Design Considerations for Low Power Internet Protocols

Draft-ayers-low-power-interop-00

Hudson Ayers

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

Motivation

- “The Working Group will generate the necessary documents to ensure interoperable implementations of 6LoWPAN networks” – 6LoWPAN WG Charter
- Analysis of existing 6LoWPAN implementations reveals significant variation across stacks

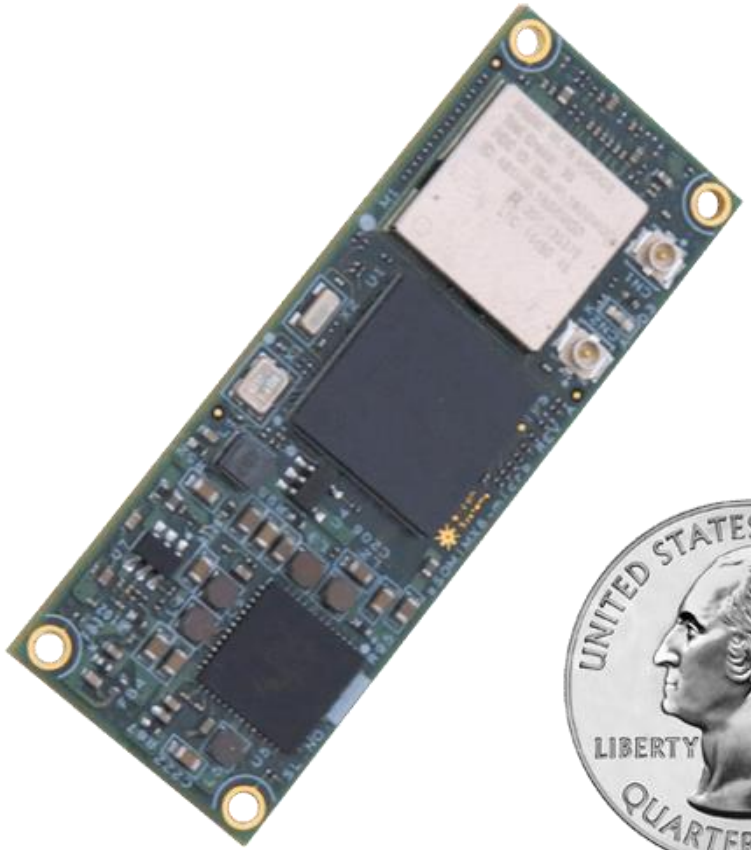
6LoWPAN Interoperability Study

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

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

Why?

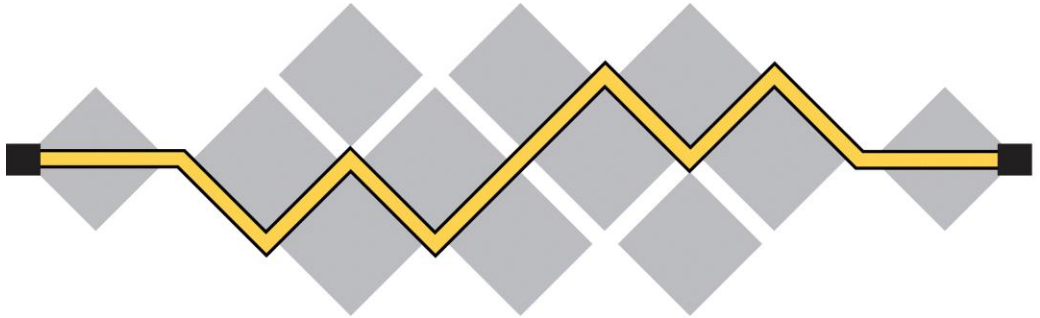
Primary Reason: **Constraints on Processor Resources (Code Size / RAM)**



How to Prevent This?

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

We Present: An informational internet-draft intended to assist in this



IETF®

Outline

1. Motivation – Interoperability problems in 6LoWPAN
2. 4 Design Guidelines
3. Example Application of the guidelines to 6LoWPAN
4. Discussion/Takeaways

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 fragment			✓	✓	
Compression of IPv6 Extension Headers		~		✓	✓
Mesh Header		✓		✓	~
Broadcast Header					✓
Regular IPv6 ND	✓		✓	✓	~
RFC 6775 6LoWPAN ND			✓	✓	
RFC 7400 Generic Header Compression Support					

~ = Partial Support

6LoWPAN Interoperability Study

Result:

- Example: Mbed OS → Contiki w/ IPv6 Extension Header

No.	Time	Source	Destination	Protocol	Length	Info
1	0.000000	fe80::212:4b00:1204…	fe80::282a:2a2a:2a2a:2a2a	UDP	171	7776 → 7776 Len=55
2	0.004450			IEEE 802.15.4	49	Ack
3	0.007478	fe80::212:4b00:1204…	fe80::282a:2a2a:2a2a:2a2a	UDP	171	7776 → 7776 Len=55
4	0.011927			IEEE 802.15.4	49	Ack

- Silent network layer drops!
- Disconcertingly easy to produce more examples of this

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

```
#ifdef MODULE_GNRC_SIXLOWPAN_IPHC_NHC
static inline size_t iphc_nhc_udp_decode(gnrc_pktsnip_t *pkt, gnrc_pktsnip_t **dec_hdr,
                                         size_t datagram_size, size_t offset)
{
    uint8_t *payload = pkt->data;
    gnrc_pktsnip_t *ipv6 = *dec_hdr;
    ipv6_hdr_t *ipv6_hdr = ipv6->data;
#ifdef MODULE_GNRC_UDP
    const gnrc_nettype_t snip_type = GNRC_NETTYPE_UDP;
#else
    const gnrc_nettype_t snip_type = GNRC_NETTYPE_UNDEF;
#endif
```

```
63  /* Save some ROM */
64  #undef UIP_CONF_TCP
65  #define UIP_CONF_TCP                    0
66
67  #undef SICSLOWPAN_CONF_FRAG
68  #define SICSLOWPAN_CONF_FRAG            0
```

```
251  /* Assuming that the worst growth for uncompression is 38 bytes */
252  #define SICSLOWPAN_FIRST_FRAGMENT_SIZE (SICSLOWPAN_FRAGMENT_SIZE + 38)
```

Problem

- IoT platforms, apps vary significantly
  - Code Size
  - Memory
  - Power
  - Size
- 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*

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

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

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 Guidelines

Author's Note: I am not strongly committed to the *exact* examples in the slides that follow – I am committed to the guidelines, but am looking for discussion regarding my example applications, and do not want the evaluation of the guidelines themselves to be tainted by any specific details of the example applications.

Guideline 1: Capability Advertisements

- Silent drops should not occur due to unimplemented portions of a specification
- There should be an explicit mechanism by which devices can efficiently learn the capabilities of other devices
- If two devices wish to communicate, they can default to the lower of their supported capability levels

Application to 6LoWPAN

Two mechanisms for capability advertisement:

- **New ICMP6 message type: 6LoWPAN Class Unsupported**
  - **Do not require state to communicate failures**
  - **Typical means of communicating lack of support**
  - **We believe that an addition of such a message should be seriously considered**
- 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

Guideline 2: Capability Spectrum

- Given that these capability advertisements exist, how can they efficiently share useful information?
- 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.
  - Somewhat similar to the idea presented in draft-hui-6lowpan-interop-00 in 2008 for testing interoperability of independent implementations

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

Guideline 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

Guideline 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 6LoWPAN interoperability is lacking
  - Historically, the embedded research community largely separate from communities dealing with protocol creation + interoperability
- Guidelines not limited to 6LoWPAN
- Central Idea: Low Power Internet protocols must allow devices to use the protocol *and* optimize for their particular resource tradeoffs