mid 1980: Digital: HART, Fieldbus, Profibus
- Late 2000: Wireless mesh networks
  - WirelessHART (09/2007)
  - ISA 100.11a (09/2009)

Previous research

- Security considerations for the WirelessHART protocol, Shahid Raza et al, 2009
  - https://ieeexplore.ieee.org/document/5347043/
- WirelessHART A Security Analysis, Max Duijsens, Master (2015)
  - https://pure.tue.nl/ws/files/47038470/800499-1.pdf
- Attacking the plant through WirelessHART, Mattijs & Erwin, S4 Miami (2016)
  - https://www.youtube.com/watch?v=AlEpgutwZvc
- Denial of service attacks on ICS wireless protocols, Blake Johnson, S4 Miami (2018)
  - https://github.com/voteblake/DIWI/ (video no longer available)

Wright's principle: "Security does not improve until practical tools for exploration of the attack surface are made available."

Industrial process control loop



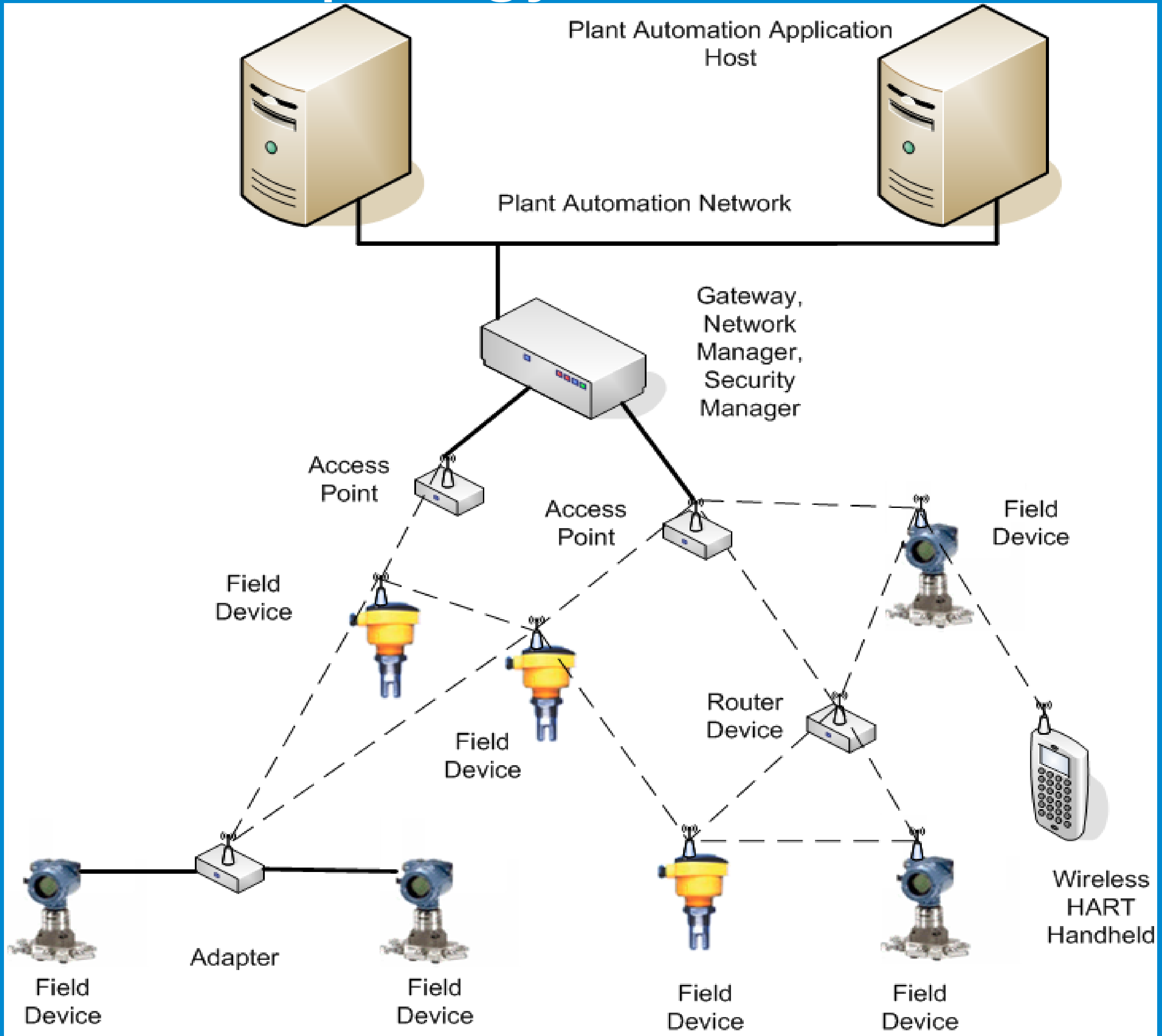
Introduction to WirelessHART

- Supports HART application layer
- Single encryption cipher/key length (AES CCM*)
- Wireless technology based on Time Synced Mesh Protocol developed by Dust Networks (now part of Analog Devices)
- Radio SoC exclusively provided by Dust Networks

Introduction to ISA 100.11a

- Relies on several standards: 6LoWPAN (IPv6/UDP)
- Ability to tunnel other protocols
- Mainly developed by Nivis
- Generic 802.15.4 chips provided by multiple vendors: STM, NXP, Texas Instruments, OKI

WISN topology



nixu

Protocol stacks

OSI	HART	WirelessHART	ISA100.11a
Application	Command oriented, predefined data types and application procedures (HART and WirelessHART)		ISA native or legacy protocols (tunneling)
Presentation			
Session			
Transport	Auto-segmented transfer of large data sets, reliable stream transport (HART and WirelessHART)		UDP
Network		Mesh network	Mesh network/6LoWPAN
Datalink	Byte oriented, token, master/slave protocol	Upper data-link sublayer IEEE 802.15.4 MAC	Upper data-link sublayer IEEE 802.15.4 MAC
Physical	Analog & digital signaling (4-20 mA)	IEEE 802.15.4 PHY (2.4 GHz)	IEEE 802.15.4 PHY (2.4 GHz)

Common denominators

- 802.15.4 MAC layer at 2.4 Ghz
- Time Slotted Channel Hopping in order to:
  - Minimize interference with other radio signals
  - Mitigate multipath fading
- Centralized network & security manager orchestrates communication between nodes
- Concluded that developing a common sniffer for both protocols should be possible

WirelessHART & ISA100.11a Security

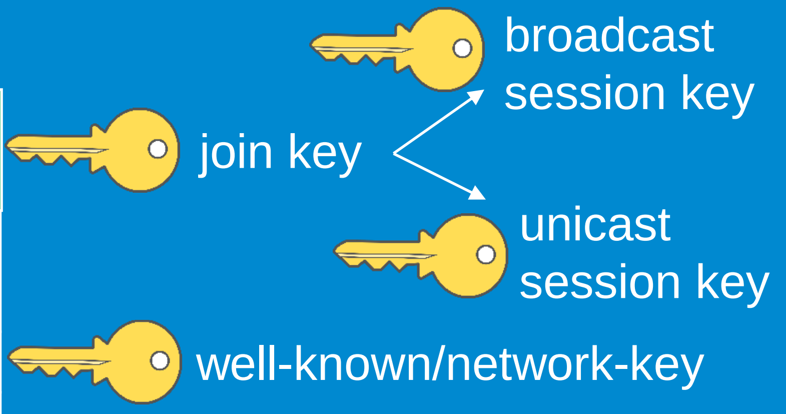
- AES CCM* (CBC-MAC with counter mode)
  - Datalink Layer (integrity only)
  - Transport Layer (encryption)
- Join process
  - Handshake with Network Manager
    - Shared secrets
    - Certificates (ISA100.11.a only)

Keys galore

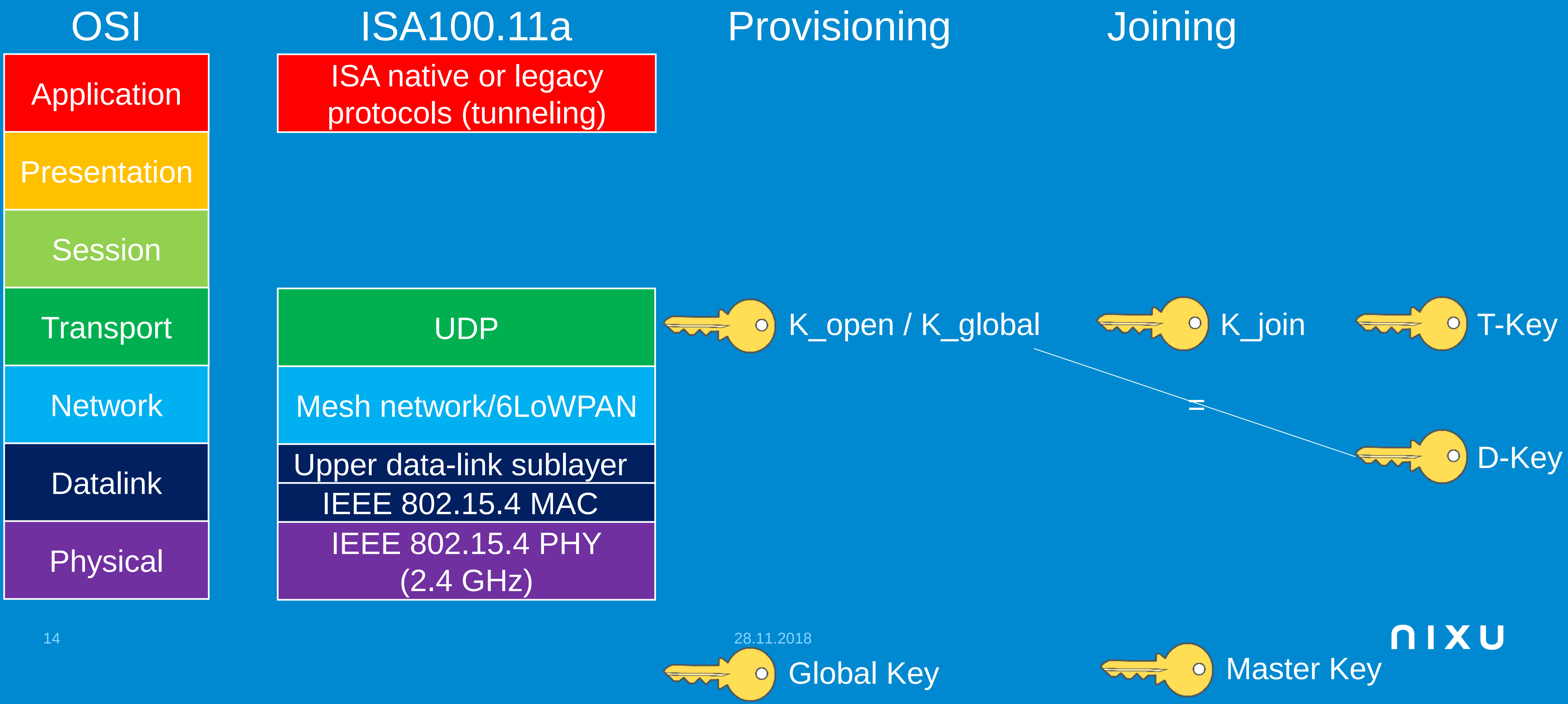
- ISA100.11a
  - **Global Key** – well-known
  - **K_open** – well-known
  - **K_global** – well-known
  - **Master Key** – derived during provisioning, used as KEK
  - **K_join** – Join process
  - **D-Key** – Hop-by-hop integrity
  - **T-KEY** – End-to-end encryption
- WirelessHART
  - **Well-known Key** – Advertisements
  - **Network Key** – Hop-by-hop integrity
  - **Join Key** – Join process
  - **Broadcast Session Key** – End-to-end
  - **Unicast Session Key** – End-to-end

WirelessHART encryption keys

OSI	WirelessHART
Application	Command oriented, predefined data types and application procedures
Presentation	
Session	
Transport	Auto-segmented transfer of large data sets, reliable stream transport
Network	Mesh network
Datalink	Upper data-link sublayer IEEE 802.15.4 MAC
Physical	IEEE 802.15.4 PHY (2.4 GHz)



ISA100.11a encryption keys



How to obtain key material

- Default keys
  - Documented, more or less
- Sniffing
  - During OTA provisioning (ISA100.11a)
- Keys stored in device NVRAM
  - Recoverable through JTAG/SPI (as demonstrated by our previous research)

WirelessHART default join keys

- **445553544E4554574F524B53524F434B** – Multiple vendors
  - DUSTNETWORKSROCK
- **E090D6E2DADACE94C7E9C8D1E781D5ED** – Pepperl+Fuchs
- **24924760000000000000000000000000** – Emerson
- **456E647265737320B20486175736572** – Endress+Hauser
  - Endress + Hauser

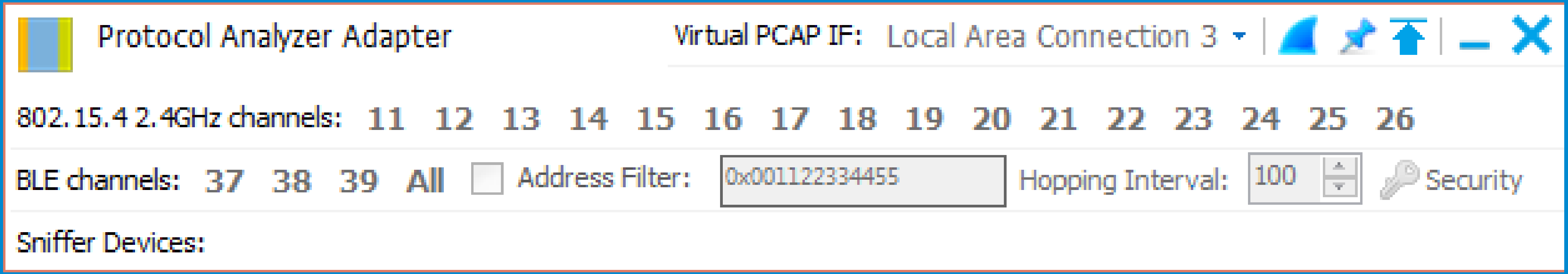
Sniffer hardware selection

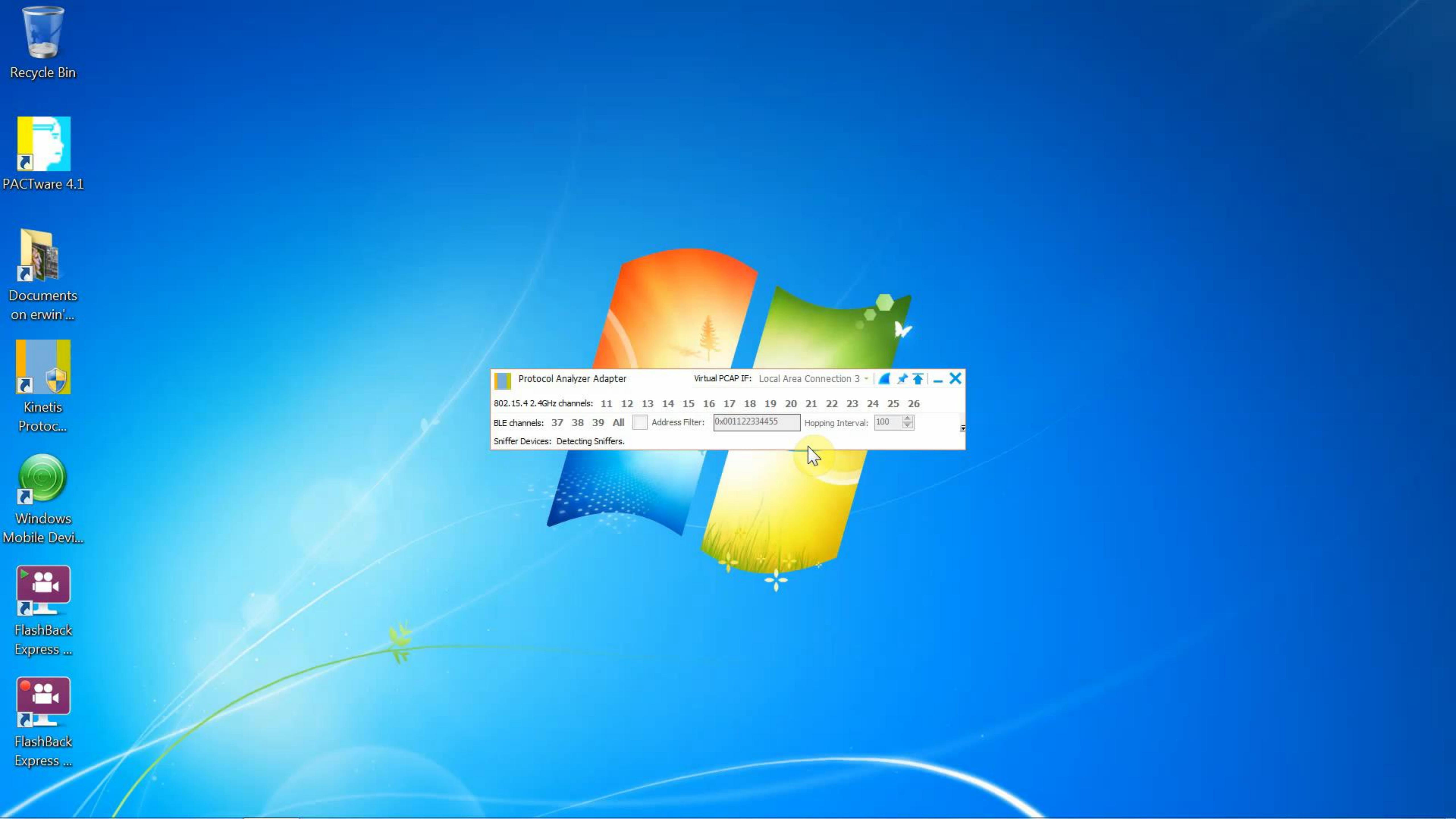
- BeamLogic 802.15.4 Site Analyzer
  - 16 channels simultaneously, no injection support, Basic Wireshark dissector, Expensive (~ $1300)
- Atmel RZ Raven
  - Single channel 802.15.4 with standard firmware, no free IDE (Atmel Studio n/a), reached EOL
- NXP BeeKit
  - Single channel 802.15.4 with standard firmware (not open source), reached EOL

NXP USB-KW41Z

- Single channel 802.15.4 with standard firmware (not open source)
- Actively supported
- Free IDE available
- Powerful microcontroller (Cortex M0+)
- PCB ready for external antenna (Wardriving!)
- Easy firmware flashing via USB mass storage (OpenSDA)
- Documentation and examples, but with a few important omissions

Demo 1: Kinetix Protocol Analyzer Adapter (sniffer)

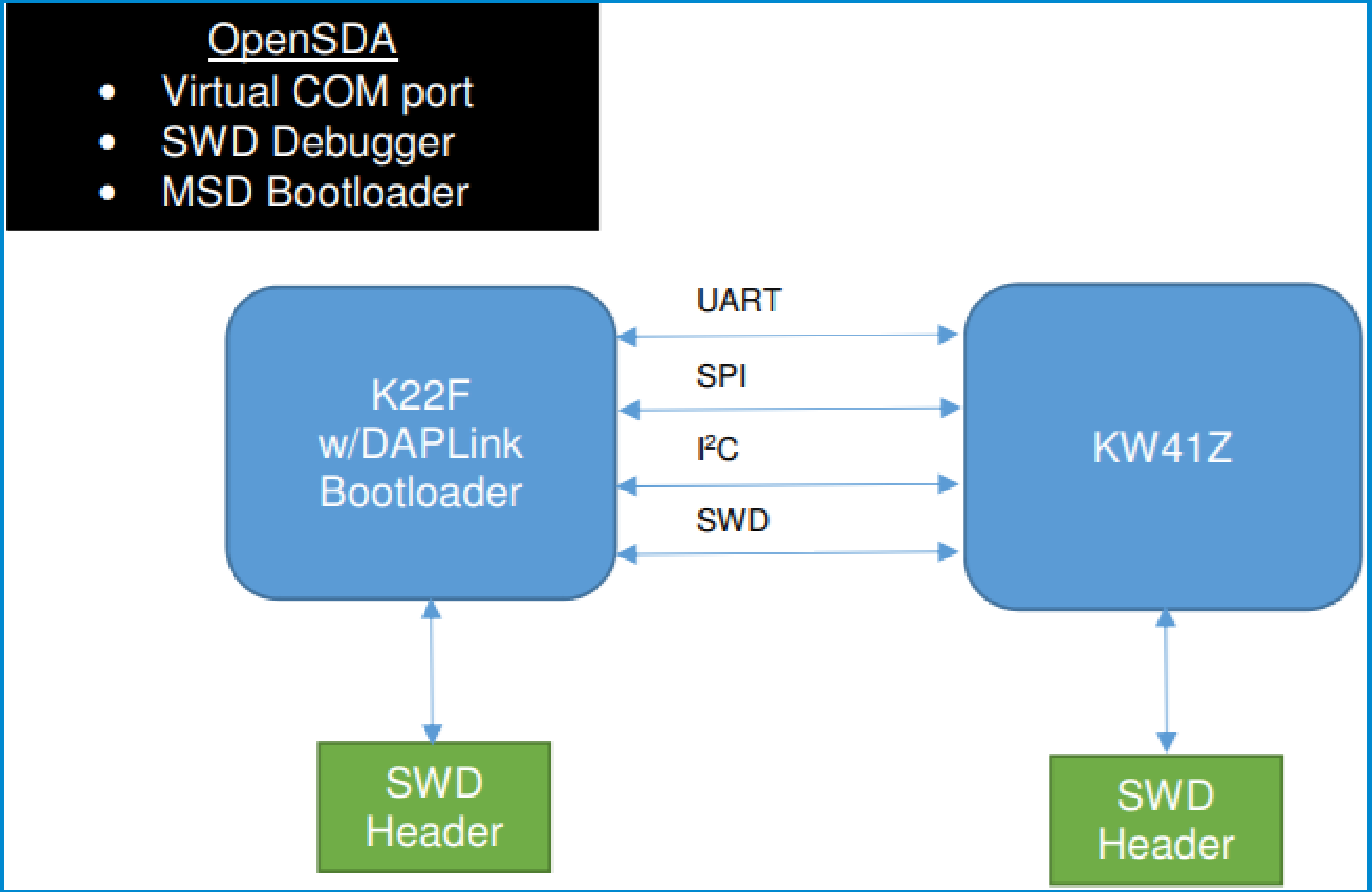



Recycle Bin
PACTware 4.1
Documents on erwin'...
Kinetis Protoc...
Windows Mobile Devi...
FlashBack Express ...
FlashBack Express ...
Protocol Analyzer Adapter
Virtual PCAP IF: Local Area Connection 3
802.15.4 2.4GHz channels: 11 12 13 14 15 16 17 18 19 20 21 22 23 24 25 26
BLE channels: 37 38 39 All
Address Filter: 0x001122334455
Hopping Interval: 100
Sniffer Devices: Detecting Sniffers.

USB-KW41Z <-> host communication

- Hardware is detected as virtual COM/UART port (Windows/Linux)
- Freescale Serial Communication Interface (FSCI) developed by NXP for communication between host and device firmware.
- Host SDK for FSCI is available (with Python bindings)
- FSCI protocol is fairly well documented
- Allowed us to communicate directly with the USB-KW41Z without requiring the SDK to be installed

USB-KW41Z block diagram

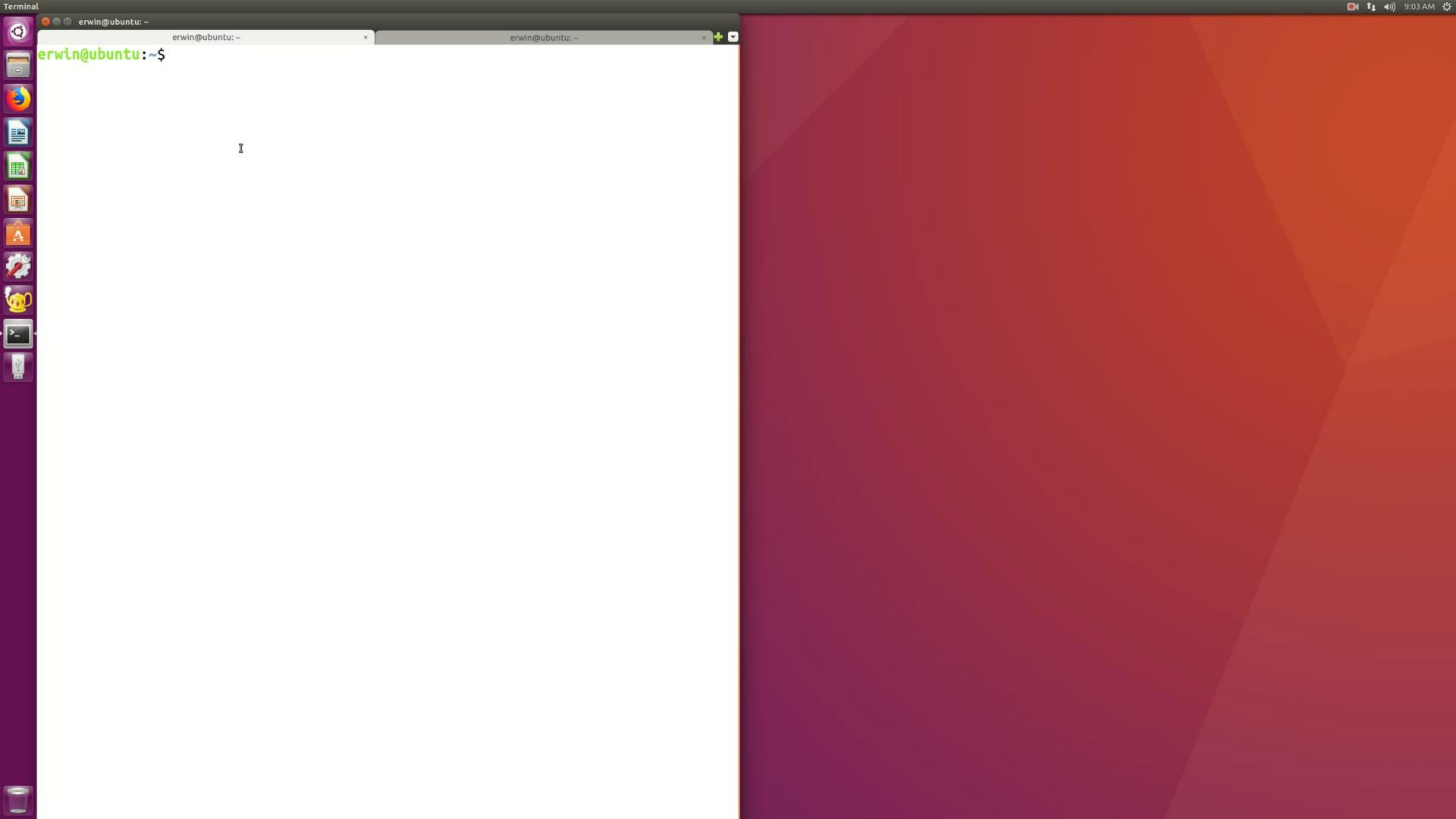


Building the toolset

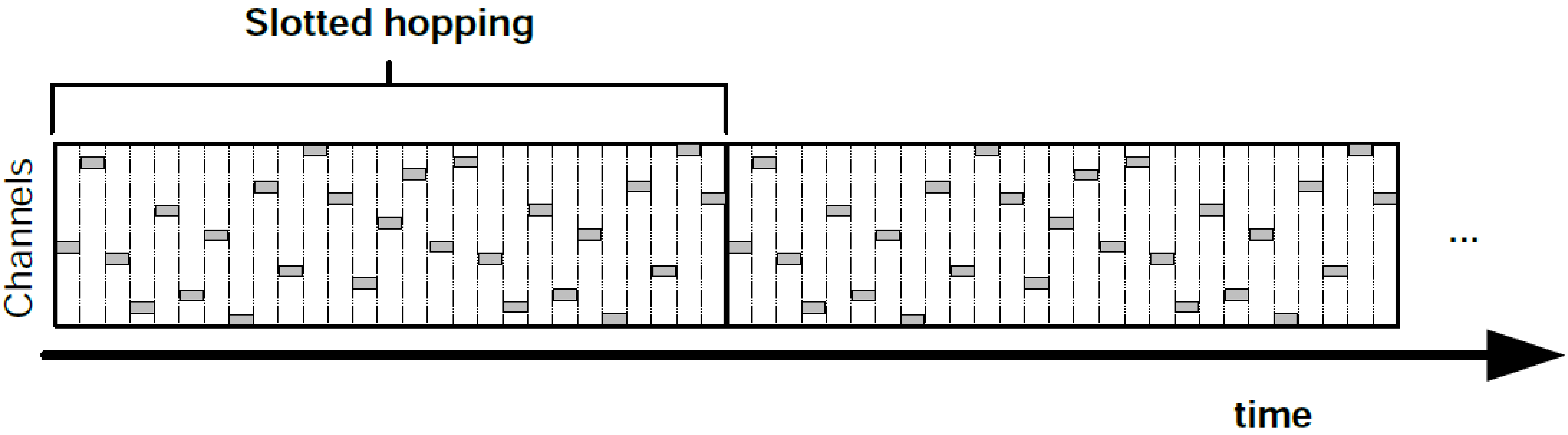
- Extended the KillerBee framework with a driver for the USB-KW41Z
  - Allows us to comfortably capture 802.15.4 traffic into PCAP format
- Developed Scapy protocol support
  - Allows us to forge and inject packets
- Developed Wireshark dissectors for WirelessHART and ISA100.11a
  - Bringing WISN packet viewing to the masses
  - Live capture and dissecting of WISN traffic on a single channel at the time

Demo 2: Sniffing traffic with KillerBee and Wireshark

```
Resetting CPU...
Command:
0000:  02 a3 08 00 ab                                           .....

Command:
0000:  02 85 09 12 52 00 01 00 00 00 00 00 00 00 00 00          ....R...........
0010:  00 00 00 00 00 00 cd                                     .......

Response:  Packet | group: 84, opcode: 0d, crc: d8 ok:True
0000:  00 52 00                                                 .R.

Set channel: 19
Channel set to: 19
Command:
0000:  02 85 09 12 21 00 13 00 00 00 00 00 00 00 00 00          ....!...........
0010:  00 00 00 00 00 00 ac                                     .......

Response:  Packet | group: 84, opcode: 0d, crc: ab ok:True
0000:  00 21 00                                                 .!.

Command:
0000:  02 85 09 12 51 00 01 00 00 00 00 00 00 00 00 00          ....Q...........
0010:  00 00 00 00 00 00 ce                                     .......

Response:  Packet | group: 84, opcode: 0d, crc: db ok:True
0000:  00 51 00                                                 .Q.

Command:
0000:  02 ba 01 00 bb                                           .....

Response:  Packet | group: a4, opcode: fe, crc: ac ok:True
0000:  f7                                                       .

zbwireshark: listening on '/dev/ttyACM0'
```


Terminal
erwin@ubuntu: ~
erwin@ubuntu: ~
erwin@ubuntu: ~
erwin@ubuntu:~$
9:03 AM

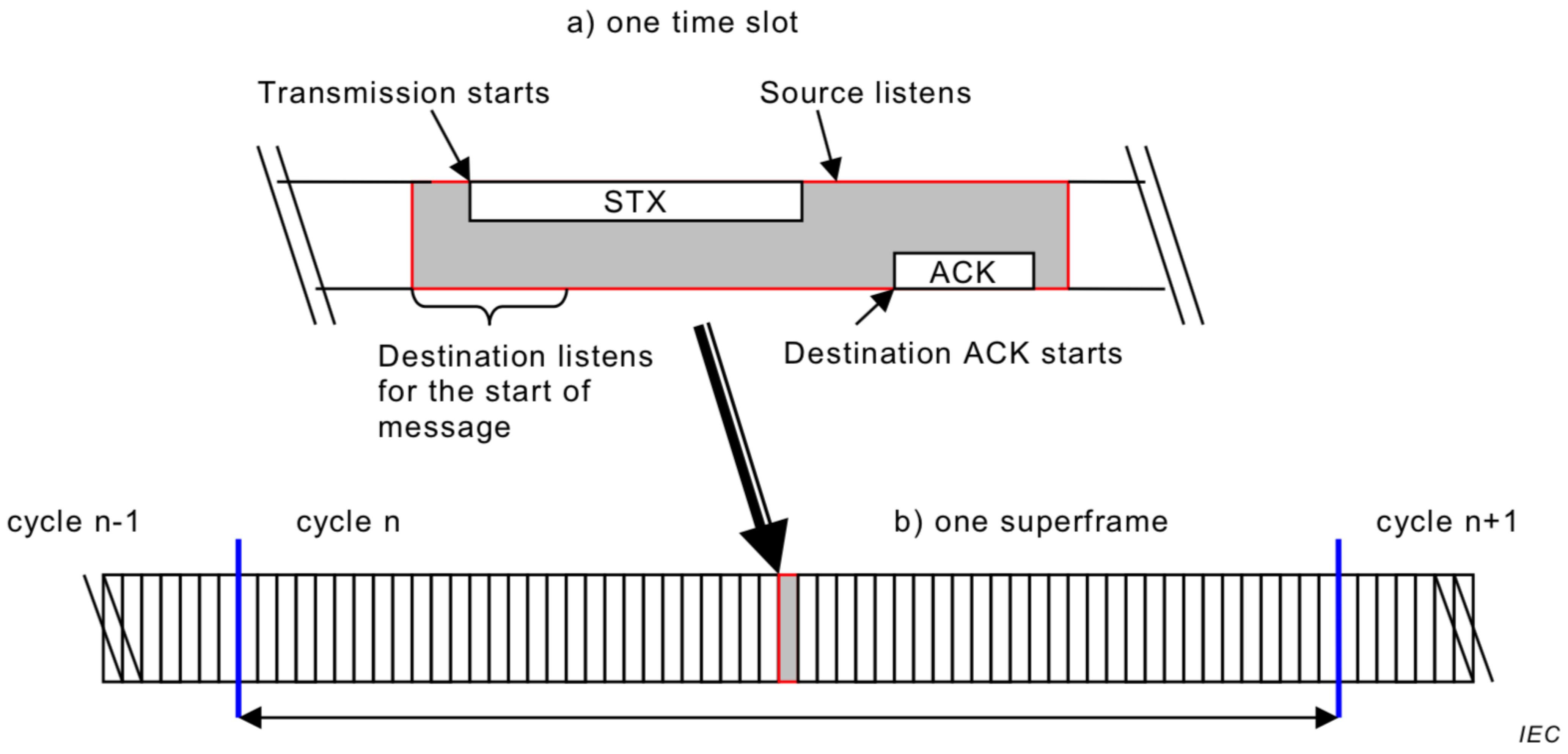
Time Slotted Channel Hopping



nixu

Superframe

- Sequence of repeating channel hopping patterns
- Period usually between 512-4096 time slots
- Time reference
  - WirelessHART: sequence number=0 (start of network manager)
  - ISA100: TAI=0 (Jan 1st 1958, 00:00:00)
- Timeslot within a superframe denotes a communication link, assigned by the Network Manager


a) one time slot
Transmission starts
Source listens
STX
ACK
Destination listens for the start of message
Destination ACK starts
cycle n-1
cycle n
b) one superframe
cycle n+1
IEC

Implementing Time Slotted Channel Hopping

- Both protocols require high speed channel hopping via predefined, but different patterns.
- FSCI communication too slow to tune into time slots (10ms)
  - **Solution: implement channel hopping in firmware**
- Two layers of encryption/authentication
  - **Solution: Implement in host software (Killerbee)**
- Ability to inject traffic
  - FSCI supports injection of arbitrary frames
  - **Solution: Implement frame injection in Killerbee, add protocol support to Scapy for crafting packets**

Firmware

Bare metal task scheduler

- Task consisting of single (endless) loop
- Blocking function waiting for events
- Once a task is running, it has full control
- Cannot run longer than ~2 ms to prevent starvation of other tasks

```
void MyTask (uint32_t param) {
    osaEventFlags_t ev;

    while(1) {
        OSA_EventWait(mAppEvent,
          osaEventFlagsAll_c, FALSE,
          osaWaitForever_c, &ev);
        if( ev && gSomeEvent) {
            /* do stuff */
            break;
        }
        break;
        ...
}
```


nixu

Bare Metal vs. RTOS

- Most RTOS use pre-emptive task scheduling
  - Nice for hard real-time requirements but:
    - Relatively large overhead
    - Context switches
    - Deal with synchronization issues
- Bare Metal uses cooperative multi tasking
  - Dependent on other tasks behaving nicely
  - Can avoid most synchronization issues
  - Faster execution

Firmware
Tasks/components

- Framework
  - Memory Manager
  - MAC/PHY
  - Serial Manager
  - Timers
  - LED driver
  - FSCI
- Application
  - 802.15.4 MAC extension layer
    - Source/destination/PAN info
  - ISA100/WirelessHART
    - Extract link information
      - Timeslots, channels
    - Timeslot synchronization
    - Channel hopping

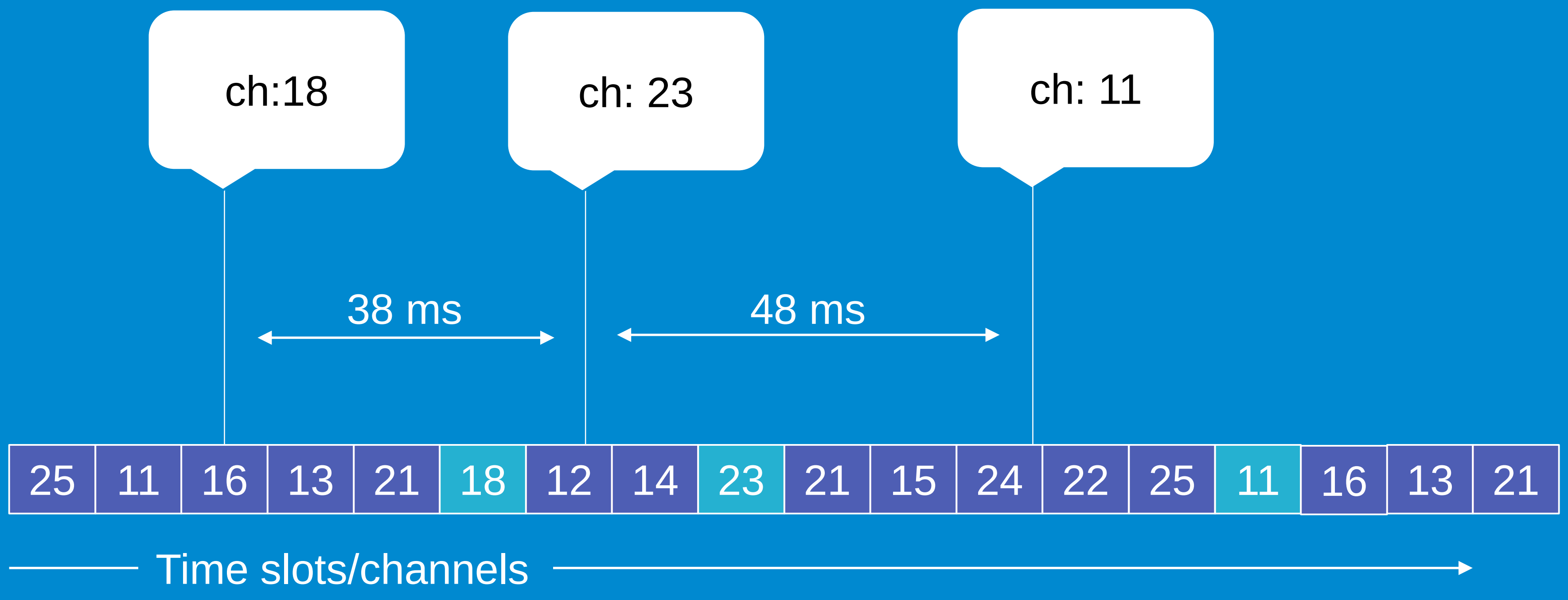
How to synchronize?

- Both protocols support advertisement packets
- Broadcast by network manager
- Contains information about free join slots
- Timing information to synchronize on
- Hopping patterns are documented in protocol specifications

Channel selection

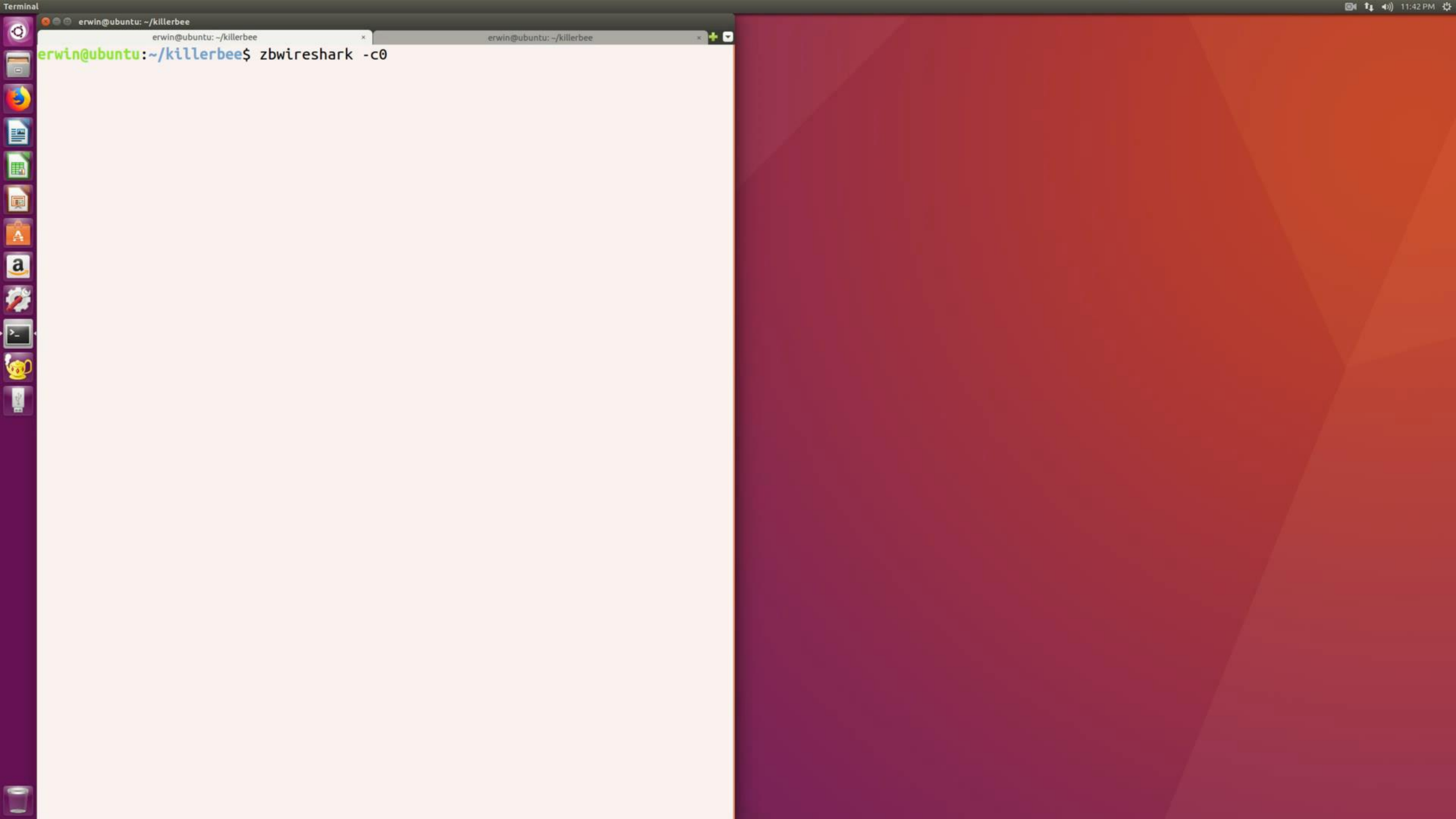
- WirelessHART
  - ActiveChannel = (Channel_Offset + absolute slot number) % number of active channels
  - Channel = ActiveChannelArray [ActiveChannel]
- ISA100
  - ActiveChannel = (absolute slot number – ChBirth) % ChCycle
  - ChBirth = Channel number assigned at t0 (International Atomic Time, TAI)
  - ChCycle = Number of channels x channel rate

Channel hopping
Scheduling



Demo 3: Sniffing with channel hopping

```
Debug: ***** WirelessHART - asn:626928 ch:12
Packet 15 | ch:0 rssi:111 time: 1388081 len: 62 crc:True
0000:  41 88 30 90 06 ff ff 01 00 31 00 00 09 91 30 11   A.0......1....0.
0010:  0f ff 7f 00 00 03 01 01 00 01 00 aa 02 00 04 00   ................
0020:  01 01 e9 45 04 00 80 06 00 12 4d 00 15 4d 00 3c   ...E......M..M.<
0030:  4d 00 42 4d 00 61 4d 00 62 4d 78 35 2a d9 1b c0   M.BM.aM.bMx5*...

Debug: ***** WirelessHART - asn:626992 ch:16
Packet 16 | ch:0 rssi:121 time: 1508080 len: 62 crc:True
0000:  41 88 f0 90 06 ff ff 01 00 31 00 00 09 91 f0 11   A........1......
0010:  0f ff 7f 00 00 03 01 01 00 01 00 aa 02 00 04 00   ................
0020:  01 01 e9 45 04 00 80 06 00 12 4d 00 15 4d 00 3c   ...E......M..M.<
0030:  4d 00 42 4d 00 61 4d 00 62 4d 6a 97 dd a5 07 47   M.BM.aM.bMj....G

Debug: ***** WirelessHART - asn:627184 ch:13
Packet 17 | ch:0 rssi:106 time: 1548079 len: 62 crc:True
0000:  41 88 30 90 06 ff ff 01 00 31 00 00 09 92 30 11   A.0......1....0.
0010:  0f ff 7f 00 00 03 01 01 00 01 00 aa 02 00 04 00   ................
0020:  01 01 e9 45 04 00 80 06 00 12 4d 00 15 4d 00 3c   ...E......M..M.<
0030:  4d 00 42 4d 00 61 4d 00 62 4d 77 ee 52 79 e4 22   M.BM.aM.bMw.Ry."
```

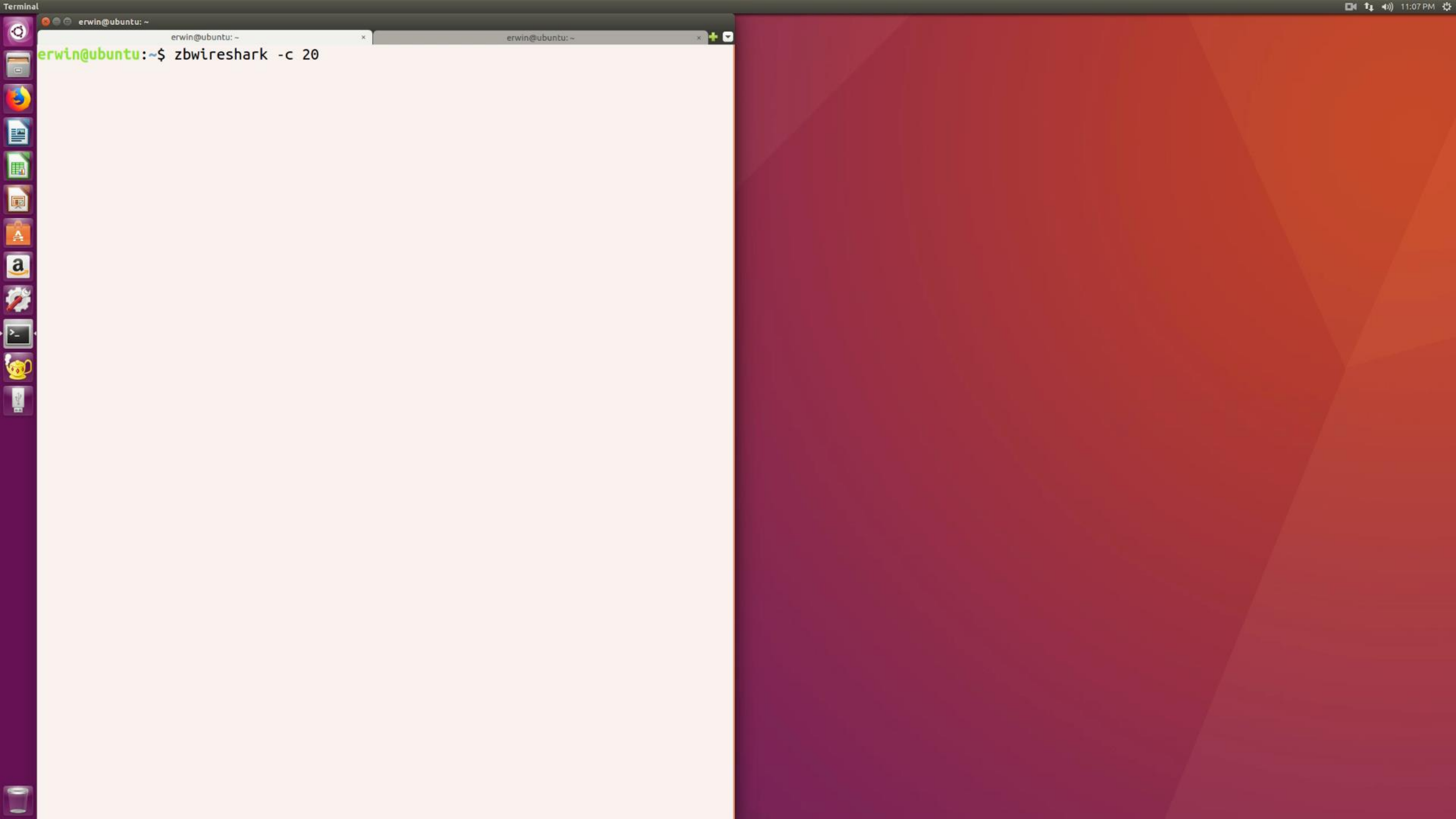

Terminal
erwin@ubuntu: ~/killerbee
erwin@ubuntu: ~/killerbee
erwin@ubuntu: ~/killerbee
erwin@ubuntu:~/killerbee$ zbwireshark -c0
11:42 PM

Unauthenticated attacks

- Signal jamming through continuous power emission
- Concurrent packet transmission
  - Join slot jamming
  - Selective jamming transmitter communication
  - Transmitting fake advertisements

Demo 4: Advertisement jamming

```
mvo@mvo-virtual-machine: ~
mvo@mvo-virtual-machine:~$ whjammer -c 20
Resetting CPU...
Tuning to channel 20
Start jamming channel 20
Enabling jammer
FSCIPacket | ch: 0 group: bb, opcode: 02, crc: b9 ok:True

Packet 1 | ch:0 rssi:108 time: 385612 len: 62 crc:True
0000:  41 88 90 90 06 ff ff 01 00 31 00 00 28 d9 90 11
0010:  0f ff 7f 00 00 03 01 01 00 01 00 cc 08 00 04 00
0020:  01 03 da 48 04 00 80 06 00 03 4d 00 05 4d 00 0e
0030:  4d 00 25 4d 00 27 4d 00 51 4d 7d e4 76 71 67 ff

Debug: Tracking network PAN ID: 1680
```


Terminal
erwin@ubuntu: ~
erwin@ubuntu: ~
erwin@ubuntu: ~
erwin@ubuntu:~$ zbwireshark -c 20
11:07 PM

Authenticated attacks

- Nonce exhaustion
  - Both protocols use a semi-predictable nonce counter to feed the AES CCM* algorithm
  - A device will reject a packet if a nonce value is lower than a previously received one
  - Spoofing a packet with a maximum nonce value, causes legitimate packets to drop
- Sending spoofed measurements to influence the process

Conclusions

- Still a large unexplored attack surfaces due to complexity of the protocols
- The released tools and research will fill this gap and enable security researchers to move forward in the field of WISN research
- Using WISN technology for process control and especially functional safety applications is probably not a good idea, and should be reconsidered

Future research

- Expand tool with more theorized attacks
- Research forced rejoin triggers
- Mapping WISN locations (wardriving)
- Implementation specific vulnerabilities (transmitters, gateways)

Questions & thank you

- Our code is soon available at: https://github.com/nixu-corp

- Thanks to the following people for their help:
  - Alexander Bolshev (@dark_k3y)
  - Sake Blok (@SYNbit)