Lightweight IPv6 Stacks for Smart Objects: the Experience of Three Independent and Interoperable Implementations

Internet Protocol for Smart Objects (IPSO) Alliance

White paper # 2

Julien Abeillé, Software Engineer, Cisco Systems
Mathilde Durvy, PhD, Software Engineer, Cisco Systems
Jonathan Hui, PhD, Arch Rock Corporation
Stephen Dawson-Haggerty, University of California, Berkeley

November 2008

Introduction

Historically, smart objects have used a plethora of communication technologies, both at the physical and medium access control layers, and at upper layers. The upper layers of the communication stack remain either proprietary or specified by exclusive alliances. This plethora of solutions renders interoperability between different sensor networks difficult. It also makes the seamless integration of sensor networks with existing IP networks impossible.

IP is an ideal solution to this end-to-end interoperability issue[1]. However, the adoption of IP as the Layer-3 protocol to connect wireless or wired smart objects has been impaired by the common belief that IP is not well-suited for the memory and energy constraints of such devices. In this white paper we give insights on key implementation aspects, based on the experience of three interoperable IPv6 stacks.

Interoperability is Essential

Interoperability is essential and the Internet Protocol (IP) is by far the most common language of computer networks today. IP allows smart objects to communicate end-to-end with other smart objects as well as existing IP-based devices. But IP devices must adhere to a number of specifications that define the Internet architecture in order for them to successfully join and participate in an IP network. IP protocols are defined by the Internet Engineering Task Force (IETF)[2] in the form of open standard protocols called RFCs.

RFC4294 summarizes requirements for IPv6 hosts and routers. In particular, it states that an IPv6 node MUST comply with the following standards: IPv6 Specification (RFC2460), IPv6 Addressing Architecture (RFC4291), IPv6 Neighbor Discovery (RFC4861), Internet Control Message Protocol for IPv6 (RFC4443), Stateless Address Autoconfiguration (RFC4862), Default Address Selection (RFC3484), and Multicast Listener Discovery (RFC2710, RFC3810).

The IPv6 Ready Logo program[3] was developed to ensure compliance with the standards required by RFC4294. IPv6 Ready delivers Phase-1, Phase-2 and Phase-3 logos that correspond to three levels of certification. Phase-1 requirements include most of the major building blocks of RFC4294. Phase-2 add support for ICMPv6 redirect messages and Path MTU discovery (RFC1981), the latter of which is not required by RFC4294. Phase-3 adds support for Internet Protocol Security (IPsec).

Leveraging Decades of Experience

Efforts in developing effective and scalable IP-based networks began 30 years ago. Since then, the Internet community has developed a number of protocols and gained a

---

[1] See the first white paper of the IPSO Alliance "IP for Smart Objects"
[2] http://www.ietf.org
[3] http://www.ipv6ready.org

significant amount of deployment experience. Open-source projects starting from the Berkeley Standard Distribution (BSD) helped to accelerate those efforts and drive implementations that are both scalable and efficient. BSD is responsible for providing the initial deployment efforts and served as the basis for improving TCP/IP over time.

The KAME[4] project is to IPv6 what the original BSD was to IPv4. The result of KAME is a high-quality open-source implementation that is used today in all BSD distributions (e.g. FreeBSD, NetBSD, OpenBSD, as well as Mac OS X). By leveraging the efforts from KAME we now have a much better understanding of how to develop fully-compliant IPv6 stacks that are efficient while providing high performance.

Today, there are a number of similar efforts that will help to drive the Internet and IP-based protocols forward. Examples include IPSec (KAME, USAGI), SNMP (openSNMP), Mobile IP (Helsinki University of Technology). In addition, the WIDE[5] project offers implementations of tens of IPv6 related protocols in many areas such as routing, management, real time applications, security, transport, etc.

Implementing Efficient IPv6 Stacks

Three independent implementations of a lightweight IPv6 stack for smart objects have already been developed. This section provides important design considerations in implementing an IPv6 network stack designed for the strict resource constraints typical to smart objects.

Required Data Structures

The data structures necessary to implement a minimal IPv6 stack are an interface address list, a neighbor cache, a prefix list, a routing table, and a default router list as defined in RFC4861. These structures are updated whenever packets are sent or received. Moreover, timers are used to manage entries that have a lifetime associated with them. Note that we assume here that the platform has a single interface even though we can expect to see constrained nodes with multiple interfaces in a near future (multiple RF, RF+PLC).

Buffering

The IPv6 specification requires the link to support a 1280-byte MTU, imposing significant challenges in supporting IPv6 on memory-constrained devices. A minimal implementation can utilize a single 1280-byte buffer for the basic data plane. However, the current Neighbor Discovery (ND) specification (RFC4861) requires a packet buffer for each neighbor on the link and is prohibitively expensive for smart objects. The IETF is currently discussing the relaxation of this buffering requirement as it is also viewed as a security issue (i.e. DoS attacks).

The single packet buffer works well for host devices that do not forward datagrams. Wireless mesh networks (e.g. IEEE 802.15.4) often span multiple hops and require some

[4] http://www.kame.net/
[5] http://www.wide.ad.jp/

or all devices to forward datagrams (and thus act as routers) to provide reachability throughout the network. As a result, the store-and-forward technique requires nodes to buffer additional packets that are being forwarded. One way to implement the forwarding queue is to allocate full packet buffers for each entry in the queue. More sophisticated approaches support variable-sized buffers that allow more effective use of the memory especially when datagrams do not consume the full 1280-byte MTU.

Layer 2 Dependencies

The layered IP protocol architecture allows the network layer to remain agnostic to the specifics of the link-layer below. IPv6 stacks implemented for smart objects do not have to give up the layered protocol architecture in order to fit within the severe resource constraints typical to smart objects.

In wired networks, the network layer only expects the link layer to provide services necessary for delivering packets. Networks operating on constrained (e.g. low throughput) links, however, can benefit from visibility between layers. On wireless links, for example, knowledge of when packets are acknowledged and some indication of signal strength can provide invaluable information to link estimators that are used by IP-based routing protocols. Similarly, routing protocols could indicate which neighbors are most important to optimize link-layer transmissions by providing visibility into the routing table. Furthermore, the link-layer could deliver packets more efficiently if it had visibility into the forwarding queue. For example, the link could optimize transmission of multiple frames to the same destination and the link could appropriately schedule transmissions to minimize average queuing delay. By providing this extra visibility in a link-independent manner, we can significantly enhance the performance of an IP-based communication architecture for smart objects while maintaining the layering.

Optimizations for Constrained Links

Link technologies used in Smart Objects are often highly constrained in terms of throughput, channel capacity, and MTU. Energy constraints can also limit how much information can be communicated. We describe a number of optimizations to enable more efficient operation over constrained links. While the optimizations described below were originally designed for low-power wireless networks, they equally apply to low-speed wired links as well.

Compressing IPv6 Headers

IPv6 has a base header of 40 bytes, which is fairly large especially when compared to the small link-layer MTUs and application payloads typical to smart objects. As a result, compressing IPv6 headers can help save memory in buffering those datagrams as well as reduce energy costs for delivering datagrams. With RFC 4944, the IETF has defined the 6LoWPAN adaptation layer that includes a compression mechanism. This mechanism is stateless which means that it creates no binding state between the compressor-decompressor pair. Stateless compression gives nodes the necessary flexibility to communicate with any neighbor in compact form at all times. While the current

specification only supports link-local communication, the IETF is currently working on mechanisms to support stateful compression for global communication. The stateful compression mechanisms rely on shared context throughout the entire network, allowing state synchronization to remain simple and nodes to communicate with any of their neighbors in compressed form.[6]

Fragmenting Packets on Link Technologies with Small MTUs

Many link technologies designed for smart objects do not support the full 1280-byte MTU. For several reasons, including cost reduction, guaranteeing an acceptable Packet Error Rates (PER), and microcontroller buffering constraints, IEEE 802.15.4 only supports a 127-byte MTU. To support the IPv6 1280-byte MTU, datagrams must thus be fragmented before they can be delivered to the link layer. With RFC4944, the IETF has defined the 6LoWPAN adaptation layer to support fragmentation. Fragmented datagrams also provide additional opportunity for more effective buffering techniques when forwarding datagrams. For example, fragments can be delivered before a node reassembles the entire datagram, allowing nodes with severe memory constraints to forward complete datagrams.

Extending Neighbor Discovery

IPv6 ND is currently defined for operation only on links that support a single broadcast domain. As such, many of the ND primitives rely on multicast to discover and communicate with neighboring nodes on the same link. For this reason, IPv6 ND does not map well to smart objects connected by a wireless network that spans multiple hops. One option involves emulating a single broadcast domain over the entire wireless network, allowing support for IPv6 ND as specified today. However, for networks that reach any reasonable scale, a simple multicast is prohibitively expensive as the message must be delivered to all nodes within the network. An alternative solution is to concentrate ND operations at routers that serve as egress points for the network. By relying on egress routers, ND can now utilize unicast communication which is much more efficient that multicast in wireless networks. The use of edge routers to support ND is currently being specified within the IETF.

Extremely small code size and low RAM usage

All three implementations target platforms that typically have less than 100 kilobytes of ROM and 10 kilobytes of RAM. Typical microcontrollers targeted by these stacks are displayed in Table 1.

Manufacturer	Device	ROM (kilobytes)	RAM (kilobytes)
Atmel	Atmega1281	128	8
Atmel	Atemga128P	128	16

[6] 6LoWPAN: Incorporating 802.15.4 into the IP architecture. IPSO White Paper #3.

Texas Instruments	MSP430F1611	48	10
Texas Instruments	MSP430F2618	116	8

Table 1. Typical MCUs targeted by IPv6 implementations for smart objects

Memory Requirements

An IPv6 Ready protocol stack can be implemented in approximately 11.5 kilobytes of ROM and 1.8 kilobytes of RAM. To support the 1280-byte minimum MTU, a single packet buffer requires 1280 bytes of RAM. Of the remaining data structures, the neighbor cache requires 35 bytes per neighbor, the prefix list requires 23 bytes per prefix, the router list requires 7 bytes per router, and the interface address list requires approximately 100 bytes. Table 2 provides a breakdown of the typical memory requirements for individual components. (Fragmentation and per-neighbor buffering are not included). Neighbor Discovery consumes the largest portion of the complexity.

Function	ROM	RAM
ND Input/Output	4800	20
ND structures	2128	238
Network interface management	1348	118
Stateless address autoconfiguration	372	16
IPv6 (headers processing, etc)	1434	44
Packet buffer (Ethernet case)	0	1296
ICMPv6	1406	16
Total	11488	1748

Table 2: Typical code and memory requirements for a complete IPv6 protocol stack.

Together with a complete run-time (timers, scheduler, etc.) as well as RFC-compliant UDP and TCP protocols above, an OS that provides a complete IPv6 network stack can be implemented within 35 kilobytes of ROM and 3 kilobytes of RAM. Complete IPv6-based applications fit comfortably within a microcontroller providing only 64 kilobytes of ROM and 4 to 8 kilobytes of RAM.

Energy Requirements

Battery-powered smart objects are constrained not only in memory, but in energy as well. With appropriate link-layer mechanisms and experimental modifications to IPv6 (e.g. compression and ND optimizations), it is possible to implement an IPv6 network stack that consumes very little energy. In production deployments, an IPv6/802.15.4 network can operate at sub-1% duty-cycles (including forwarding nodes), with very low per-hop latency and high reliability under realistic work-loads.

Table 3 presents network statistics from an IPv6-based home-monitoring application. The application consists of 15 nodes deployed in refrigerators, solar power inverters, outdoors, and indoors in or near the intended sense point. Nodes periodically report environmental data (e.g., temperature and humidity) as well as a variety of sensors attached to each node. Application traffic was about one datagram per minute per node, each datagram nearly

filling a full 802.15.4 frame. The routing topology consisted of 7 nodes within 1-hop of the egress router, the remaining half being 2 to 3 hops away.

Application Data Rate	1 UDP datagram/minute
Average Radio Duty Cycle	0.65%
Expected Per-Hop Latency	0.125 seconds
Datagram Delivery Rates	99.98%

Table 3: Network statistics for a production home-monitoring application.

Conclusion

This work demonstrates that even the most memory constrained devices can be IPv6 Ready. We have shown that it is also possible to implement an IPv6-based network on energy constrained devices and are currently specifying optimizations (e.g. IPv6 header compression and IPv6 ND modifications) to reduce the energy requirements of supporting a complete IPv6 network stack. These efforts provide a framework that will enable further advancement of IPv6 support for smart objects. For example, the IETF is currently working on routing algorithms for low-power wireless devices (ROLL Working Group). In addition to routing, support for end-to-end IPv6-based security mechanisms (e.g. IPsec, SSL, or TLS) still require some evaluation.

RFC References

[1] Conta, S. Deering, and M. Gupta. Internet Control Message Protocol (ICMPv6) for the Internet Protocol Version 6 (IPv6) Specification. RFC 4443, IETF, Mar. 2006.

[2] S. Deering, W. Fenner, and B. Haberman. Multicast Listener Discovery (MLD) for IPv6. RFC 2710, IETF, Oct. 1999.

[3] S. Deering and R. Hinden. Internet Protocol, Version 6 (IPv6) Specification. RFC 2460, IETF, Dec. 1998.

[4] R. Draves. Default Address Selection for Internet Protocol version 6 (IPv6). RFC 3484, IETF, Feb. 2003.

[5] R. Hinden and S. Deering. IP Version 6 Addressing Architecture. RFC 4291, IETF, Feb. 2006.

[6] J. Loughney. IPv6 Node Requirements. RFC 4294, IETF, Apr. 2006.

[7] G. Montenegro, N. Kushalnagar, J. Hui, and D. Culler. Transmission of IPv6 Packets over IEEE 802.15.4 Networks. RFC 4944, IETF, Sept. 2007.

[8] T. Narten, E. Nordmark, W. Simpson, and H. Soliman. Neighbor Discovery for IP version 6 (IPv6). RFC 4861, IETF, Sept. 2007

[9] S. Thomson, T. Narten, and T. Jinmei. IPv6 Stateless Address Autoconfiguration. RFC 4862, IETF, Sept. 2007.

[10] R. Vida and L. Costa. Multicast Listener Discovery Version 2 (MLDv2) for IPv6. RFC 3810, IETF, June 2004.