Design Considerations for Low Power Internet Protocols

Hudson Ayers, Paul Crews, Hubert Teo, Conor McAvity, Amit Levy [†], Philip Levis

 Stanford University, [†]Princeton University

July 29, 2020

Outline

Finding 1

6LoWPAN implementations do not interoperate

Outline

Finding 1

6LoWPAN implementations do not interoperate

Finding 2

This is in large part because of code size issues

Outline

Finding 1

6LoWPAN implementations do not interoperate

Finding 2

This is in large part because of code size issues

Finding 3

Traditional protocol design principles lead to bloated code

Outline

Finding 1

6LoWPAN implementations do not interoperate

Finding 2

This is in large part because of code size issues

Finding 3

Traditional protocol design principles lead to bloated code

Contribution

Low-power protocol design principles can solve this

Outline

Evaluation 1

Modified version of 6LoWPAN, based on these principles

Outline

Evaluation 1

Modified version of 6LoWPAN, based on these principles

Evaluation 2

Implementation of modified 6LoWPAN in an embedded OS

Outline

Evaluation 1

Modified version of 6LoWPAN, based on these principles

Evaluation 2

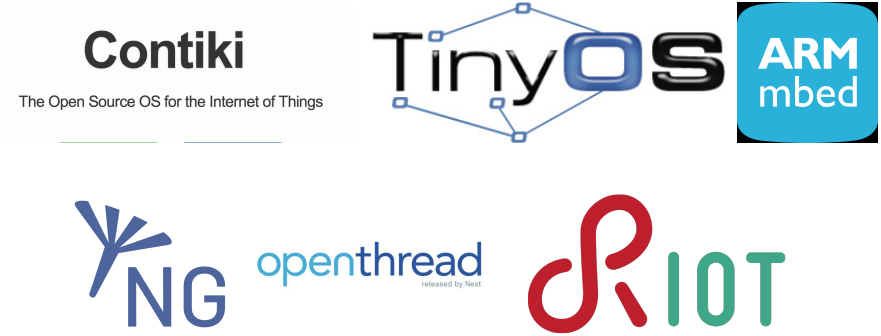
Implementation of modified 6LoWPAN in an embedded OS

Evaluation 3

Compare modified and unmodified versions of 6LoWPAN

6LoWPAN Interoperability

We conducted an interoperability test between 6 open source 6LoWPAN implementations



6LoWPAN Interoperability

Each implementation could communicate with the broader internet! But...

6LoWPAN Interoperability

Each implementation could communicate with the broader internet! But...

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

In many cases, implementations could not communicate **with each other**.

- Antithetical to the goals of IP!

6LoWPAN Interoperability

Each implementation could communicate with the broader internet! But...

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

In many cases, implementations could not communicate **with each other**.

- Antithetical to the goals of IP!

Failure Manifestation

Silent network layer drops; despite link layer acknowledgements

Complete Interoperability Matrix

Sender \ Receiver	Contiki	Contiki-NG	OpenThread	Riot	Arm Mbed	TinyOS
Contiki		■	■	■	■	■
Contiki-NG	■		■	■	■	■
OpenThread	■	■		■	■	■
Riot	■	■	■		■	■
Arm Mbed	■	■	■	■		■
TinyOS	■	■	■	■	■	

For each pairing, we found valid (per the standard) packets that were generated by one stack but were dropped by the other.

Digging in: Interoperability per-feature

Feature	Stack					
	Contiki	Contiki-NG	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 Address Compression	✓	✓	✓	✓	✓	✓
IPv6 Stateful multicast address compression			✓	✓	✓	
IPv6 Traffic Class and Flow label compression	✓	✓	✓	✓	✓	✓
IPv6 NH Compression: Ipv6 (tunneled)			✓		✓	✓
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						

~ = Partial Support

Digging in: Interoperability per-feature

Feature	Stack					
	Contiki	Contiki-NG	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 Address Compression	✓	✓	✓	✓	✓	✓
IPv6 Stateful multicast address compression			✓	✓	✓	
IPv6 Traffic Class and Flow label compression	✓	✓	✓	✓	✓	✓
IPv6 NH Compression: Ipv6 (tunneled)			✓		✓	✓
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						

~ = Partial Support

Digging in: Interoperability per-feature

Feature	Stack					
	Contiki	Contiki-NG	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 Address Compression	✓	✓	✓	✓	✓	✓
IPv6 Stateful multicast address compression			✓	✓	✓	
IPv6 Traffic Class and Flow label compression	✓	✓	✓	✓	✓	✓
IPv6 NH Compression: Ipv6 (tunneled)			✓		✓	✓
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						

~ = Partial Support

Digging in: Interoperability per-feature

Feature	Stack					
	Contiki	Contiki-NG	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 Address Compression	✓	✓	✓	✓	✓	✓
IPv6 Stateful multicast address compression			✓	✓	✓	
IPv6 Traffic Class and Flow label compression	✓	✓	✓	✓	✓	✓
IPv6 NH Compression: Ipv6 (tunneled)			✓		✓	✓
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						

~ = Partial Support

Digging in: Interoperability per-feature

Feature	Stack					
	Contiki	Contiki-NG	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 Address Compression	✓	✓	✓	✓	✓	✓
IPv6 Stateful multicast address compression			✓	✓	✓	
IPv6 Traffic Class and Flow label compression	✓	✓	✓	✓	✓	✓
IPv6 NH Compression: Ipv6 (tunneled)			✓		✓	✓
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						

~ = Partial Support

Digging in: Interoperability per-feature

Feature	Stack					
	Contiki	Contiki-NG	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 Address Compression	✓	✓	✓	✓	✓	✓
IPv6 Stateful multicast address compression			✓	✓	✓	
IPv6 Traffic Class and Flow label compression	✓	✓	✓	✓	✓	✓
IPv6 NH Compression: Ipv6 (tunneled)			✓		✓	✓
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						

~ = Partial Support

Digging in: Interoperability per-feature

Feature	Stack					
	Contiki	Contiki-NG	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 Address Compression	✓	✓	✓	✓	✓	✓
IPv6 Stateful multicast address compression			✓	✓	✓	
IPv6 Traffic Class and Flow label compression	✓	✓	✓	✓	✓	✓
IPv6 NH Compression: Ipv6 (tunneled)			✓		✓	✓
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						

~ = Partial Support

Why?

```
#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)
```

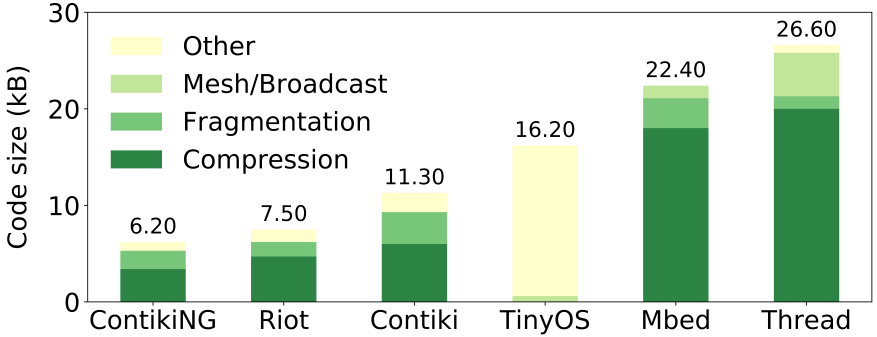
```
85  To minimize memory usage, i.e. disable everything (at the moment only
86  the UDP cli) to determine minimum RAM/ROM requirements, use
87          CFLAGS="-D'MINIMIZE_MEMORY=1'
```

Flash size varies widely across IoT Platforms

IoT Platform	Code (kB)	Year
EMB-WMB	64	2012
Zolertia Z1	92	2013
TI CC2650	128	2015
NXP MKW40Z	160	2015
SAMR21 XPro	256	2014
Nordic NRF52840 DK	512	2018

- OSes must support variety of applications and boards with different constraints
- This includes severely restricted low-cost or low-energy boards
- In research code size is rarely an issue; in real-world deployments it is

6LoWPAN Code Size is significant



TinyOS's whole-program optimization model precluded separating out components

These numbers represent strict lower bounds, as they were conservatively calculated

Code Size, Energy, and Traditional Principles

- Advanced MAC/PHY layers, compression, tracking network state, etc. saves energy, and are very important for some deployments
- These techniques require larger and more complex code
- If protocols focus too much on saving energy, it can ironically force applications using that protocol to require more expensive, power hungry microcontrollers.
- But traditional protocol principles like Postel's Law – "Be liberal in what you accept, and conservative in what you send" – require receivers support all non-optional complexity in a protocol

Low Power Protocol Design Principles

Low Power Protocol Design Principles

Principle 1: Capability Spectrum

Low power protocols should support both ultra-low energy devices as well as devices with very limited code space. Rather than all devices paying the code size costs of complex energy optimizations, protocols should support a linear spectrum of device capabilities.

Low Power Protocol Design Principles

Principle 1: Capability Spectrum

Low power protocols should support both ultra-low energy devices as well as devices with very limited code space. Rather than all devices paying the code size costs of complex energy optimizations, protocols should support a linear spectrum of device capabilities.

Principle 2: Capability Discovery

There should be a way for devices to determine what capability level to communicate with.

Low Power Protocol Design Principles

Principle 1: Capability Spectrum

Low power protocols should support both ultra-low energy devices as well as devices with very limited code space. Rather than all devices paying the code size costs of complex energy optimizations, protocols should support a linear spectrum of device capabilities.

Principle 2: Capability Discovery

There should be a way for devices to determine what capability level to communicate with.

Principle 3: Explicit and Finite Bounds

Protocols must specify explicit and reasonable bounds on recursive or variable features so implementations can bound RAM use.

Low Power Protocol Design Principles

Principle 1: Capability Spectrum

Low power protocols should support both ultra-low energy devices as well as devices with very limited code space. Rather than all devices paying the code size costs of complex energy optimizations, protocols should support a linear spectrum of device capabilities.

Principle 2: Capability Discovery

There should be a way for devices to determine what capability level to communicate with.

Principle 3: Explicit and Finite Bounds

Protocols must specify explicit and reasonable bounds on recursive or variable features so implementations can bound RAM use.

These may seem obvious, but low power protocols today do not do these things!

Evaluating the Principles

- Can capability levels provide a good range of implementation complexity?
- What is the overhead of capability discovery?
- Is a linear capability spectrum really better than arbitrary feature selection?

Approach: Apply the principles to 6LoWPAN

Applying to 6LoWPAN

Principle 1: Capability Spectrum

- Break down individual features of 6LoWPAN into 6 capability levels
- Lower levels prioritize features with high energy savings relative to added complexity

Capability	Basic Description / Added Features
Level 0	Uncompressed IPv6 + ability to send ICMP errors • Uncompressed IPv6 • 6LoWPAN Fragmentation (Fragment Header) • 1280 Byte Packets • Stateless decompression of source addresses
Level 1	IPv6 Compression Basics + Stateless Addr Compression • Support for the Dispatch_IPHC Header Prefix • Correctly handle elision of IPv6 length and version • Stateless compression of all unicast addresses • Stateless compression of multicast addresses • Compression + 16 bit link-layer addresses • IPv6 address autoconfiguration
Level 2	Stateful IPv6 Address Compression • Stateful compression of unicast addresses • Stateful compression of multicast addresses

Applying to 6LoWPAN

Principle 1: Capability Spectrum

- Break down individual features of 6LoWPAN into 6 capability levels
- Lower levels prioritize features with high energy savings relative to added complexity

Capability	Basic Description / Added Features
Level 3	IPv6 Traffic Class and Flow Label Compression • Traffic Class compression • Flow Label Compression • Hop Limit Compression
Level 4	Next Header 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 • Handle headers past the first fragment, when first fragment compressed.
Level 5 (all routers)	Entire Specification • Support the broadcast header and the mesh header • Support compression of all IPv6 Extension headers

Applying to 6LoWPAN

Principle 2: Capability Discovery

- New ND Option in Router Soliciations and Neighbour Advertisements
  - Nodes can store capability levels alongside link layer addresses
- New ICMPv6 message type: 6LoWPAN Class Unsupported
  - Robust error reporting is vital to interoperability
  - ICMP reporting prevents silent drops

Applying to 6LoWPAN

Principle 3: Provide reasonable bounds

- No recursive decompression of tunneled IPv6
- Hard limit on maximum growth from header decompression
- Details in draft!

Implementation

Principled 6LoWPAN (P6LoWPAN): Implementing these changes to 6LoWPAN

- Based on Contiki-NG's 6LoWPAN stack
- Add compile time flags to build stack at each capability level
- Add code for ICMP errors when packet decoding fails
- Add code for sending and storing capability levels
- Required changing about 500 lines of code

Can capability levels provide a good range of implementation complexity?

Table: 6LoWPAN code size of different capabilities levels in Contiki-NG.

Capability	Code Size (kB)	Increase (kB)
Level 0	3.2	-
Level 1	4.2	1.0
Level 2	4.8	0.6
Level 3	5.1	0.3
Level 4	5.6	0.5
Level 5	6.2	0.6

Takeaway

The spectrum spans a nearly 100% increase in code size.

What is the overhead of capability discovery?

Table: The cost of implementing capability discovery in Contiki-NG

Capability	6LoWPAN Code Size (kB)		
	Base	w/Discovery	Increase
Level 0	3.2	3.4	188 bytes
Level 1	4.2	4.4	260 bytes
Level 2	4.8	5.2	388 bytes
Level 3	5.1	5.4	340 bytes
Level 4	5.6	5.9	296 bytes
Level 5	6.2	6.3	172 bytes

Takeaway

Capability discovery costs less than 5% of the total 6LoWPAN size; the maximum size reduction from choosing a lower capability level is 10x the discovery cost.

Is a linear capability spectrum really better than arbitrary feature selection?

FLEX-6LoWPAN: We modified P6LoWPAN to allow implementations to select arbitrary features

Table: Resource requirements for modified 6LoWPAN in Contiki-NG, at equivalent of capability level 4

–	**Linear Spectrum**	**Arbitrary Bitfield**
6LoWPAN Code Size	5.9 kB	6.5 kB
RAM per neighbor	19 Bytes	22 Bytes
Typical ICMP message	12 Bytes	48 Bytes

Takeaway

Arbitrary feature selection has significant runtime costs, because low-capability nodes cannot compress responses.

Conclusions

1. Low power interoperability is hard! Device deployments are forced to make tradeoffs specific to their application requirements
2. Low power protocols must support different points in the spectrum of tradeoffs between code size and energy use
3. We present three principles that can reframe the discussion around how low power protocols are designed
4. Applying these principles to 6LoWPAN, we find that they enable a good range of implementation complexity and introduce acceptably low overhead.

Thank you!

Appendix

Run Time Overhead - Neighbour Discovery

Best approximation of overhead is link-layer payload bits

$$C = \text{Router Solicitation } \{RS\} + \text{Min. IP Hdr } \{2\} + \text{Router Advertisement } \{104\} + \text{Min. IP Hdr } \{2\} + (\text{Neighbor Solicitation } \{24\} + \text{Neighbor Advertisement } \{NA\} + 2 * \text{Min. IP Hdr } \{2\}) * N \quad \text{Registration Options in first NS } \{24\} + \text{Registration Options in first NA } \{16\} \tag{1}$$

where:

C = Minimum link-layer payload sent/received for ND

N = # of endpoints requiring address resolution

	RS	NA	C (Total ND Cost)
6LoWPAN	20	24	$168 + 52 * N$
P6LoWPAN	24	28	$172 + 56 * N$
FLEX-6LoWPAN	28	32	$176 + 60 * N$

Run Time Overhead - ICMP

- Overhead is one ICMP packet per failure between any two nodes
- link layer frame bits: Compressed Header Size + 4 bytes for Capability Option
- With linear spectrum, failures can only happen in one direction
- With arbitrary bitfield, failures can happen in both directions, so can't compress IP header in error message!
- Linear spectrum typical error payload: 12 bytes
- Arbitrary bitfield typical error payload: 48 bytes