Portland State UNIVERSITY

OpenWSN: The Open-Source Wireless Sensor Network

Natalie Kashoro, Undergraduate Researcher
David C. Burnett, Ph.D., Assistant Professor
Wireless Environmental Sensor Technology (WEST) Lab -- https://ece.pdx.edu/~dburnett
Dept. of Electrical and Computer Engineering, Portland State University, Portland, OR, USA
nkashoro@pdx.edu, dburnett@pdx.edu

Teardown, Portland OR, June 21, 2024

What is OpenWSN(.org)?

- The Open Wireless Sensor Network
- A complete network stack with reliability features at every level
- Firmware that can run on dozens of different microcontrollers
    - uC/OS-II, FreeRTOS, RIOT, OpenOS
- A wireless mesh that has been demonstrated with hundreds of nodes
- An ancient codebase! Python 2.7...
    - MANY contributors over the years! We (Natalie and David) are just users

Figures on following slides from OpenWSN.org's "Learn" section unless otherwise noted

What does it look like?

TRANSACTIONS ON EMERGING TELECOMMUNICATIONS TECHNOLOGIES
Trans. Emerging Tel. Tech. 2012; **23**:480–493
Published online in Wiley Online Library (wileyonlinelibrary.com). DOI: 10.1002/ett.2558

RESEARCH ARTICLE

OpenWSN: a standards-based low-power wireless development environment

Thomas Watteyne[1,2*], Xavier Vilajosana[1,3], Branko Kerkez[4,1], Fabien Chraim[1], Kevin Weekly[1], Qin Wang[1,5], Steven Glaser[4] and Kris Pister[1]

[1] BSAC, University of California, Berkeley, CA, USA
[2] Dust Networks/Linear Technology, Hayward, CA, USA
[3] Universitat Oberta de Catalunya, Barcelona, Spain
[4] Civil and Environmental Engineering, University of California, Berkeley, CA, USA
[5] University of Science and Technology, Beijing, China

ABSTRACT

The OpenWSN project is an open-source implementation of a fully standards-based protocol stack for capillary networks, rooted in the new IEEE802.15.4e Time Synchronized Channel Hopping standard. IEEE802.15.4e, coupled with Internet of Things standards, such as 6LoWPAN, RPL and CoAP, enables ultra-low-power and highly reliable mesh networks, which are fully integrated into the Internet. The resulting protocol stack will be cornerstone to the upcoming machine-to-machine revolution.

This article gives an overview of the protocol stack, as well as key integration details and the platforms and tools developed around it. The pure-C OpenWSN stack was ported to four off-the-shelf platforms representative of hardware currently used, from older 16-bit microcontroller to state-of-the-art 32-bit Cortex-M architectures. The tools developed around the low-power mesh networks include visualisation and debugging software, a simulator to mimic OpenWSN networks on a PC, and the environment needed to connect those networks to the Internet.

Experimental results presented in this article include a network where motes operate at an average radio duty cycle well below 0.1% and an average current draw of 68 μA on off-the-shelf hardware. These ultra-low-power requirements enable a range of applications, with motes perpetually powered by micro-scavenging devices. OpenWSN is, to the best of our knowledge, the first open-source implementation of the IEEE802.15.4e standard.

***Correspondence**
T. Watteyne, BSAC, 403 Cory Hall 1774, University of California Berkeley, CA 94720-1774, USA.
E-mail: watteyne@eecs.berkeley.edu

Received 22 June 2012; Revised 27 June 2012; Accepted 27 June 2012

Stacks you may already be familiar with

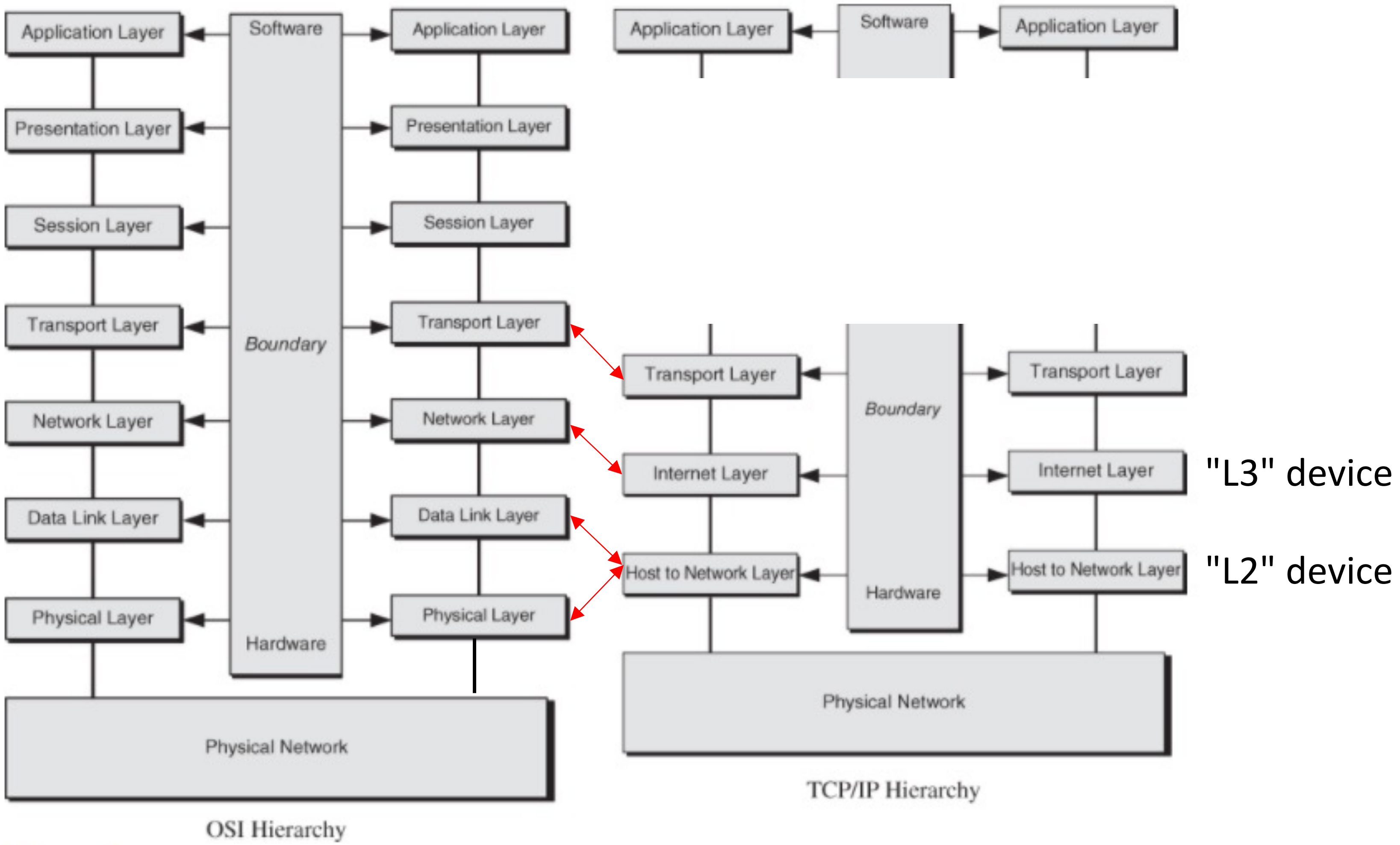
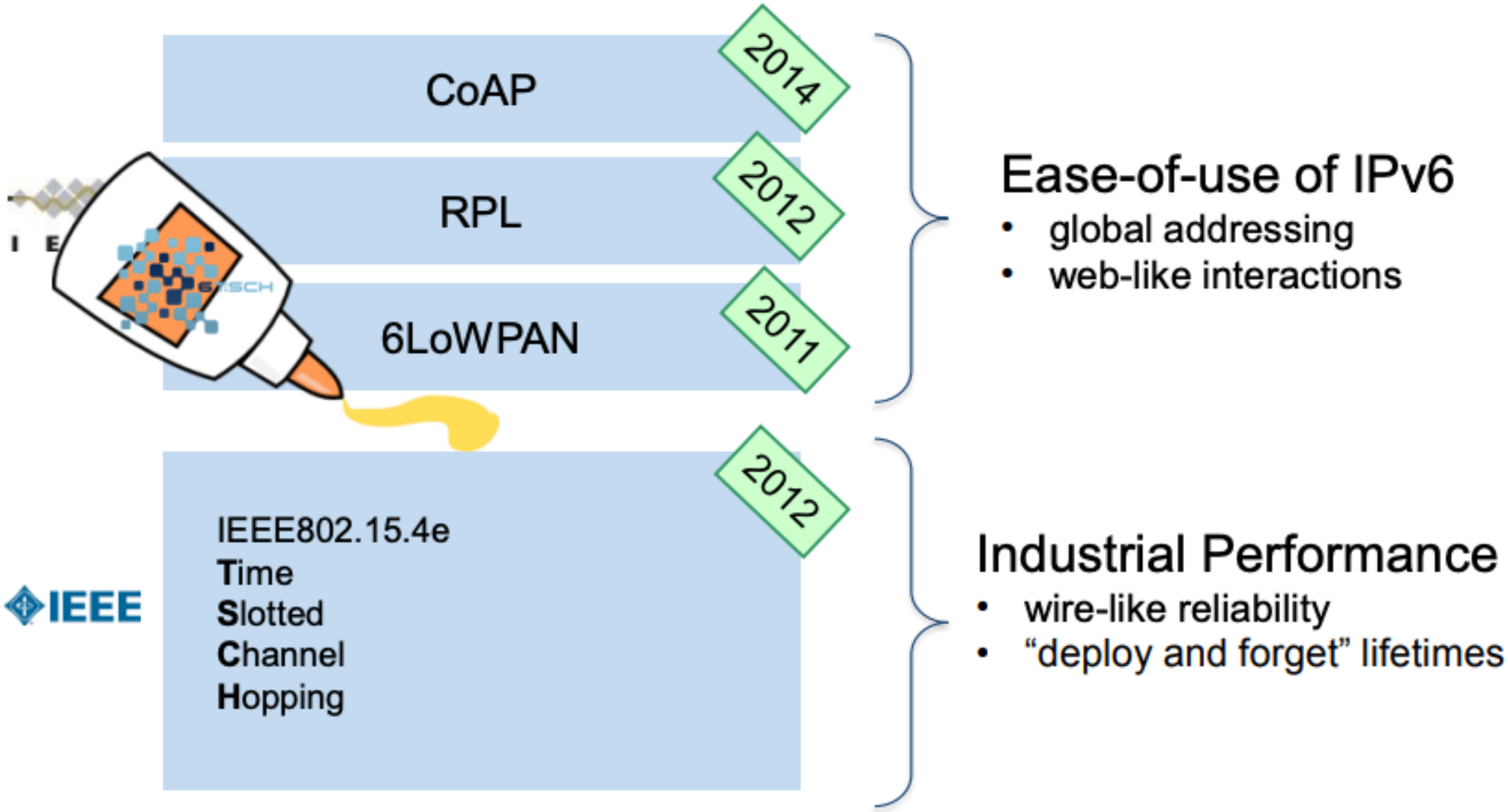


Figure 16.43 The Network Architecture for the OSI and TCP/IP Models

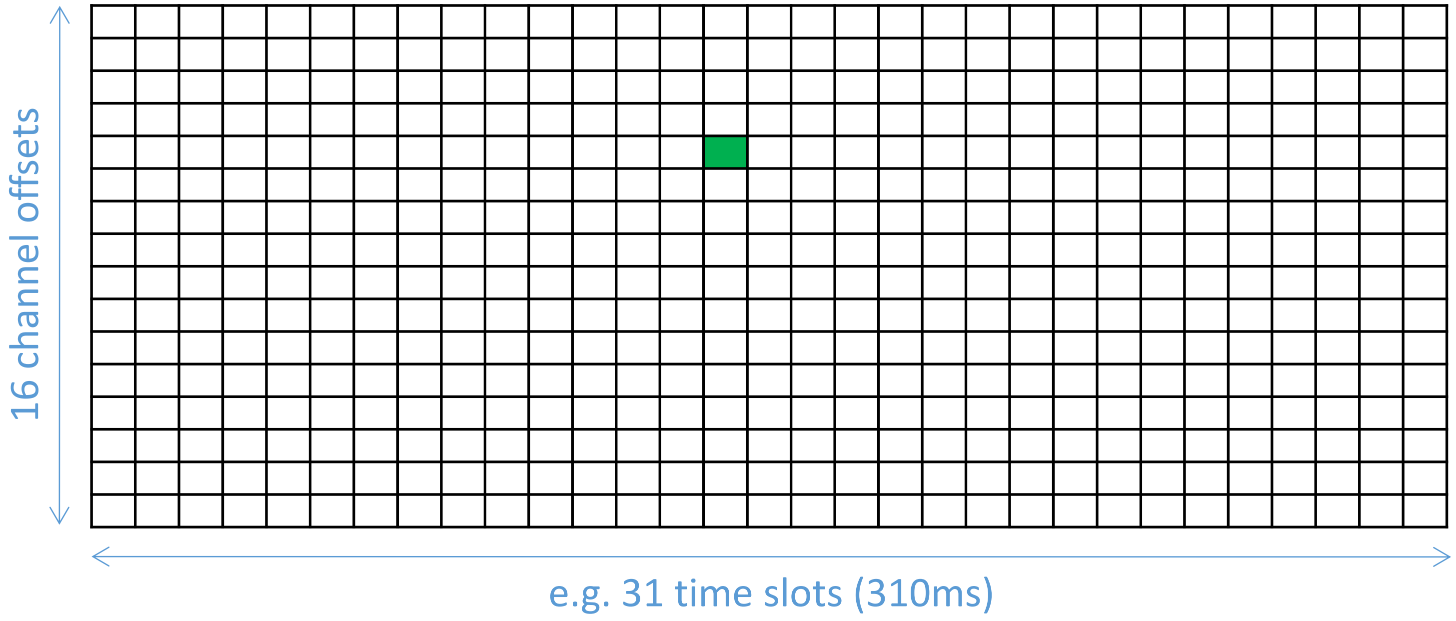
Figure adapted from J. Peckol, “Embedded Systems” 2nd ed, Wiley

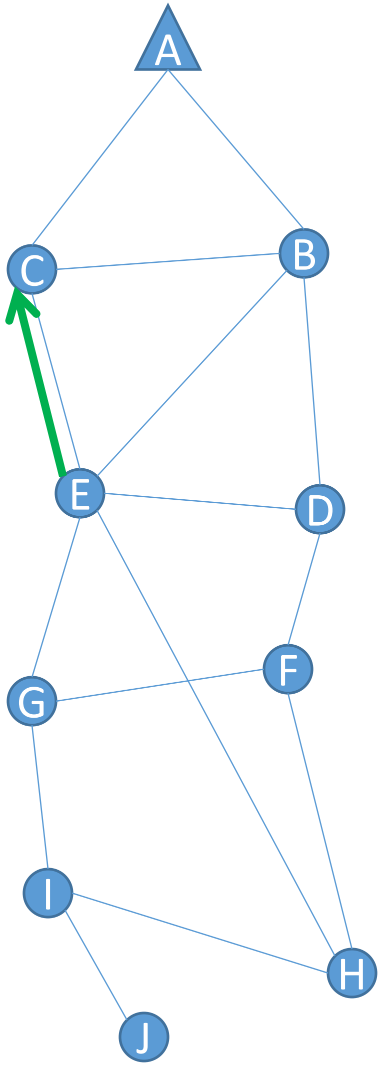
Each layer standardized over the years



Time/channel schedule: superframe

- A superframe repeats over time
  - Number of slots in a superframe is tunable
  - Each cell can be assigned to a pair of motes, in a given direction

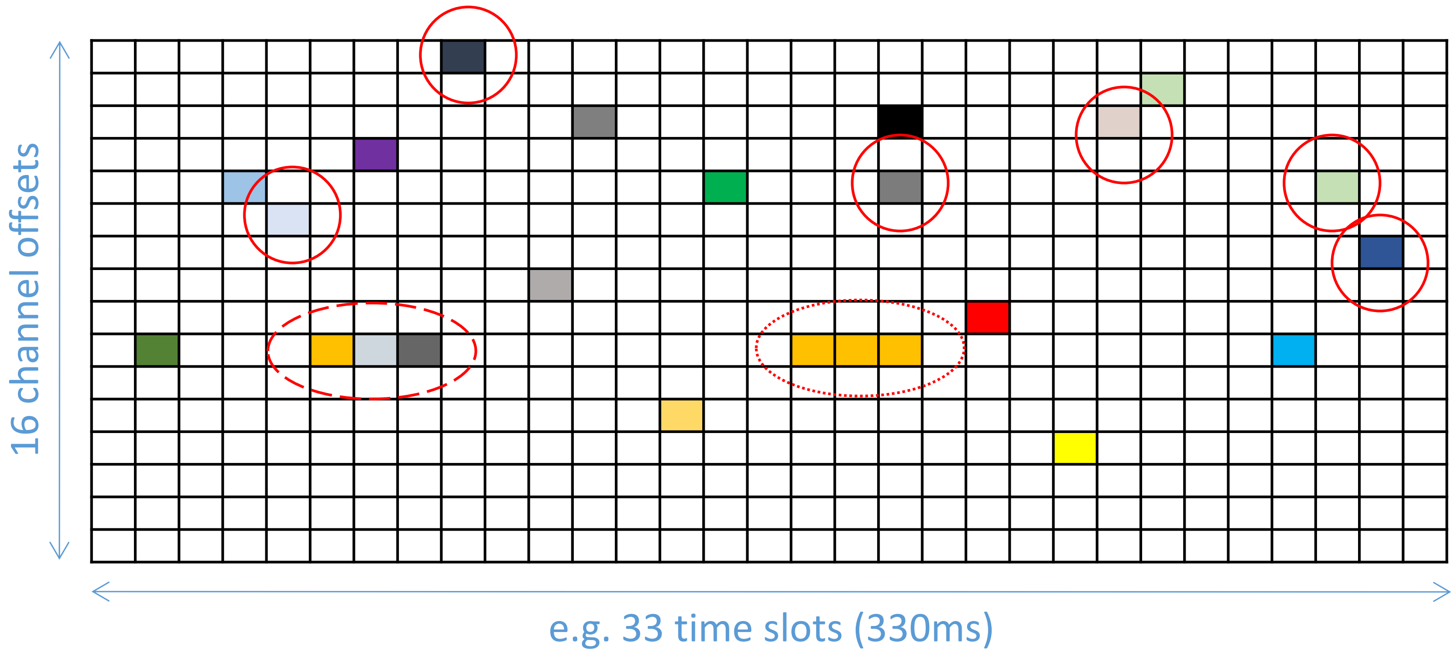


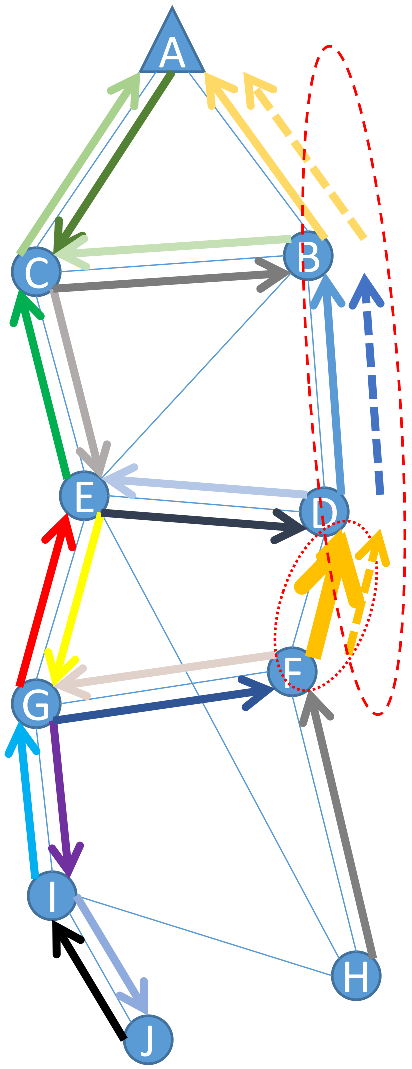


Building the schedule

- Cells are assigned according to application requirements
- Tunable trade-off between
  - packets/second
  - latency
  - robustness

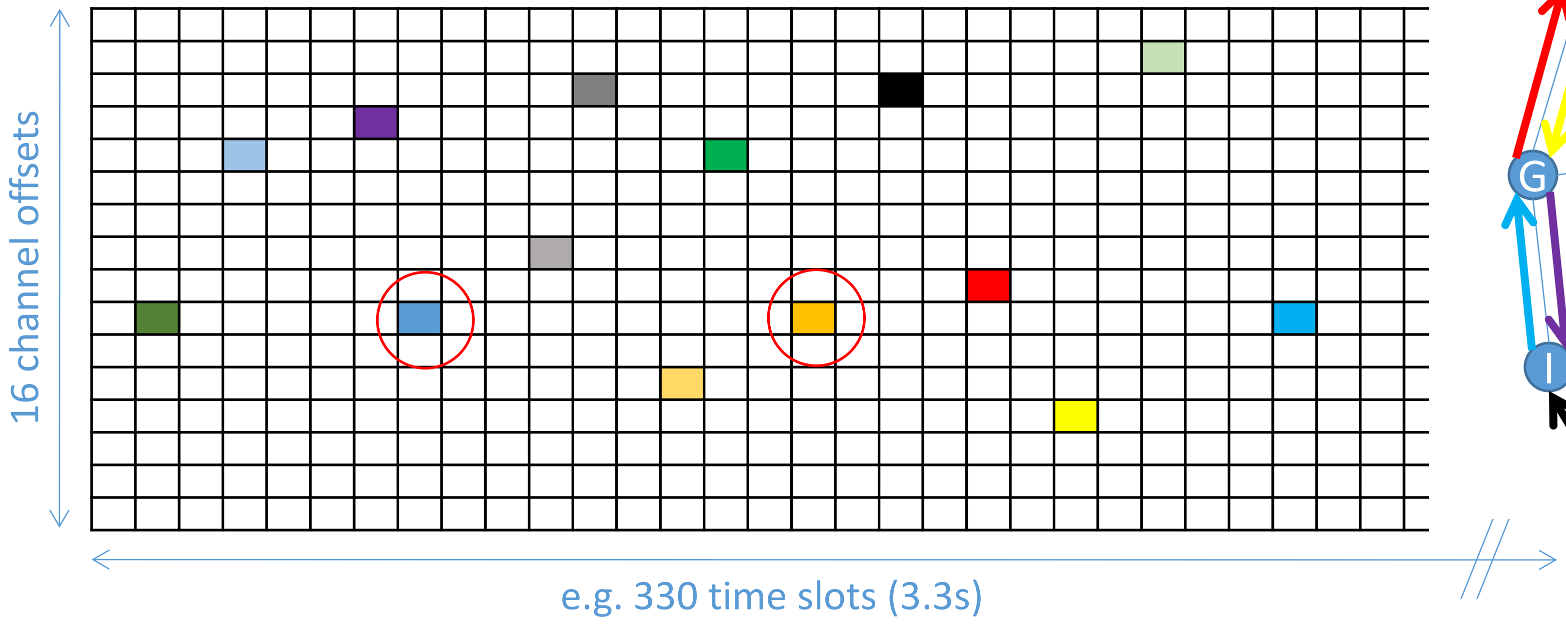
  and energy consumption

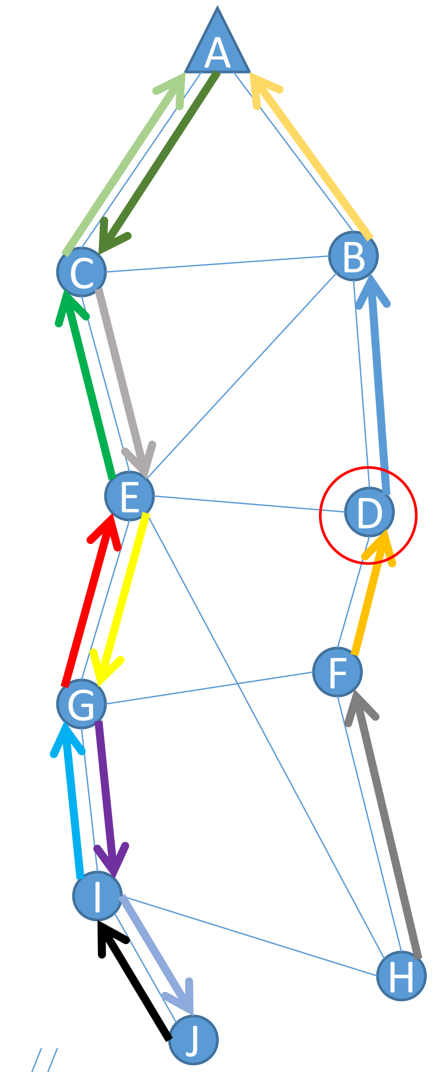




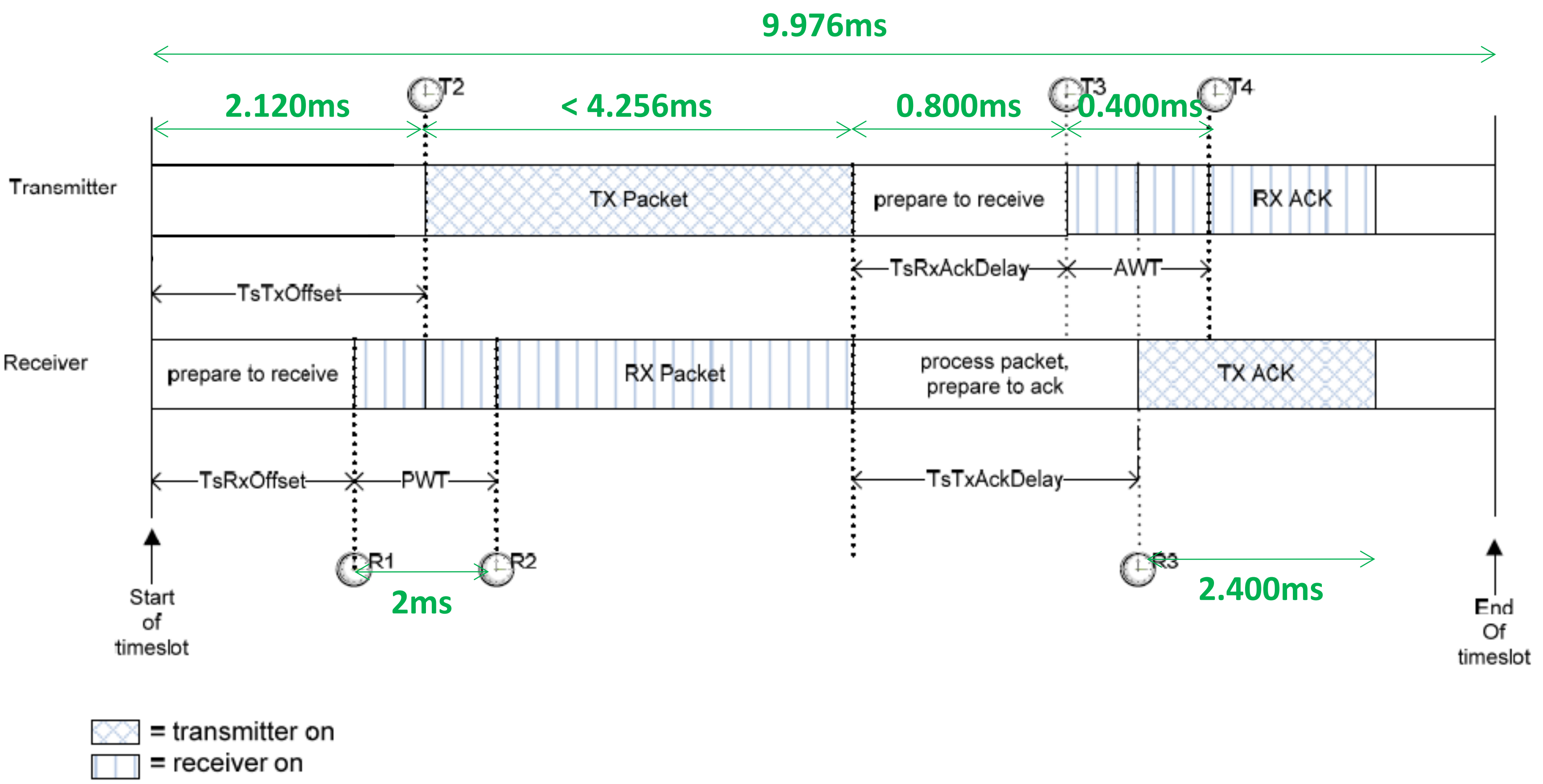
IEEE802.15.4e – Lifetime

- Looking at node D
  - “normal” case
    - Assume 1 AA battery & 14mA if radio on (AT86RF231)
    - 1 reception, 1 transmission (15ms) every 3.3 seconds
    - .45% duty cycle → 4 years lifetime

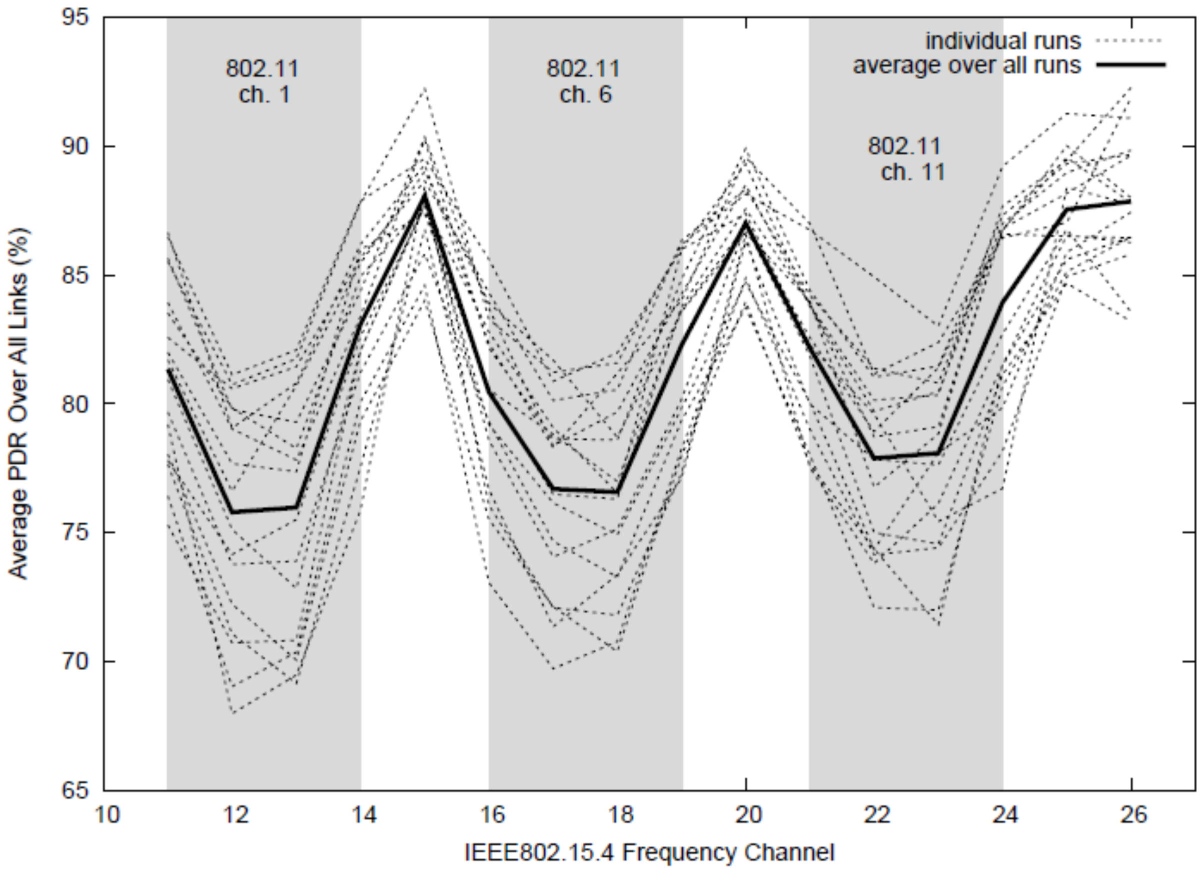




Inside each slot



Impacts of 2.4 GHz interference



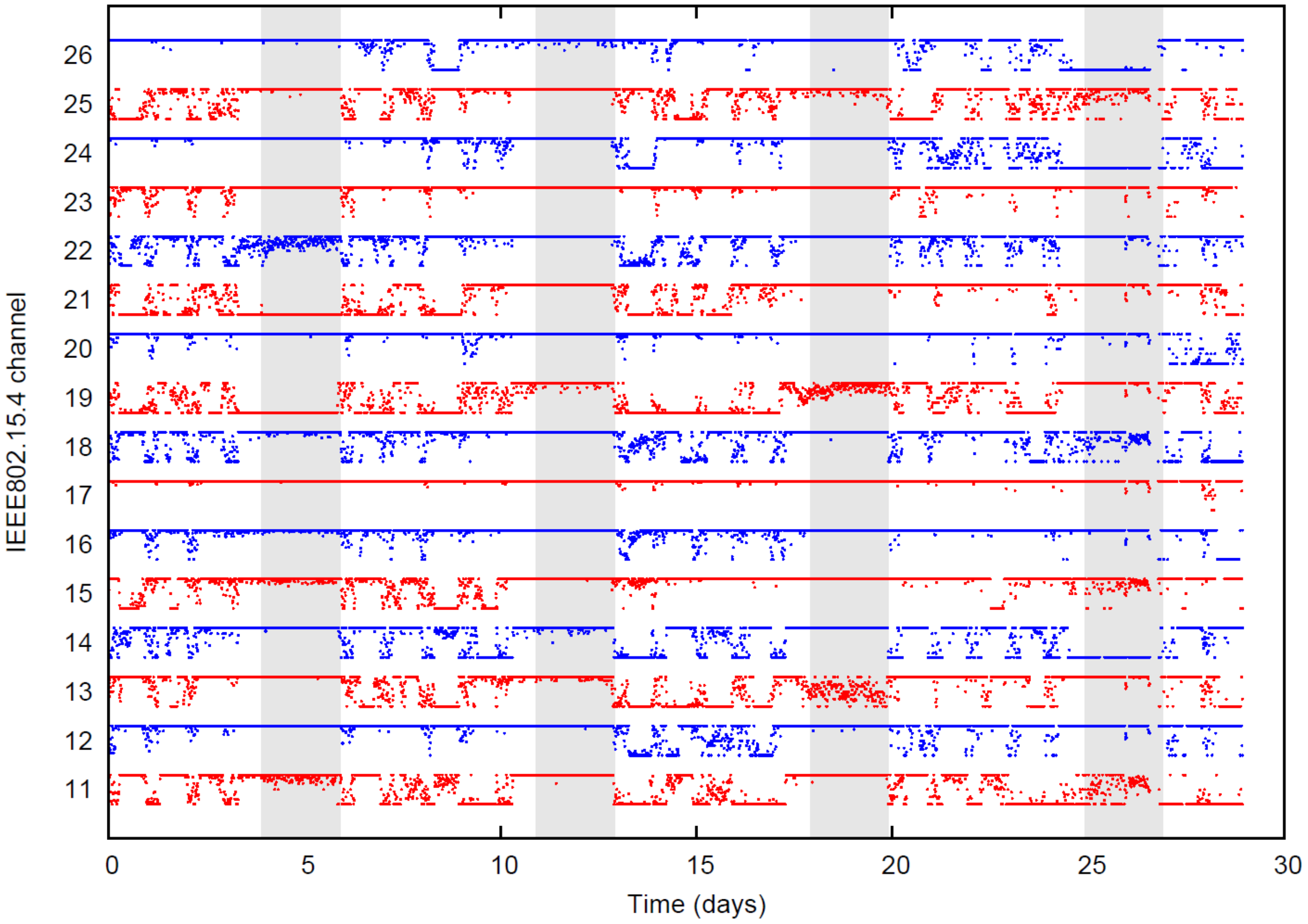
- 45 motes*
- 50x50m office environment
- 12 million packets exchanged, equally over all 16 channels
- Data from 2009!!

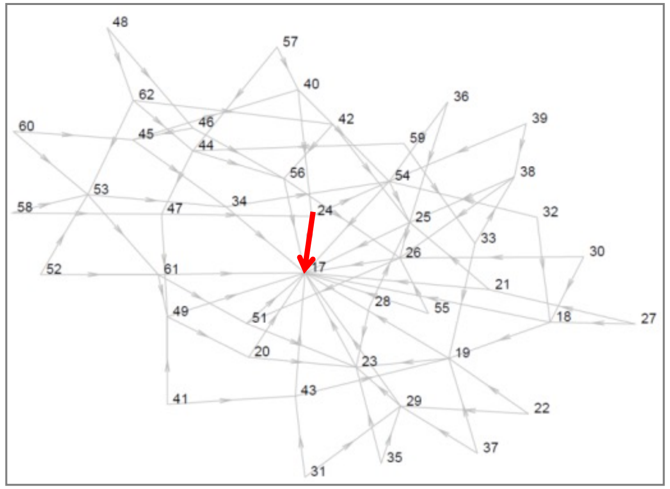
- Channels have wide packet delivery ratio (PDR) distribution

"Reliability Through Frequency Diversity: Why Channel Hopping Makes Sense," T. Watteyne, A. Mehta, K. Pister, *PE-WASUN, Oct. 2009.*

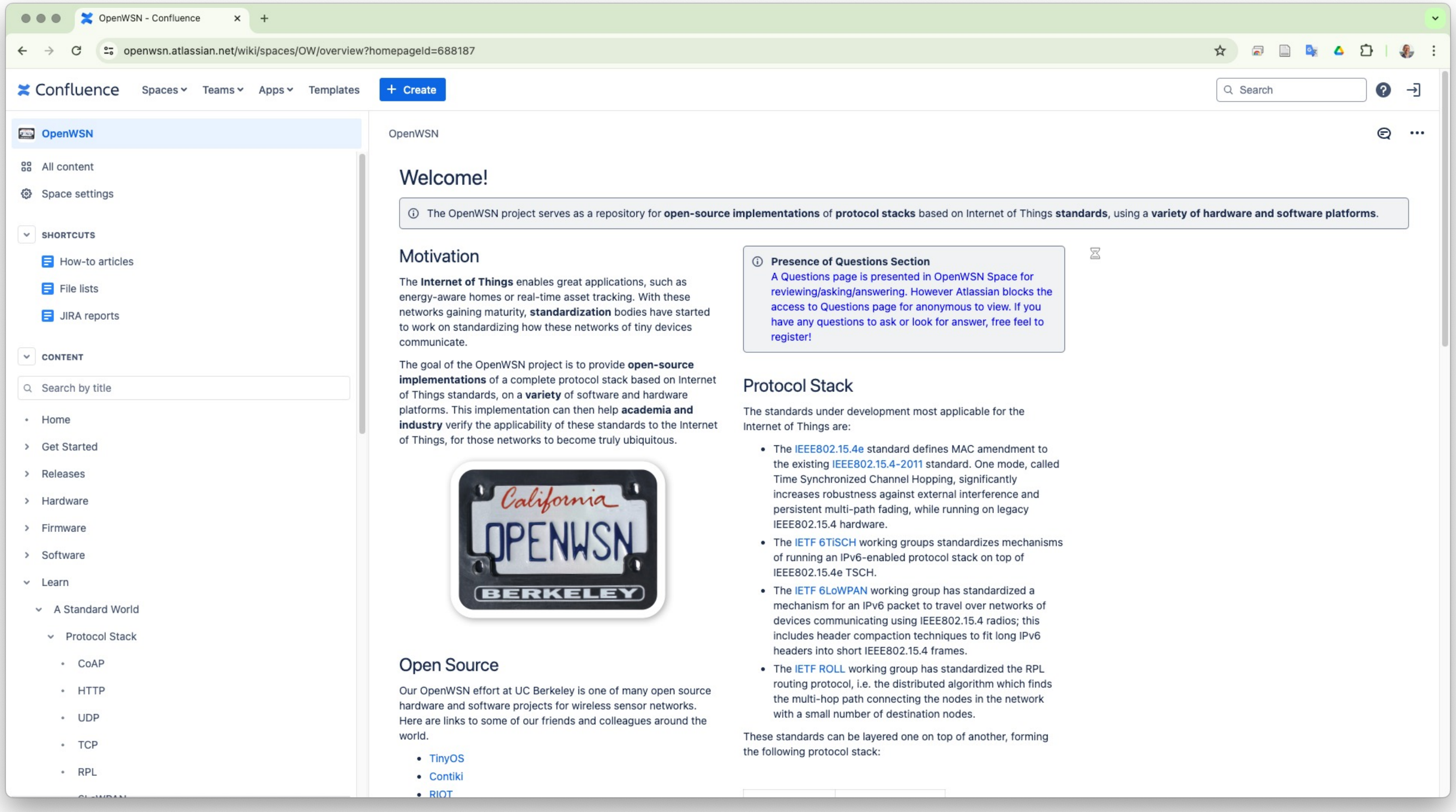
*data collected by Jorge Ortiz and David Culler, UCB

Inteference + multipath effects over time





Feasibility Analysis of Optimal Controller Design for Adaptive Channel Hopping, B. Kerkez, T. Watteyne, M. Magliocco, S. Glaser, K. Pister, *WSNPerf, October 2009.*



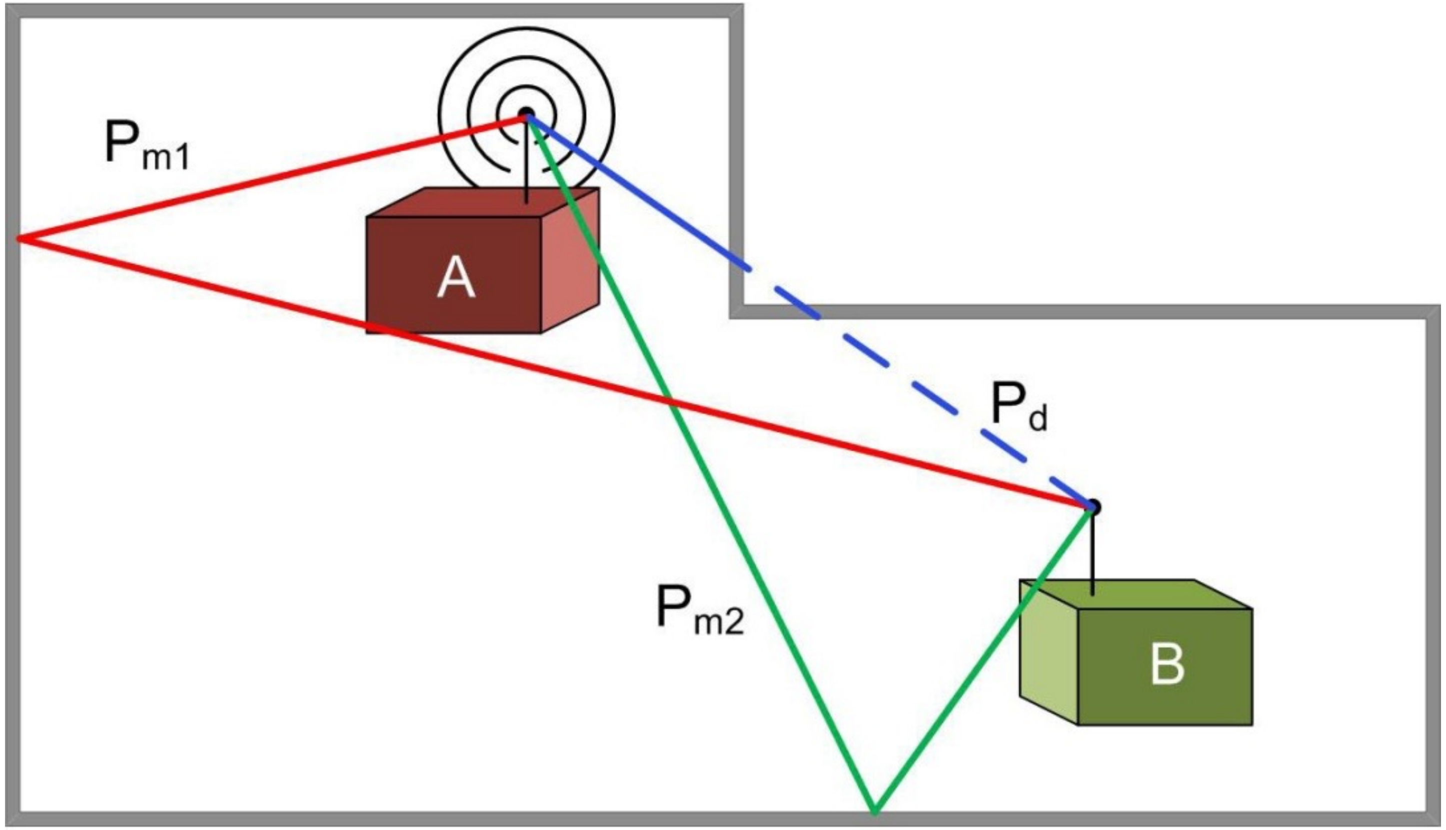
https://openwsn.org

https://github.com/openwsn-berkeley/

Personal recommendation to get started: nRF52840 dev kit $50

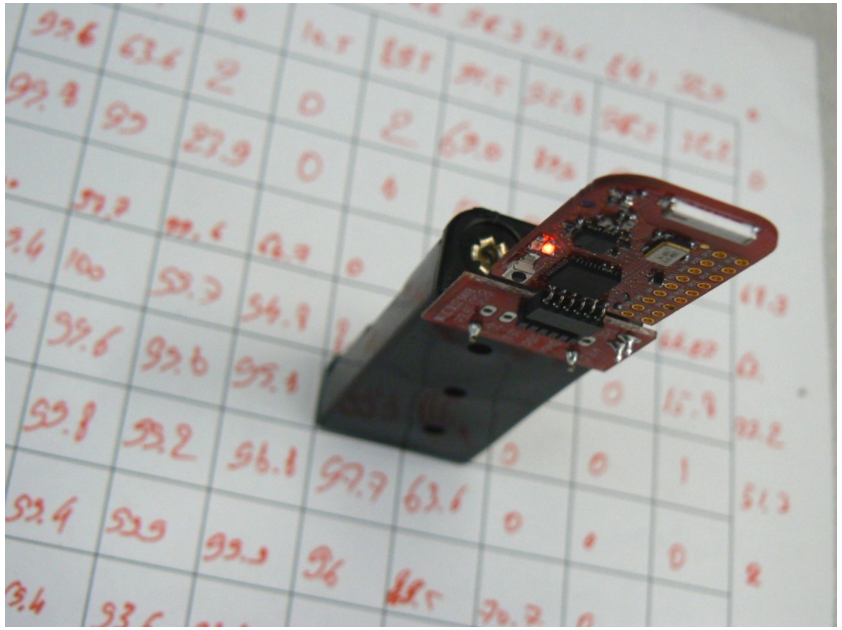
- backup slides

Second reliability challenge: multipath fading

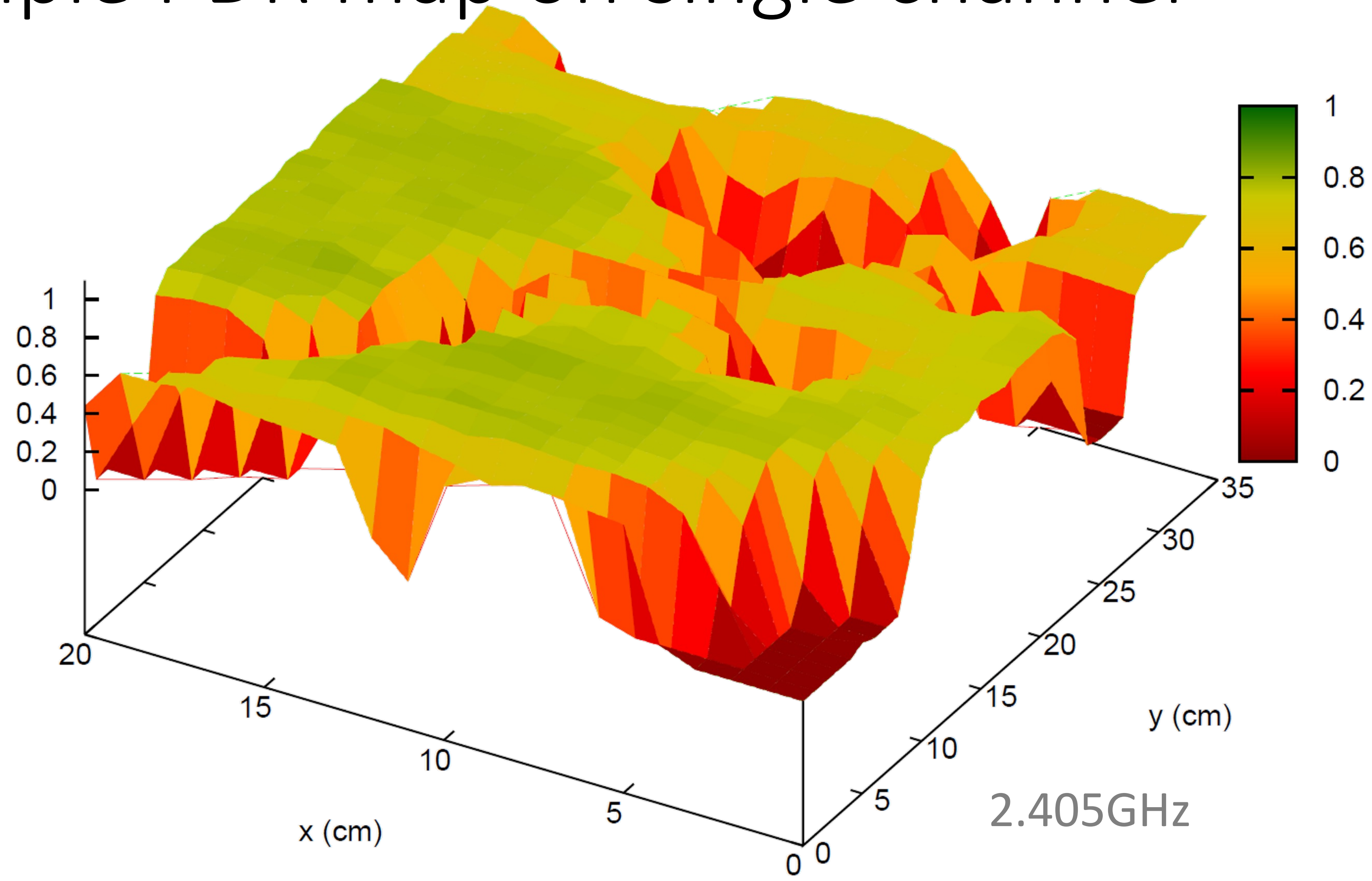


Quantifying impact of multipath fading

- Separate sender and receiver by 100cm
- Have sender send bursts of 1000 packets
- Have receiver count the number of received packets
- Move transmitter to next position in a 20cmx35cm area and send next burst of 1000 packets



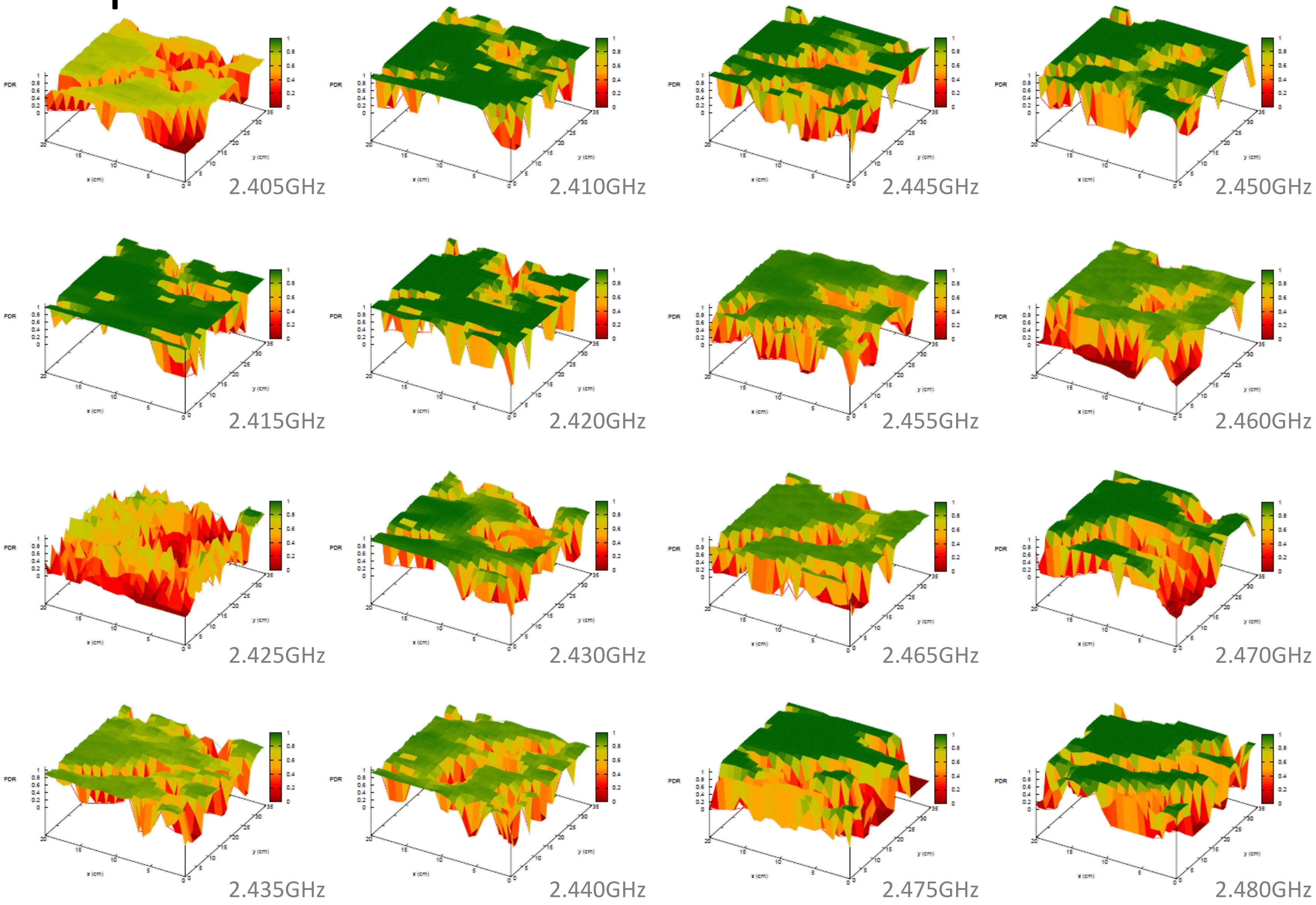
Example PDR map on single channel



2.405GHz

Mitigating Multipath Fading Through Channel Hopping in Wireless Sensor Networks, T. Watteyne, S. Lanzisera, A. Mehta, K. Pister, *IEEE International Conference on Communications (ICC), Cape Town, South Africa, 23-27 May, 2010.*

PDR maps across all 16 channels



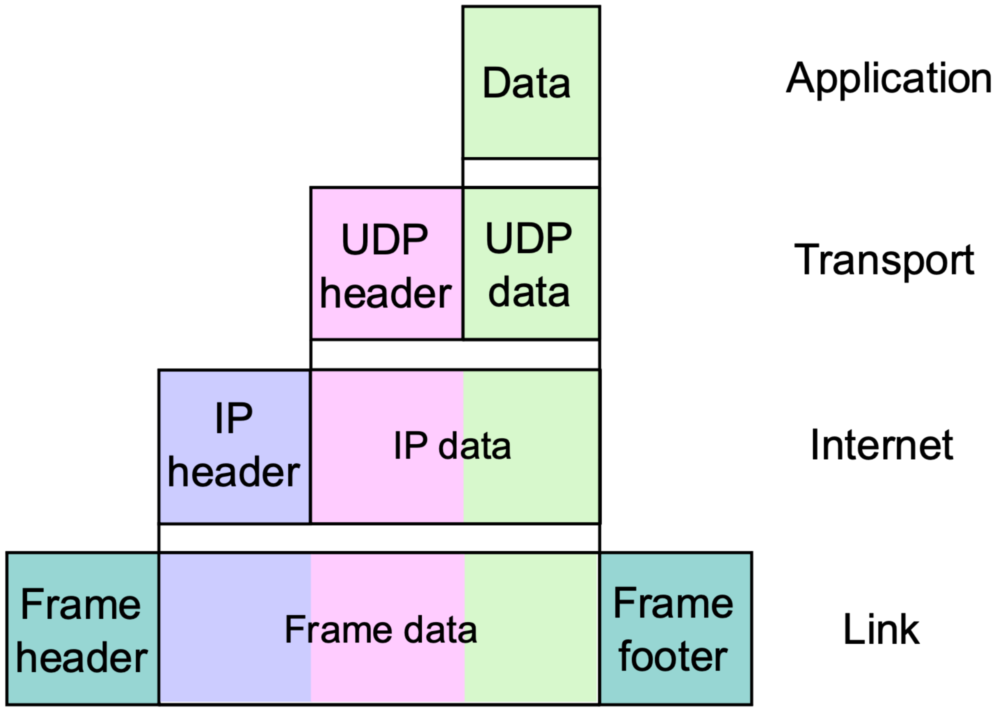
Dealing with unreliability

- At the medium access layer (i.e., between neighbors)
  - acknowledgements and retransmission upon failure
  - Time Division Multiple Access (TDMA) to avoid collisions
  - channel hopping to avoid successive transmission failures

- At the routing layer (i.e., over multiple hops)
  - dynamic routing topology to adapt to topology changes
  - Multiple paths possible for redundancy

- At the transport layer (i.e., for a given session)
  - acknowledgements and retransmission upon failure
  - application-aware resource reservation and allocation

Example message diagram



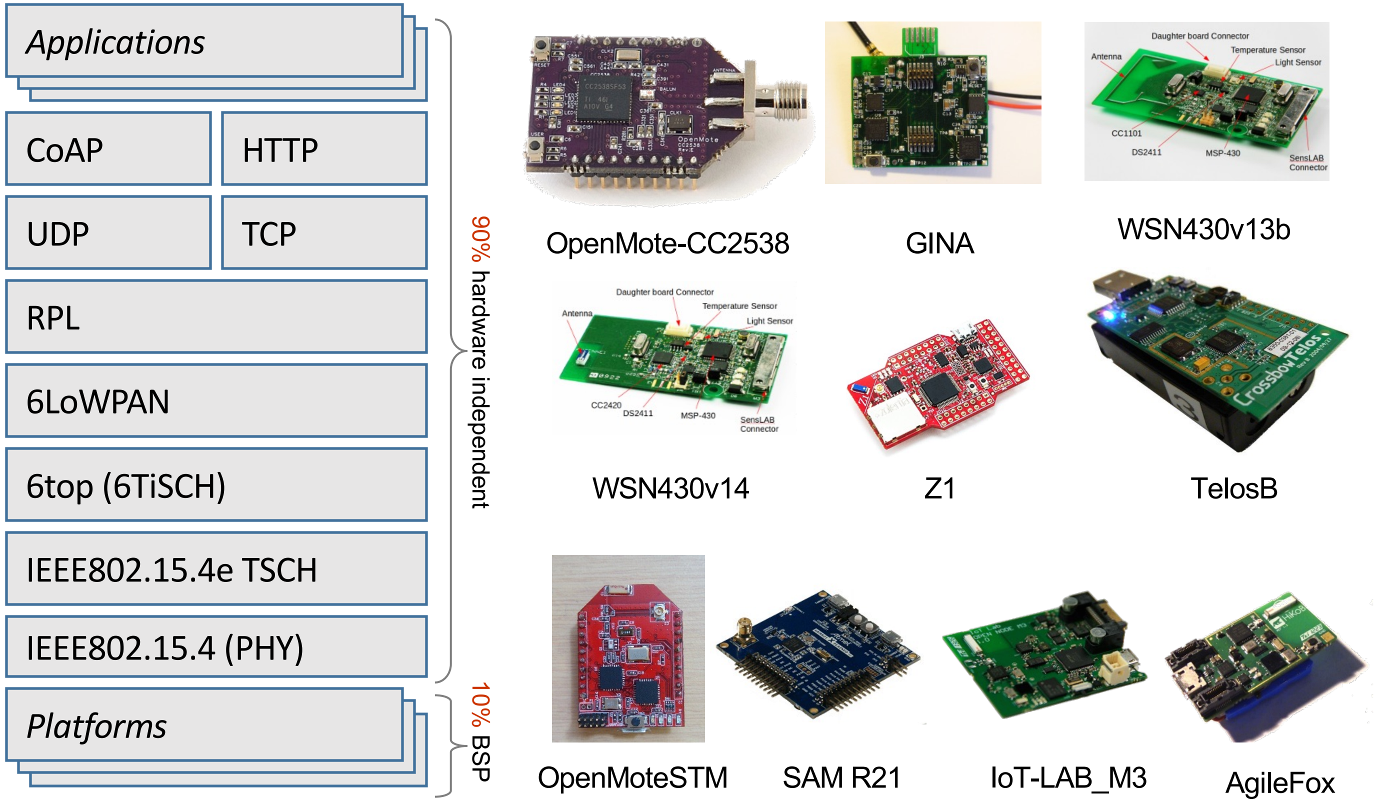
- Traffic analyzers, packet sniffers/loggers, etc. try to be "helpful" by removing headers, footers automatically
- Traffic analyzers may only be presented with frame data only or IP data only, etc., depending on where they are virtually or physically connected

https://en.wikipedia.org/wiki/Internet_protocol_suite#/media/File:UDP_encapsulation.svg

```
⊟ IEEE 802.15.4 Data, Dst: 14:15:92:0b:03:01:00:29, Src: 14:15:92:09:02:2b:00:51
  ⊞ Frame Control Field: Data (0xcc61)
    Sequence Number: 5
    Destination PAN: 0xbaad
    Destination: 14:15:92:0b:03:01:00:29 (14:15:92:0b:03:01:00:29)
    Source: 14:15:92:09:02:2b:00:51 (14:15:92:09:02:2b:00:51)
    FCS: 0x6c4a (Correct)
⊟ 6LoWPAN
  ⊞ IPHC Header
    Next header: UDP (0x11)
    Hop limit: 64
    Source: fe80::1615:9209:22b:51 (fe80::1615:9209:22b:51)
    Destination: fe80::1615:920b:301:29 (fe80::1615:920b:301:29)
⊟ Internet Protocol Version 6
  ⊞ 0110 .... = Version: 6
  ⊞ .... 0000 0000 .... .... .... .... .... = Traffic class: 0x00000000
    .... .... .... 0000 0000 0000 0000 0000 = Flowlabel: 0x00000000
    Payload length: 38
    Next header: UDP (0x11)
    Hop limit: 64
    Source: fe80::1615:9209:22b:51 (fe80::1615:9209:22b:51)
    Destination: fe80::1615:920b:301:29 (fe80::1615:920b:301:29)
⊟ User Datagram Protocol, Src Port: tivoconnect (2190), Dst Port: http-alt (8080)
    Source port: tivoconnect (2190)
    Destination port: http-alt (8080)
    Length: 38
  ⊞ Checksum: 0x06e3 [incorrect, should be 0x02e3 (maybe caused by "UDP checksum offload"?)]
⊟ Data (30 bytes)
    Data: 06ed07f7050d050f0835ffe2ffc6fe4584fc870091c0c500...
    [Length: 30]
```

```
                                                61 cc
05 ad ba 29 00 01 03 0b  92 15 14 51 00 2b 02 09
92 15 14 78 33 11 40 08  8e 1f 90 00 26 06 e3 06
ed 07 f7 05 0d 05 0f 08  35 ff e2 ff c6 fe 45 84
fc 87 00 91 c0 c5 00 ff  fc ff d7 00 02 4a 6c
```

Case study: OpenWSN stack



Followng OpenWSN slides from Berkeley Sensor & Actuator Center tutorial material hosted at:
https://openwsn.atlassian.net/wiki/spaces/OW/pages/688195/Tutorials