Design Considerations for Low Power Internet Protocols

Hudson Ayers

Paul Crews, Hubert Teo, Conor McAvity, Amit Levy, Philip Levis

Motivation

- Seamless interoperability foundational to the growth of IoT
- 6LoWPAN – Bringing the Internet Protocol to IoT
- Analysis of existing 6LoWPAN implementations reveals significant variation across stacks:
  - Inconsistent handling of ambiguities
  - Stacks don’t completely implement the specification

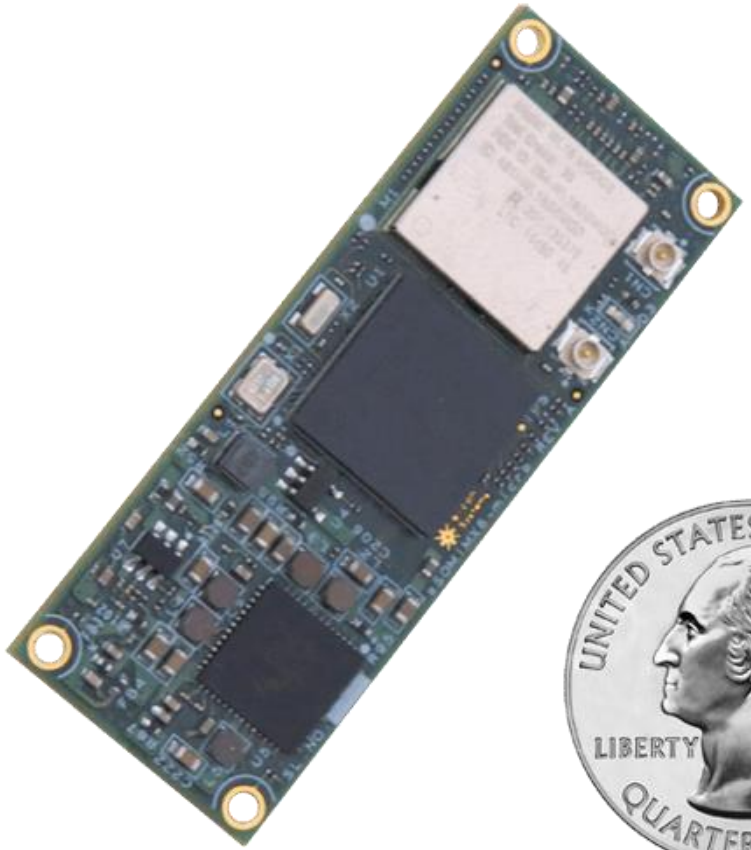
6LoWPAN Interoperability Study

- Conducted an Interoperability study between five Open Source 6LoWPAN implementations:

- **No pairing of these implementations completely interoperates!**

Why?

Primary Reason: **Constraints on Processor Resources (Code Size / RAM)**



How to Prevent This?

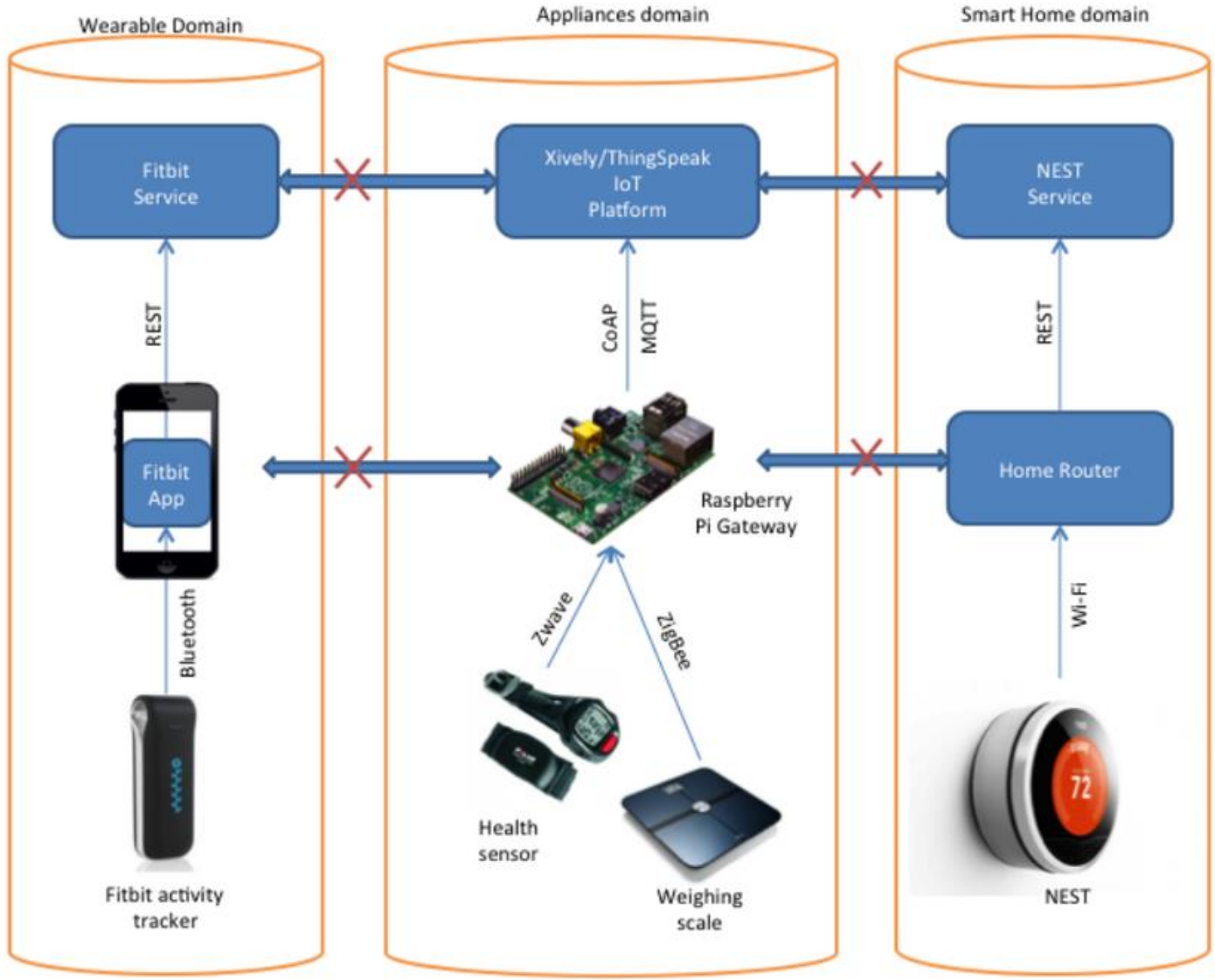
Our Take: Principled protocol design that anticipates problems that stem from the nature of the IoT space

Outline

1. Background – Internet Protocols for low power embedded devices
2. Motivation – Interoperability problems in 6LoWPAN
3. 4 Design Principles
4. Example Application of the principles to 6LoWPAN
5. Discussion/Takeaways

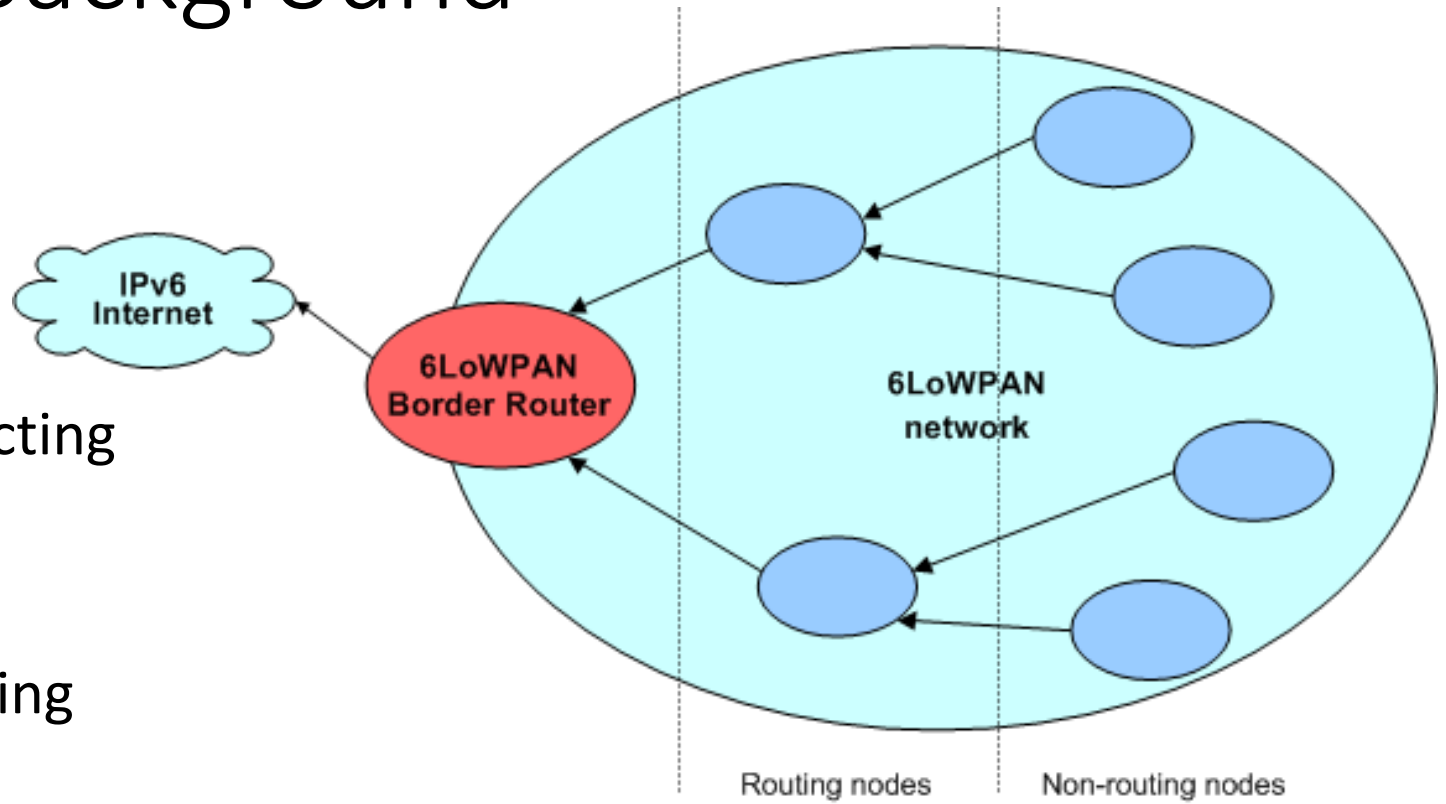
Background

- Historical IoT Networking Model: Vertical Silos
  - Inefficient
  - Limited Interoperability



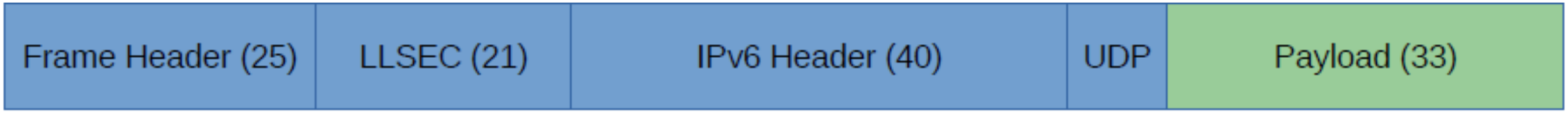
Background

- Preferable IoT Networking Model: IPv6
  - Large address space
  - Proven model for interconnecting networks
- Enter: 6LoWPAN
  - Set of IETF RFCs for transmitting IPv6 over IEEE 802.15.4 links



6LoWPAN

- IEEE 802.15.4 Links – 128 byte Maximum Transmission Unit
- IPv6 Packets – 1280 bytes
- 6LoWPAN Fragmentation (RFC 4944) allows for sending IPv6 over 15.4 Links
- 6LoWPAN compression (RFC 6282) mitigates overhead of large IPv6 headers to reduce radio energy consumption



Worst Case Scenario – Uncompressed IPv6 over 802.15.4

6LoWPAN Fragmentation – RFC 4944

- Specifies 4 LoWPAN Header Types
  - Mesh Addressing Header
  - Broadcast Header
  - Fragmentation Header
  - Compression Header
- Details how to fragment packets longer than 128 bytes

6LoWPAN Compression – RFC 6282

- Compression of Fields within the IPv6 Header
  - Traffic Class, Flow Label, Hop Limit
  - Stateless and Context-based compression of 128 bit IPv6 Addresses
  - Next Header Compression
  - Payload Length Elision
- Compression of Subsequent Headers
  - UDP Header Compression
  - IPv6 Options Compression
  - Tunneled IPv6 interior header compression
- And More…

6LoWPAN Interoperability Study

Feature	Stack				
	Contiki	**OpenThread**	**Riot**	**Arm Mbed**	**TinyOS**
Uncompressed IPv6	✓		✓	✓	✓
6LoWPAN Fragmentation	✓	✓	✓	✓	✓
1280 byte packets	✓	✓	✓	✓	✓
Dispatch_IPHC header prefix	✓	✓	✓	✓	✓
IPv6 Stateless Address Compression	✓	✓	✓	✓	✓
Stateless multicast address compression	✓	✓	✓	✓	✓
802.15.4 16 bit short address support		✓	✓	✓	✓
IPv6 Address Autoconfiguration	✓	✓	✓	✓	✓
IPv6 Stateful (Context Based) Address Compression	✓	✓	✓	✓	✓
Stateful multicast address compression		✓	✓	✓	
IPv6 Traffic Class and Flow label compression	✓	✓	✓	✓	✓
IPv6 NH Compression: Ipv6 (tunneled IPv6)		✓		✓	✓
IPv6 NH Compression: UDP	✓	✓	✓	✓	✓
UDP port compression	✓	✓	✓	✓	✓
UDP checksum elision					✓
Compression + headers past first first fragment			✓	✓	
Compression of IPv6 Extension Headers		~[2]		✓	✓
Mesh Header		✓		✓	~[3]
Broadcast Header					✓
Regular IPv6 ND	✓		✓	✓	~[4]
RFC 6775 6LoWPAN ND			✓	✓	
RFC 7400 Generic Header Compression Support					

~ = Partial Support

Why?

Possible Reasons:

- Incomplete development
- Difficulty keeping up with the changing protocol
- Lack of an established reference implementation
- **Constraints on Processor Resources (Code Size / RAM)**

Why?

- Evidence of concerns about code size and RAM can be found throughout each 6LoWPAN stack
  - Notes in documentation
  - Compile time options to reduce code size by omitting features
  - Comments in code about RAM savings
  - More…

Problem

- IoT platforms vary significantly
- IoT Applications vary significantly
  - Code Size
  - Memory
  - Power
  - Size
- Difficult to design one-size fits all protocols
- Embedded OSes must support broad range of boards

6LoWPAN Platforms supported by Contiki OS or Riot OS

Platform	Program Memory (kB)	RAM (kB)
Tmote Sky	48	10
Zolertia Z1	92	8
Atmel RZRaven	128	8
TI CC2650	128	28
SAMR21 Xpro	256	32
Nordic nRF52 DK	512	64
Arduino Due	512	96
Nest Protect*	750+	100

6LoWPAN Code Size Study

Stack	Code Size Measurements (kB)				
	Full IP stack	6LoWPAN-All	Compression	Fragmentation	Mesh/Broadcast Headers
Contiki	37.5	11.5	6.0	3.3	N/A
OpenThread	42.5	26.5	5-20	1.3	4.5
Riot	31.0	7.5	>4.8	1.5	N/A
Arm Mbed	46.0	22.1	17.9	3.1	1.3
TinyOS	37.3	16.2	– –	– –	0.6

* TinyOS inlines compression code into fragmentation code, and does not completely implement the mesh header

The results in the above table should generally be considered lower bounds

Platform	Program Memory (kB)	RAM (kB)
Tmote Sky	48	10
Zolertia Z1	92	8
Atmel RZRaven	128	8
TI CC2650	128	28
SAMR21 Xpro	256	32
Nordic nRF52 DK	512	64
Arduino Due	512	96
Nest Protect*	750+	100

Code Size vs. Compression

- Advanced MAC and physical layers, tracking network state, etc. can reduce radio energy consumption.
- These techniques require larger and more complex implementations.
- If too much of emphasis is put on saving energy through techniques that require substantial code space or RAM, it can force a requirement for more expensive, power hungry microcontrollers.

Other Considerations

- As a general rule, increased complexity is harmful to expectations of interoperability
- Postel's Law: "an implementation should be conservative in its sending behavior, and liberal in its receiving behavior"
  - Difficult to assume this law will be followed in the embedded space, where the cost of completely supporting reception of complicated protocols can be expensive
  - Instead, designers of low power protocols must prepare for the reality that some implementations may skimp wherever possible to conserve memory

4 Design Principles

Principle 1: Capability Spectrum

- A protocol should support a spectrum of device capabilities.
  - Defines a clear ordering via which devices can reduce code size or RAM use by eliding features
  - Makes a protocol usable by extremely low resource devices without forcing more resourceful devices to communicate inefficiently.

Application to 6LoWPAN

Replace the large collection of “MUST” requirements with 6 levels of functionality:

(0) Uncompressed IPv6
- Uncompressed IPv6
- 6LoWPAN Fragmentation and the Fragment Header
- 1280 Byte Packets

(1) IPv6 Compression Basics + Stateless Address Compression
- Support for the Dispatch_IPHC Header Prefix
- Correctly handle elision of IPv6 length and version
- Stateless compression of unicast addresses
- Stateless compression of multicast addresses
- Compression even when 16 bit addresses are used at the link layer
- IPv6 address autoconfiguration

(2) Stateful IPv6 Address Compression
- Stateful compression of unicast addresses
- Stateful compression of multicast addresses

(3) IPv6 Traffic Class and Flow Label Compression
- Traffic Class compression
- Flow Label Compression
- Hop Limit Compression

(4) IPv6 and UDP NH Compression + UDP Port Compression
- Handle Tunneled IPv6 correctly
- Handle the compression of the UDP Next Header
- Correctly handle elision of the UDP length field
- Correctly handle the compression of UDP ports
- Correctly handle messages for which headers go on longer than the first fragment, and the headers in the first fragment are compressed.

(5) Entire Specification
- Support the broadcast header and the mesh header as described in RFC 4944
- Support compression of all IPv6 Extension headers

Principle 2: Capability Negotiation

- There should be an explicit mechanism by which two devices can efficiently negotiate what level to use when they communicate
- If two devices wish to communicate, they default to the lower of their supported capability levels

Application to 6LoWPAN

Two mechanisms for capability negotiation:

- New ICMP6 message type: 6LoWPAN Class Unsupported
  - Do not require state to communicate failures
  - Typical means of communicating lack of support
- New 6LoWPAN ND Option
  - 6LoWPAN ND + small amount of state would allow for storing capability class alongside addresses
  - Minimizes energy cost of classes by preventing failures before they occur

Principle 3: Provide Reasonable Bounds

- Specifications should specify reasonable bounds on recursive or variable features.
  - Allows implementations to safely limit their RAM use without silent interoperability failures.

Application to 6LoWPAN

- Bound 6LoWPAN Header decompression to 50 bytes
  - Allows for simple implementations which conserve RAM by preventing need for initial fragment buffers to be much larger than 128 bytes
- Remove requirement for compression of Interior Headers for Tunneled IPv6
  - Many interop failures associated with this requirement
  - Limits complexity of next header compression and removes possibility of unbounded recursion

Principle 4: Don’t Break Layering

- Energy-saving optimizations should not make assumptions about the rest of the stack despite the appeal of cross-layer optimization in embedded systems
- Long-lived IoT systems will evolve and change, and systems use and draw on existing operating systems as well as libraries. Enforcing layering ensures developers need not own and customize the entire software stack.

Application to 6LoWPAN

- Remove UDP Checksum elision from RFC 6282
  - Rarely used
  - Complex to implement with application layer + link layer checks
  - Breaks end-to-end argument
  - Breaks layering

Discussion + Conclusion

- Worth thinking about why these problems with 6LoWPAN exist
  - Historically, the embedded research community largely separate from communities dealing with protocol creation + interoperability
- Principles not limited to 6LoWPAN
- Central Idea: Low Power Internet protocols must allow devices to use the protocol *and* optimize for their particular resource tradeoffs