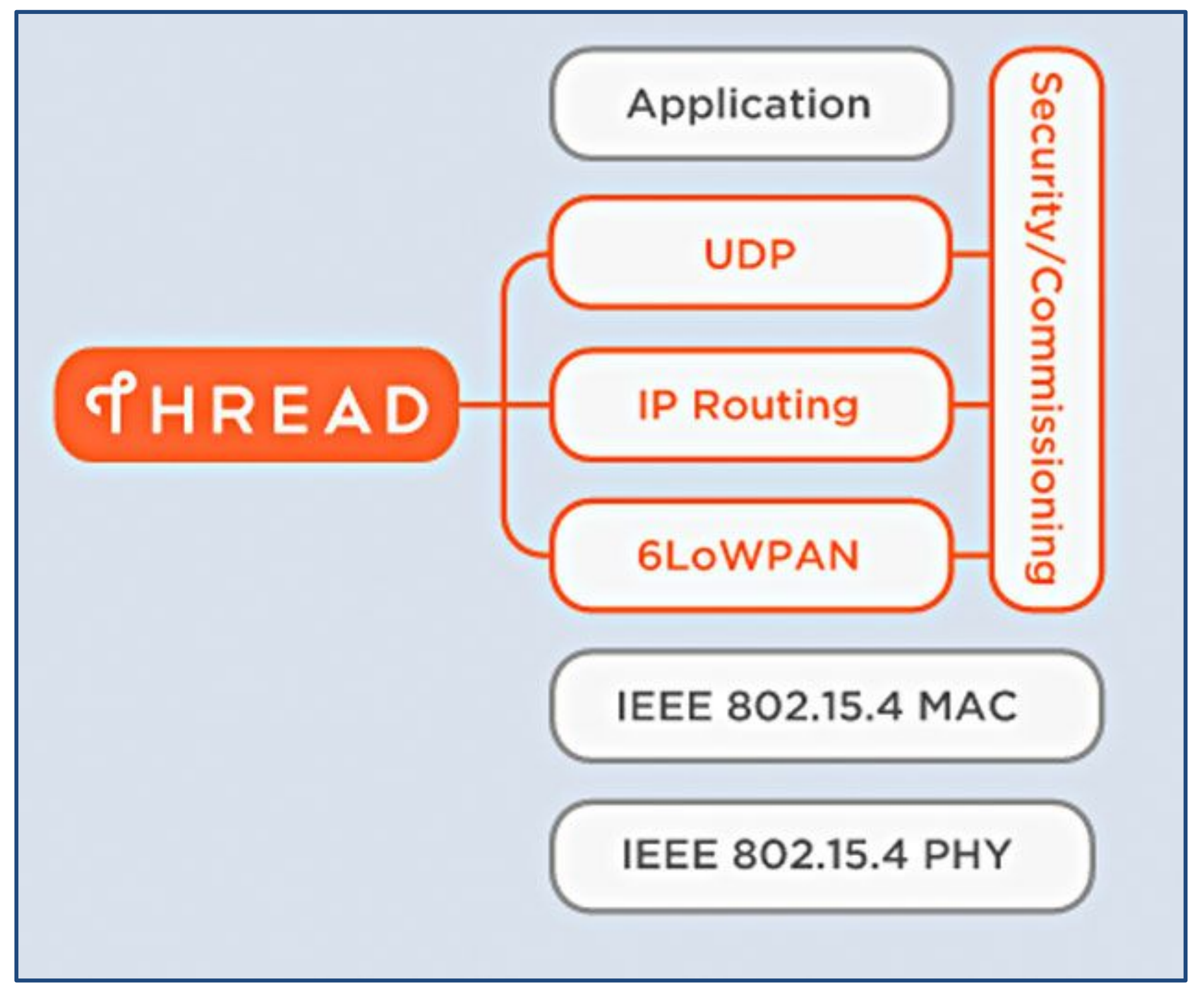
Bringing IP Networking to the Internet of Things

Paul Crews, Mateo Garcia, Hubert Teo
Professor Philip Levis, CURIS 2017

Motivation

The increasing prevalence of internet-connected embedded devices represents a significant security challenge. Hardware and resource constraints inhibit standard security mechanisms, while internet connectivity further exposes these devices. To address these challenges, the Tock operating system is an attempt to write a secure embedded OS.

Our work this summer focused on adding IP networking to Tock. Specifically, we implemented the IPv6 LoW power Personal Area Network (6LoWPAN) protocol and the IEEE 802.15.4 link/physical layer. The IEEE 802.15.4 protocol is a popular link/physical layer used for a number of low-power embedded systems, and 6LoWPAN is a specification that enables IPv6 packets to be sent over IEEE 802.15.4 links. Future work will implement the Thread protocol from Nest labs on top of 6LoWPAN and IEEE 802.15.4, allowing for full interoperability between Tock and other IoT devices.



Our Contribution

Both 6LoWPAN and IEEE 802.15.4 protocols require significant handling and manipulation of untrusted network-provided data. Although there exist several other implementations of these protocols, they are written in unsafe languages, and are prone to vulnerabilities. By implementing both in Rust, we can enforce stronger guarantees about the safety and security of our implementations, while remaining efficient and functional on low-power embedded systems.

6LoWPAN Overview

6LoWPAN is a specification which enables IPv6 packets to be sent over low-power wireless networks. The specification defines a compression algorithm and fragmentation process, which allows for large IPv6 packets to be sent over links with small MTUs (such as IEEE 802.15.4).

6LoWPAN Compression

The 6LoWPAN compression scheme can substantially reduce the size of IPv6 packet headers and extension headers. With IPv6 packets, these headers represent a significant portion of the packet size, and often contain redundant or compressible information. This increases the viability of sending IPv6 packets over low power links with small MTUs.

6LoWPAN Fragmentation

Although the 6LoWPAN compression algorithm can reduce the size of the IPv6 headers, sometimes upper-level protocols need to send IPv6 packets with large payloads. In order to send such packets over IEEE 802.15.4 links, 6LoWPAN defines a link-by-link fragmentation and reassembly procedure.

Challenges

Rust and Embedded Resource Constraints: The Rust type system ensures that there is at most a single mutable reference or multiple immutable references to any particular object. To allow other layers access to an object, we must pass references around. However, this becomes complicated as we consider who “owns” the buffer. Copying avoids many issues, but uses unnecessary resources, and with the lack of dynamic memory allocation, any additional buffers must be allocated at compile time.
Rust, Compression, and Fragmentation: Since pointer arithmetic and arbitrary type casting are unsafe, performing efficient, in-place compression and fragmentation becomes challenging. Initially, we performed operations solely on byte arrays, but then developed a more idiomatic encode/decode stream-like utility using macros.
Rust and Variable Sized Headers: For both 6LoWPAN and IEEE 802.15.4, there is substantial variability in overall header size set by various options. These interdependencies and lack of dynamic memory or scatter-gather pointers means that additional buffers are largely unavoidable. For example, to conditionally prepend or postpend bytes, we need to copy the payload from upper layers to different places, and for different sizes. This makes it difficult to determine ahead of time where and how large the payload will be.
Rust Lifetimes: Since asynchronous callbacks provide no guarentees about when particular functions will execute or complete, it is impossible to reason about certain object lifetimes. To solve this, a large number of objects in Tock have the static lifetime, ensuring that they will live for the lifetime of the program.

Related Work

- 6LoWPAN specification (RFC 4944, RFC 6282)
- RTOS and Linux kernel 6LoWPAN implementations
- OpenThread reference implementation
- Zigbee networking protocol